Nama : Muhamad Dimas Stiyawan

NPM : 22312087

Environtment Setup

1. Menginstall module nextjs

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable
it, run 'Import-Module PSReadLine'.

PS D:\Pemrograman Web 2\ecommerce_VisionVortex> npx create-next-app@latest
Need to install the following packages:
create-next-app@15.1.3
Ok to proceed? (y) y

√ What is your project named? ... ecommerce-visionvortex
√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like your code inside a `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to use Turbopack for `next dev`? ... No / Yes
√ Would you like to customize the import alias (`@/*` by default)? ... No / Yes
Creating a new Next.js app in D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex.
```

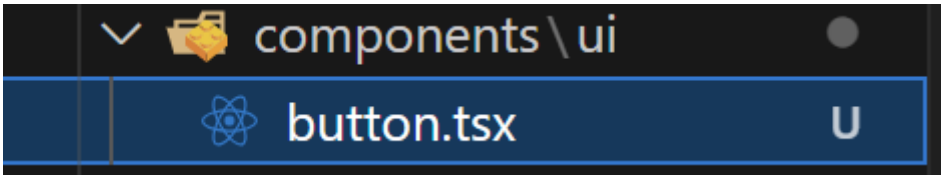
2. menginstall shadcn ui

```
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex> npx shadcn@latest init
 ✓ Preflight checks.
 ✓ Verifying framework. Found Next.js.
 ✓ Validating Tailwind CSS.
 ✓ Validating import alias.
 √ Which style would you like to use? » Default
 √ Which color would you like to use as the base color? » Slate
 √ Would you like to use CSS variables for theming? ... no / yes
 ✓ Writing components.json.
 ✓ Checking registry.
 ✓ Updating tailwind.config.ts
 ✓ Updating app\globals.css
   Installing dependencies.
```

- Menginstall button ui

```
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex> npx shadcn@latest add button
✓ Checking registry.

PS D:\Pemrograman Web 2\ecommerce_VisionVortex> cd ecommerce-visionvortex
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex> npx shadcn@latest add button
✓ Checking registry.
  Installing dependencies.
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex> npx shadcn@latest add button
✓ Checking registry.
  Installing dependencies.
  Installing dependencies.
```

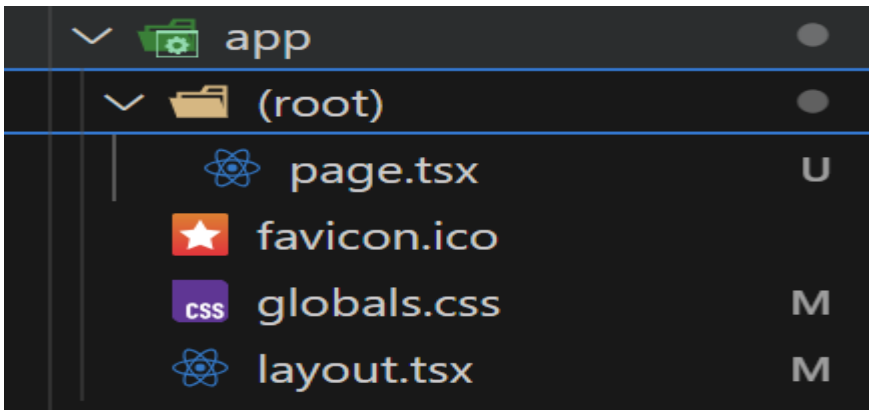
- jika sudah akan muncul folder components yg berisi file button.tsx



3. Memanggil button yang telah diinstal pada halaman page.tsx

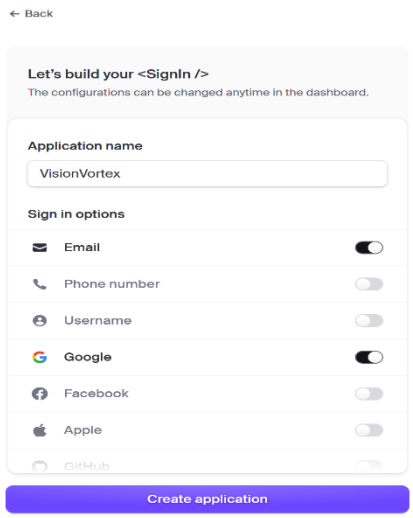
```
ecommerce-visionvortex > app > page.tsx > ...
1  import { Button } from "@/components/ui/button";
2  import Image from "next/image";
3
4  export default function Home() {
5    return (
6      <div className="p-4">
7          <Button>Click Me</Button>
8      </div>
9
10   );
11 }
```

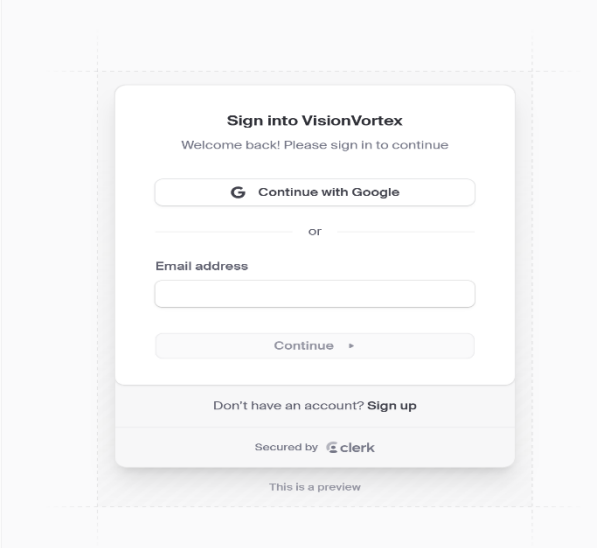
4. membuat folder (root) pada folder app dan memindahkan file page.tsx ke dalam folder (root)



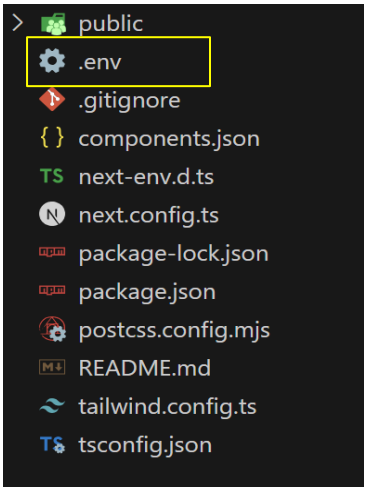
Clerk Authentication (Admin)

1. buat akun clerk di https:/clerk.com , kemudian pilih authentikasi email dan google

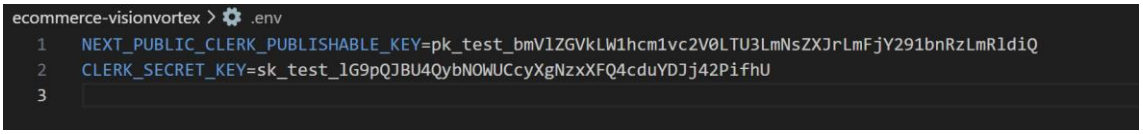




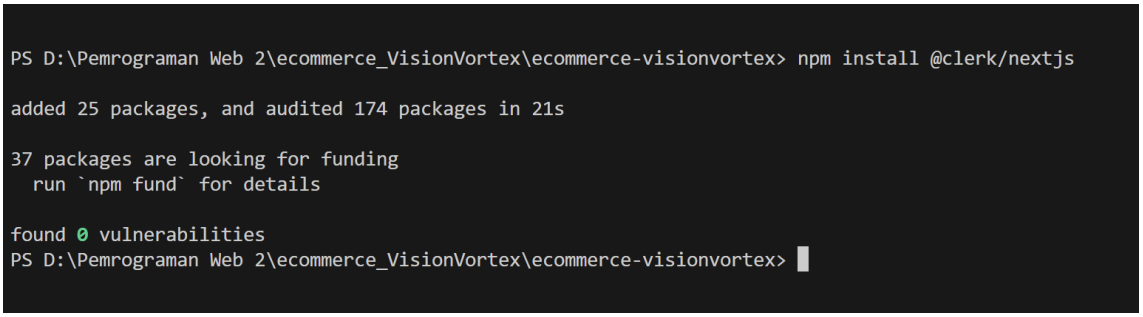
2. buat file .env pada project



3. memasukkan API Keys clerk ke dalam file .env



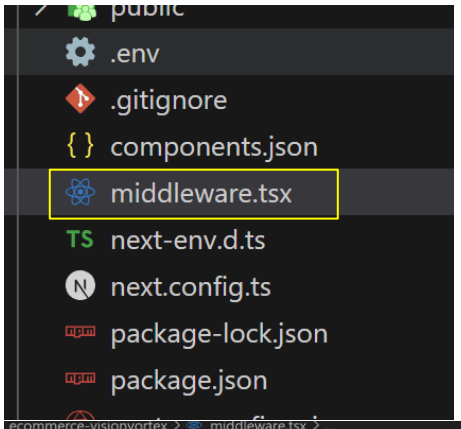
4. menginstal clerk pada project



5. mengimport clerkProvider pada file layout.tsx

```tsx
import type { Metadata } from "next";
import { Geist, Geist_Mono } from "next/font/google";
import { ClerkProvider } from "@clerk/nextjs";
import "./globals.css";

const geistSans = Geist({ ···
});

const geistMono = Geist_Mono({ ···
});

export const metadata: Metadata = {
  title: "Admin Dashboard",
  description: "Admin Dashboard",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <ClerkProvider>
      <html lang="en">
        <body
          className={`${geistSans.variable} ${geistMono.variable} antialiased`}
        >
          {children}
        </body>
      </html>
    </ClerkProvider>
  );
}
```

6. membuat file middleware.tsx dan menambahkan program config clerk authentication

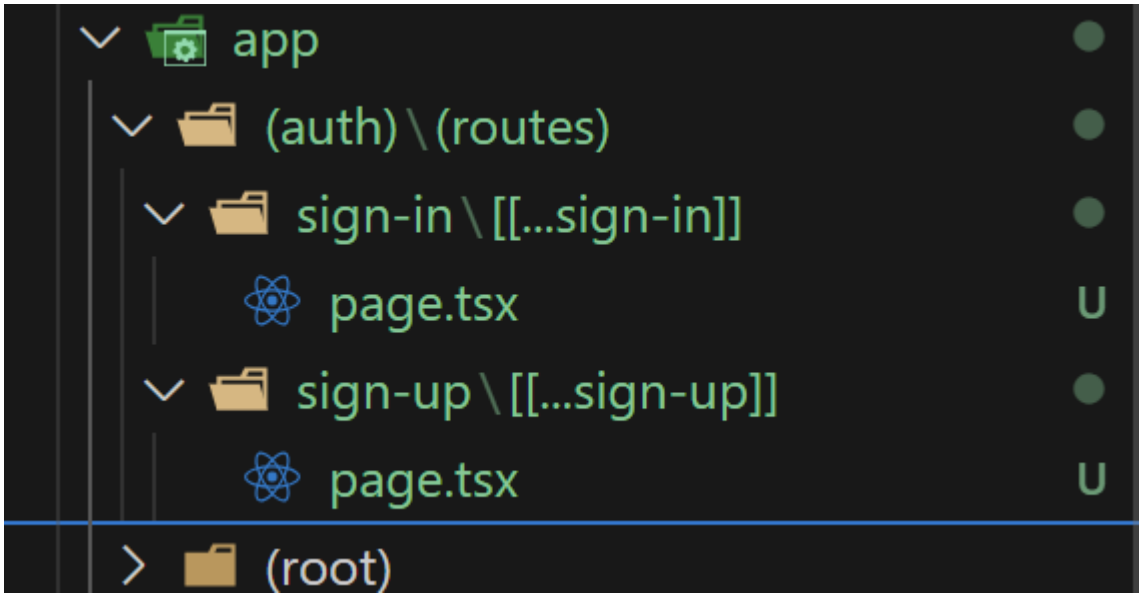


ecommerce-visionvortex > middleware.tsx > ...

```tsx
import { clerkMiddleware } from "@clerk/nextjs/server";

export default clerkMiddleware();

export const config = {
  matcher: [
    // Skip Next.js internals and all static files, unless found in search params
    '/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)',
    // Always run for API routes
    '/(api|trpc)(.*)',
  ],
};
```

SIGN-UP dan SIGN-IN

1. membuat folder (auth) kemudian didalam folder (auth) terdapat folder (routes) dan didalam folder routes membuat folder sign-in kemudian dalam folder sign-in terdapat folder [[..sign-in]] kemudian membuat file page.tsx dalam folder [[...sign-in]] begitu juga dengan sign-up seperti digambar.



2. memasukkan program sign-in pada file page.tsx dalam folder sign-in

ecommerce-visionvortex > app > (auth) > (routes) > sign-in > [[...sign-in]] > page.tsx > Page

```tsx
import { SignIn } from '@clerk/nextjs'

export default function Page() {
  return <SignIn />
}
```

3. memasukkan program sign-up pada file page.tsx dalam folder sign-up

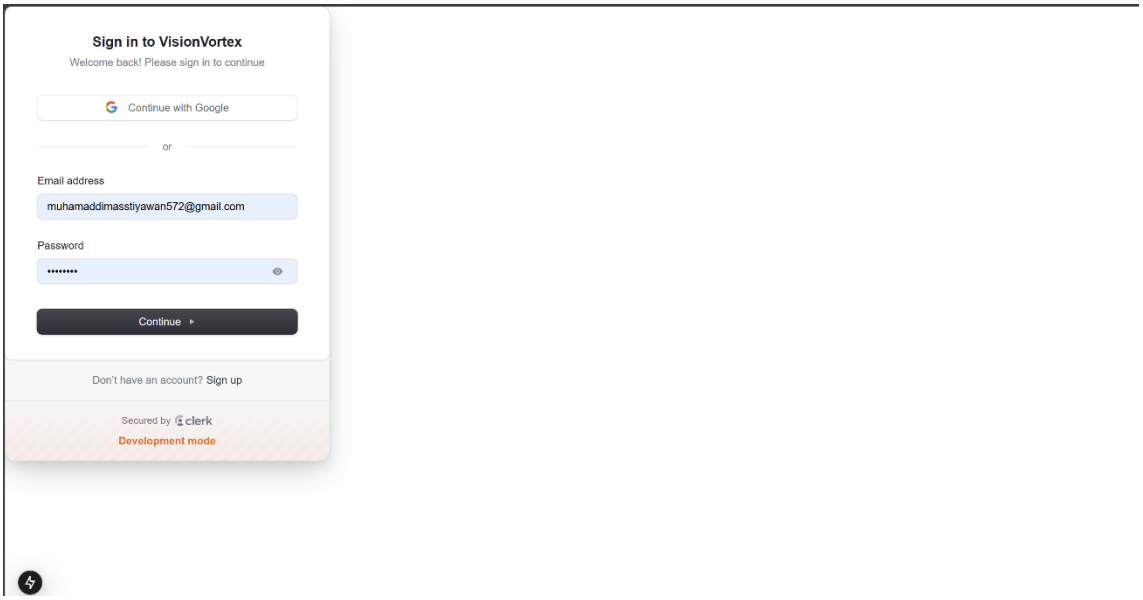
ecommerce-visionvortex > app > (auth) > (routes) > sign-up > [[...sign-up]] > page.tsx > Page

```tsx
import { SignUp } from '@clerk/nextjs'

export default function Page() {
  return <SignUp />
}
```

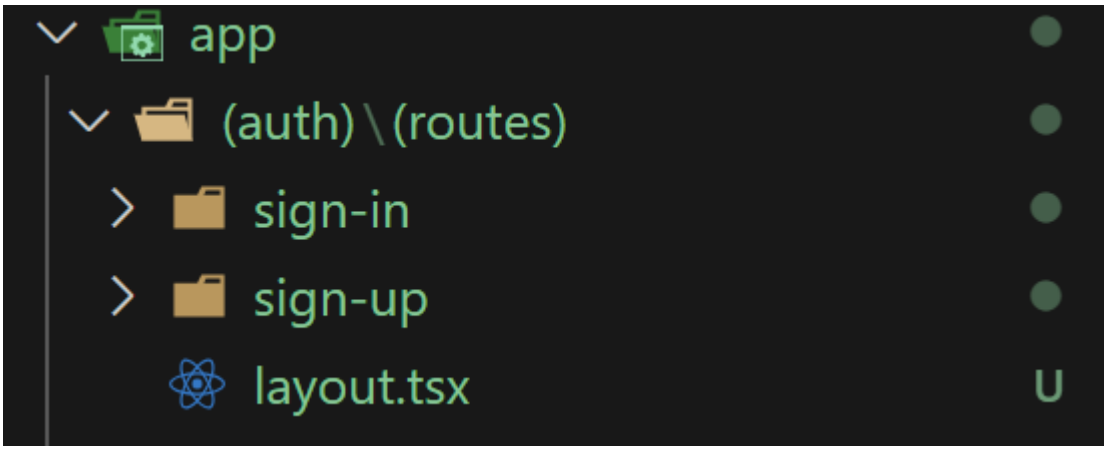
4. menambahkan program pada file .env

```
NEXT_PUBLIC_CLERK_SIGN_IN_URL=/sign-in
NEXT_PUBLIC_CLERK_SIGN_UP_URL=/sign-up
```

5. tampilan setelah program dijalankan



6. membuat file layout.tsx pada folder (routes)

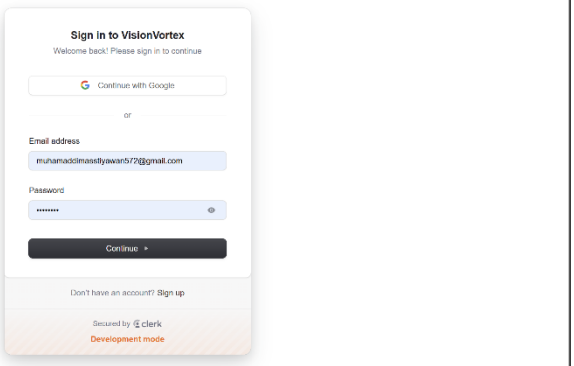


7. menambahkan program pada file layout.tsx untuk mengubah layout sign-in dan sign-up

ecommerce-visionvortex > app > (auth) > (routes) > layout.tsx > AuthLayout

```tsx
export default  function AuthLayout({
    children
}: {
    children: React.ReactNode
}) {
    return (
        <div className="flex items-center justify-center min-h-screen ">{children}</div>
    )
}
```

8. tampilan setelah program dijalankan



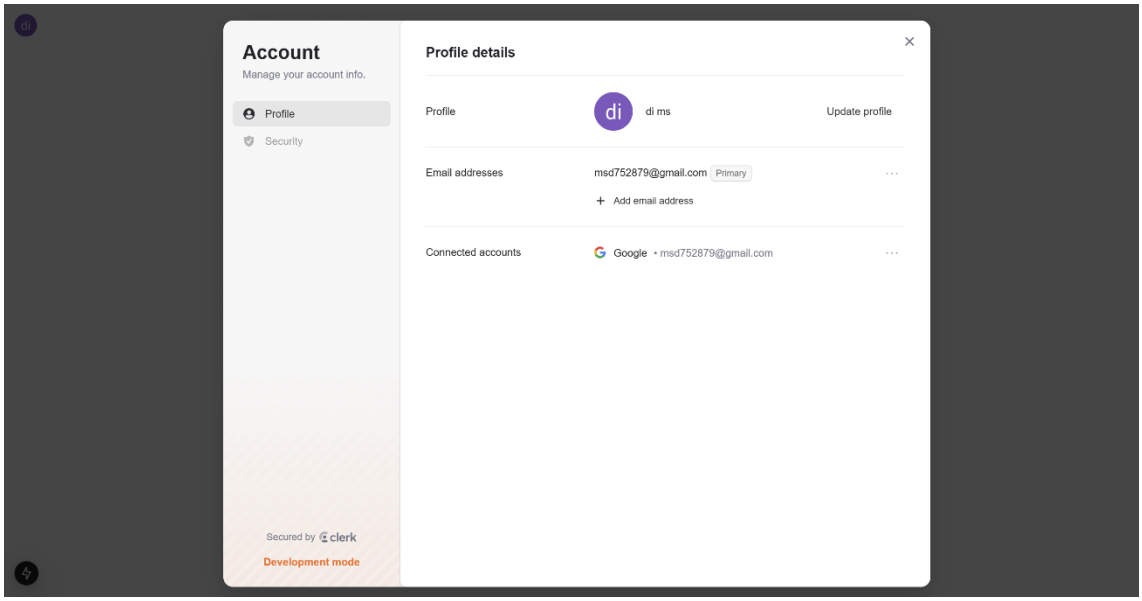
9. tampilan setelah sign-in menggunakan google

10. mengubah program pada file page.tsx dalam folder (root) untuk memanggil profile user

ecommerce-visionvortex > app > (root) > page.tsx > ...

```tsx
import { Button } from "@/components/ui/button";
import { UserButton } from "@clerk/nextjs";
import Image from "next/image";

const setUpPage = () => {
  return (
    <div className="p-4">
        <UserButton></UserButton>
    </div>

  );
}

export default setUpPage
```

11. tampilan setelah program dijalankan

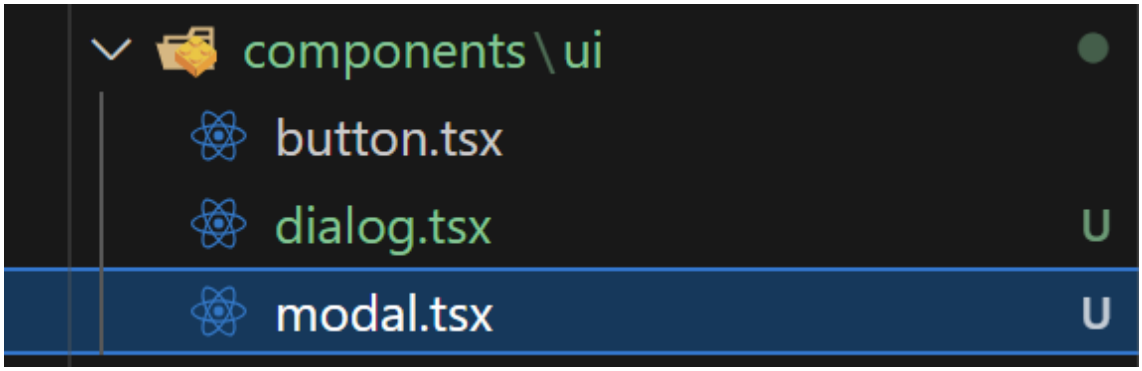


Modal Components

1. menginstal ui dialog dari Shadcnui.com

```
npx shadcn@latest add dialog
```

2. membuat file modal.tsx pada folder components/ui



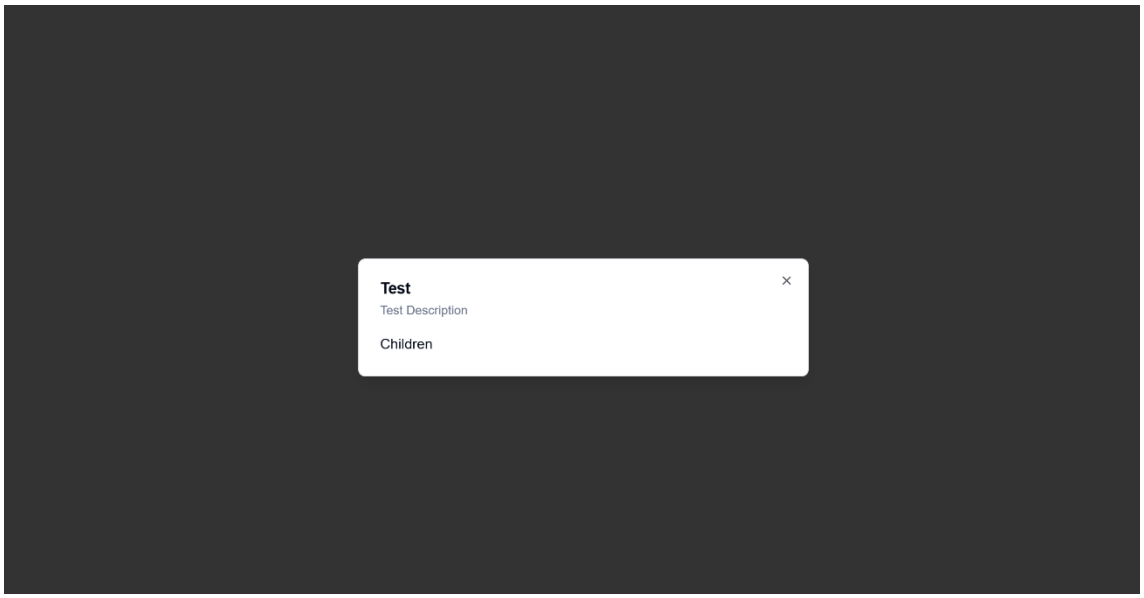
3. membuat program untum modal dialog pada file modal.tsx

```tsx
"use client"

// Mengimpor komponen yang dibutuhkan dari library U
import { Dialog, DialogContent, DialogDescription, DialogHeader, DialogTitle } from "@/components/ui/dialog";

// Interface untuk mendefinisikan properti yang akan diterima oleh komponen Modal
interface ModalProps{
    title: string;
    description: string;
    isOpen: boolean;
    onClose: () => void;
    children?: React.ReactNode;
}
// Deklarasi komponen Modal menggunakan React.FC dengan properti yang sesuai dengan interface ModalProps
export const Modal: React.FC<ModalProps> =({
    title,
    description,
    isOpen,
    onClose,
    children
}) => {
    // Fungsi untuk menangani perubahan status dialog (terbuka atau tertutup)
    const onChange =(open: boolean) => {
        if (!open){
            onClose();
        }
    };
       // Komponen yang dirender
    return(
        <Dialog open={isOpen} onOpenChange={onChange}>
            <DialogContent>
                <DialogHeader>
                    <DialogTitle>{title}</DialogTitle>
                    <DialogDescription>
                        {description}
                    </DialogDescription>
                </DialogHeader>
                <div>
                    {children}
                </div>
            </DialogContent>
        </Dialog>
    );
};
```

4. mengubah program pada file page.tsx dalam folder (root)

```tsx
"use client"
// Mengimpor komponen `Modal` dari path `@/components/ui/modal`.
import { Modal } from "@/components/ui/modal";


// Fungsi `setUpPage` adalah komponen fungsional React yang digunakan untuk merender halaman.
const setUpPage = () => {
  return (
    <div className="p-4">
       <Modal title="Test" description="Test Description" isOpen onClose={() => {}}>
            Children
        </Modal>
    </div>

  );
}

// Mengekspor `setUpPage` sebagai default, sehingga komponen ini dapat digunakan di file lain.
export default setUpPage

```

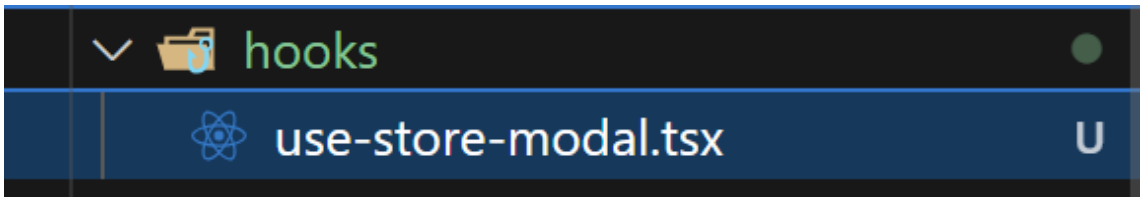
tampilan ketika program dijalankan



5. menginstall libraray zustand untuk mengontrol apakah modal kita terbuka atau tidak.

```
npm install zustand
```

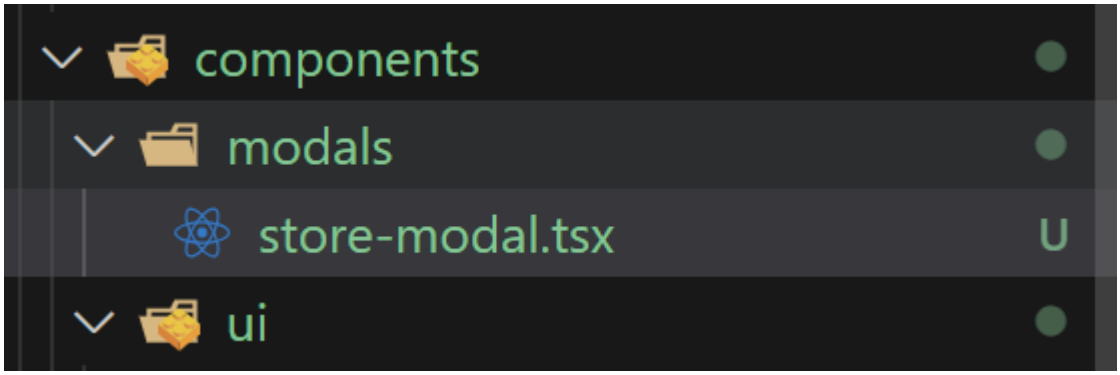
6. membuat folder hooks dalam project kemudian membuat file use-store-modal.tsx dalam folder hooks



7. menambahkan program pada file use-store-modal.tsx berfungsi untuk mengelola status modal secara global menggunakan pustaka Zustand.

```tsx
// Mengimpor fungsi `create` dari pustaka Zustand untuk membuat store global pada aplikasi React.
import { create } from "zustand";

// Mendefinisikan tipe data untuk store yang mengelola status modal.
interface useStoreModalStore{
    isOpen: boolean;
    onOpen: () => void;
    onClose: () => void;
};

// Membuat store dengan Zustand dan mendefinisikan status dan fungsi pengelolaan modal.
export  const useStoreModal = create<useStoreModalStore>((set) => ({
    isOpen: false,
    onOpen: () => set({isOpen: true}),
    onClose: () => set({isOpen: false}),
}));
```

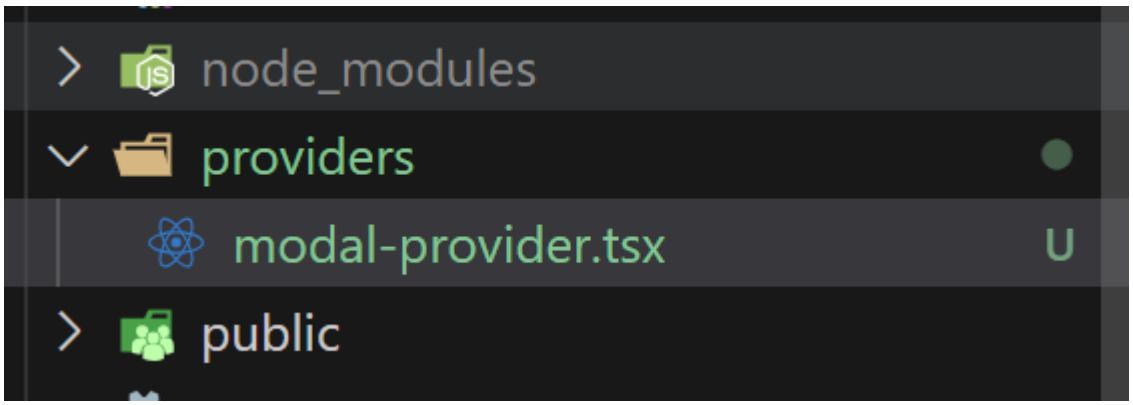
8. membuat folder modals dalam folder components, kemudian membuat file store-modal.tsx dalam folder modals



9. menambahkan program pada file store-modal.tsx untuk menampilkan sebuah modal bernama `StoreModal`, yang digunakan untuk menambahkan toko (store) baru. Modal ini dikelola menggunakan Zustand store **melalui hook `useStoreModal`.**

```tsx
"use client"

// Mengimpor hook `useStoreModal` yang digunakan untuk mengakses state global modal dari Zustand.
import { useStoreModal } from "@/hooks/use-store-modal"
// Mengimpor komponen `Modal` dari path lokal. Komponen ini bertugas untuk menampilkan dialog/modal.
import { Modal } from "../ui/modal";

// Mendefinisikan komponen fungsional React bernama `StoreModal`.
 export const StoreModal = () => {
    // Mengakses state dan fungsi dari store modal menggunakan hook `useStoreModal`.
    const storeModal = useStoreModal();

    return(
        <Modal
        title="create store"
        description="Add a new store to manage products and categories"
        isOpen={storeModal.isOpen}
        onClose={storeModal.onClose}
        >
            future Create Sore Form
            {/* Konten modal yang akan muncul di dalamnya. Saat ini berupa placeholder teks,
                tetapi akan digantikan dengan form atau elemen lain di masa mendatang. */}
        </Modal>
    )
}
```

10. membuat folder providers kemudian didalmnya terdapat file modal-provider.tsx



11. menambahkan program pada file modal-provider.tsx yang berfungsi sebagai provider untuk modal. Komponen ini memastikan bahwa modal hanya dirender di sisi klien (bukan di server) dan memanfaatkan modal `StoreModal` sebagai salah satu contoh modal yang dikelola.

```tsx
"use client"
// Mengimpor komponen `StoreModal` yang bertugas untuk menampilkan modal untuk menambahkan toko.
import { StoreModal } from "@/components/modals/store-modal";
// Mengimpor hook `useEffect` dan `useState` dari React untuk mengelola efek samping dan state lokal.
import { useEffect, useState } from "react"

// `ModalProvider` yang bertugas mengelola render modal di sisi klien.
export const ModalProvider = () =>{
    // State `isMounted` untuk mengecek apakah komponen sudah ter-mount.
    const [isMounted, setIsMounted] = useState(false);

    useEffect(() =>{
        // Mengubah `isMounted` menjadi `true` setelah komponen ter-mount di klien.
        setIsMounted(true)
    }, []);

    if(!isMounted){
        // Tidak merender komponen hingga ter-mount di klien.
        return null;
    }

    return(
        <>
        {/* Render `StoreModal` setelah komponen ter-mount. */}
            <StoreModal/>
        </>
    );
};
```

12. memanggil ModalProvider pada halaman layout.tsx

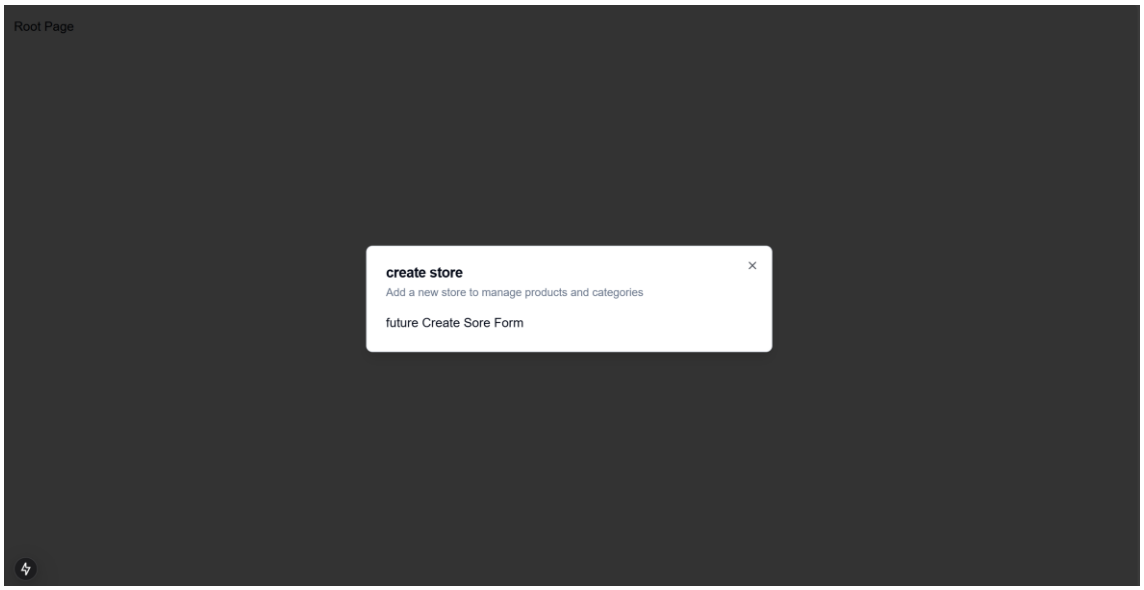
```tsx
import type { Metadata } from "next";
import { Geist, Geist_Mono } from "next/font/google";
import { ClerkProvider } from "@clerk/nextjs";

import { ModalProvider } from "@/providers/modal-provider";
import "./globals.css";


const geistSans = Geist({ ···
});

const geistMono = Geist_Mono({ ···
});

export const metadata: Metadata = {
  title: "Admin Dashboard",
  description: "Admin Dashboard",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <ClerkProvider>
    <html lang="en">
      <body className={`${geistSans.variable} ${geistMono.variable} antialiased`}>
        <ModalProvider/>
        {children}
      </body>
    </html>
    </ClerkProvider>
  );
}
```

13. Mengubah dan menambahkan program pada file page.tsx yang terdapat pada folder (root).

ecommerce-visionvortex > app > (root) > page.tsx > setUpPage

```tsx
"use client"

// Mengimpor hook `useStoreModal` untuk mengakses store modal.
import { useStoreModal } from "@/hooks/use-store-modal";

import { useEffect } from "react";


// Fungsi `setUpPage` adalah komponen fungsional React yang digunakan untuk merender halaman.
const setUpPage = () => {

  // Mengambil fungsi `onOpen` dan status `isOpen` dari store `useStoreModal`.
  const onOpen = useStoreModal((state) => state.onOpen);
  const isOpen = useStoreModal((state) => state.isOpen);

  // Menggunakan `useEffect` untuk membuka modal jika `isOpen` bernilai false.
  useEffect(() => {
    if(!isOpen){
      onOpen();
    }
  }, [isOpen,onOpen]);


  return (
    <div className="p-4">
      {/* Konten dari halaman, menampilkan teks "Root Page". */}
      Root Page
    </div>

  );
}

// Mengekspor `setUpPage` sebagai default, sehingga komponen ini dapat digunakan di file lain.
export default setUpPage
```

14. Tampilan ketika program dijalankan



BACKEND PRISMA, NEON DAN PostgreSQL

1. menginstal prisma

```
npm install -D prisma
```

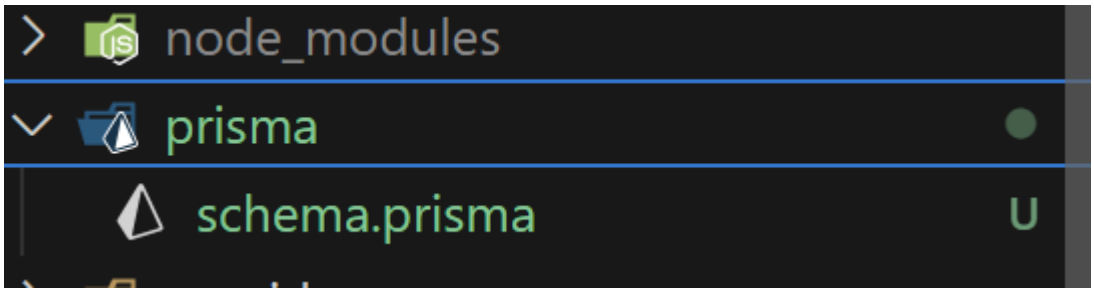
2. menginstal prisma client

```
npm install @prisma/client
```

3. menginstal prisma init

```
npx prisma init
```

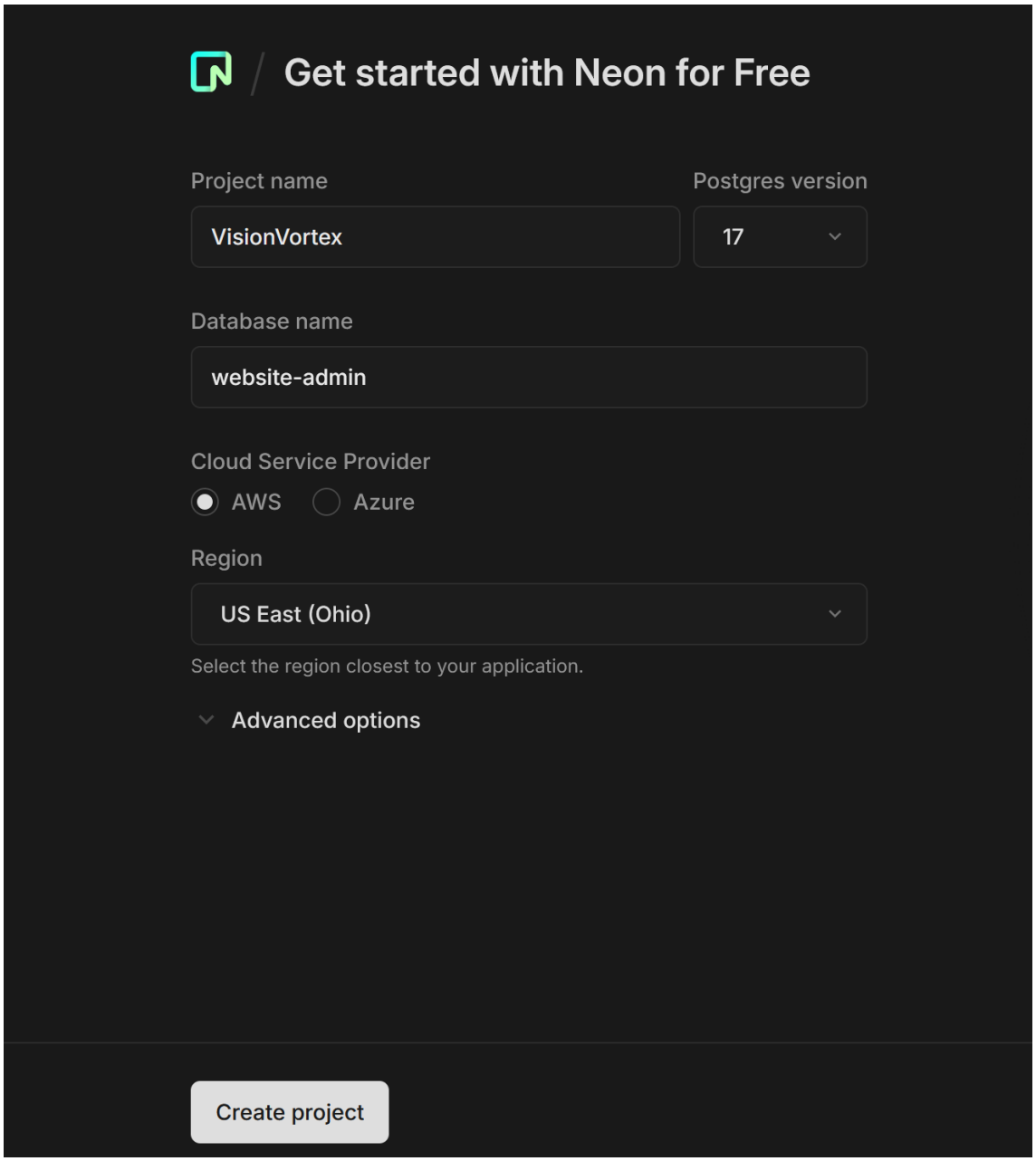
Setelah menginstal prisma init akan muncul folder baru yang bernama prisma



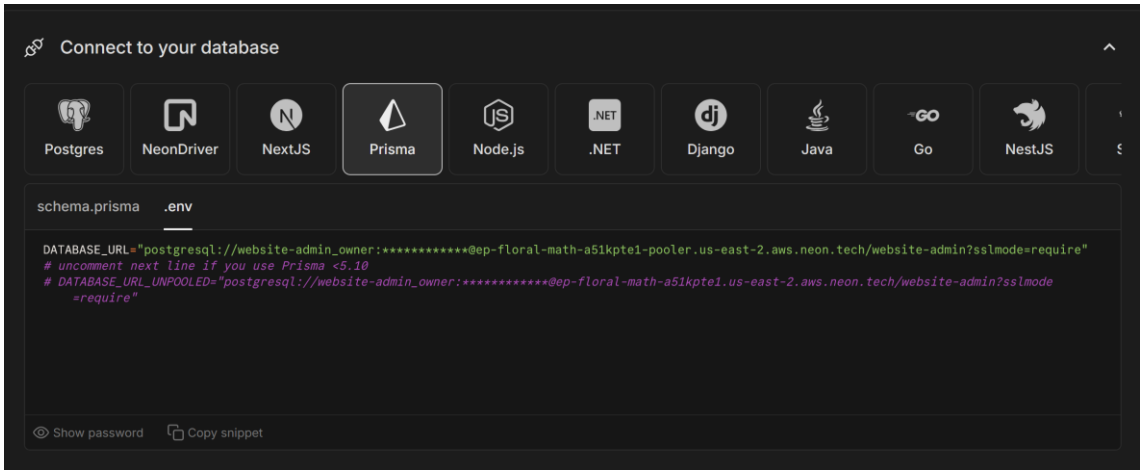
4. membuat file baru bernama db.ts pada folder lib

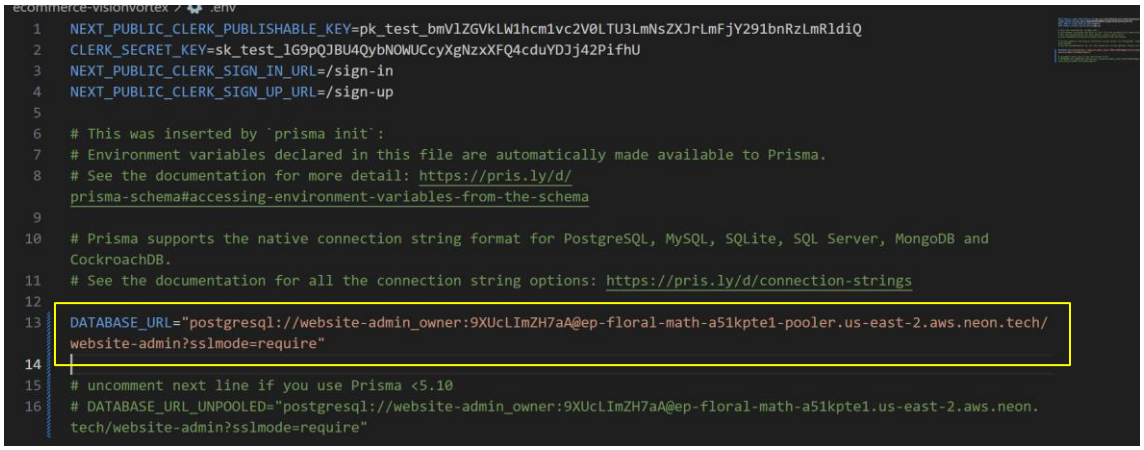
5. menambahkan program pada file db.ts untuk konfigurasi menggunakan Prisma Client

6. membuat akun dan database di neon.tech

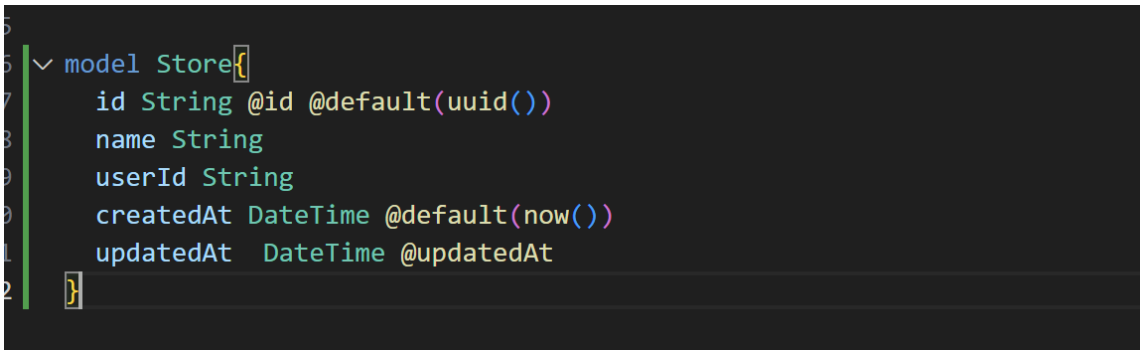


7. menyalin database url pada halaman neon.tech kemudian di tempelkan pada file .env

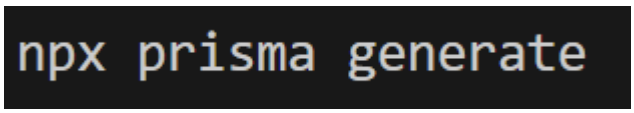




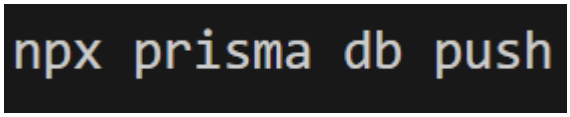
8. Membuat model database pada schema.prisma



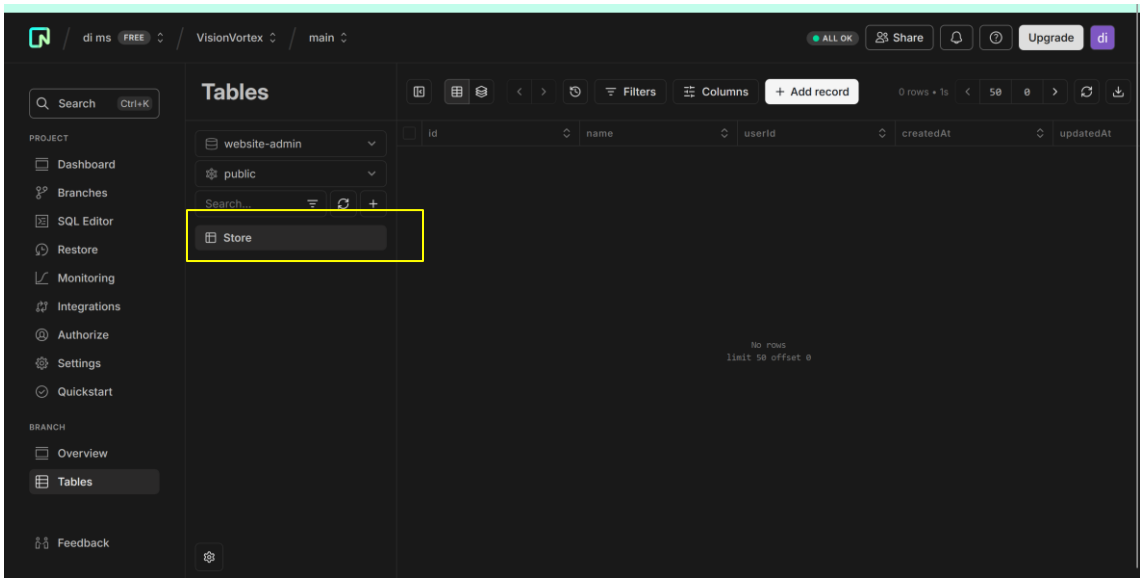
Lalu ketik npx prisma generate pada terminal



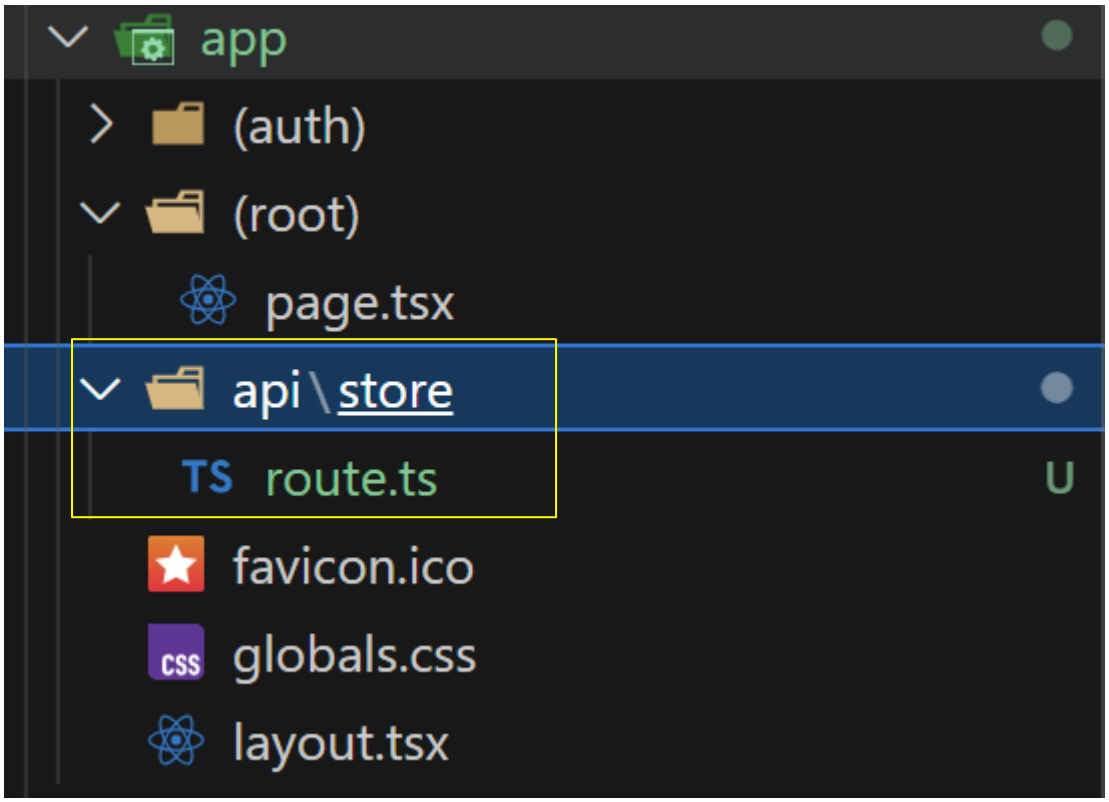
Push schema ke dalam database



Jika berhasil maka didalam web neon akan seperti ini



9. Membuat folder api kemudian didalam folder api tambahkan folder store, didalam folder store tambahkan file route.ts



10. menambahkan program pada file route.ts untuk memproses permintaan POST, dengan autentikasi melalui Clerk, yang merupakan library untuk autentikasi pengguna.

ecommerce-visionvortex > app > api > store > TS route.ts > POST

```ts
import { auth } from "@clerk/nextjs/server"
import { NextResponse } from "next/server"

export async function POST(req: Request){
    try{
        const { userId} = await auth()
        const body = await req.json();

        const {name} = body

    }catch (error){
        console.log("[STORES_POST]", error)
        return new NextResponse("Internal error", {status: 500})
    }
}
```

11.menambahkan program pada file route.ts untuk Membuat Entri baru pada tabel “store “

```ts
// Membuat entri baru pada tabel 'store' dalam database.
const store = await db.store.create({
    data: {
        name,
        userId
    },
});
```

12. mengisntall axios pada project. *Axios merupakan library opensource yang digunakan untuk request data melalui http*

```
npm install axios
```

13. Menambahkan program pada file store-modal.tsx

Menambahkan variabel dan fungsi loading dan setLoading pada bagian Komponen StoreModal

```ts
// Mendefinisikan komponen fungsional React bernama `StoreModal`.
export const StoreModal = () => {
    const [loading, setLoading] = useState(false)

    // Mengakses state dan fungsi dari store modal menggunakan hook `useStoreModal`.
    const storeModal = useStoreModal();
```

Menambahkan Program fungsi `onSubmit` menangani proses pengiriman data dari form ke server melalui API. Fungsi ini mengaktifkan status loading saat pengiriman data dimulai, mengirimkan data ke server menggunakan `axios`,

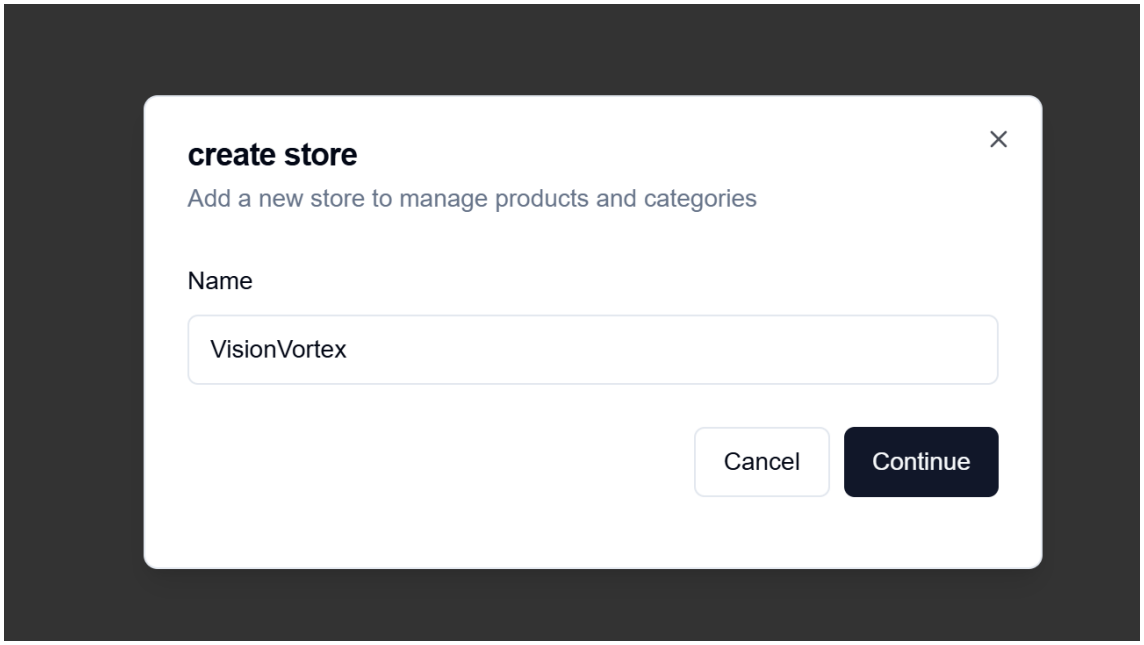
menampilkan respons yang diterima dari server jika berhasil, menangani kesalahan jika terjadi error, dan akhirnya mematikan status loading setelah proses selesai, baik berhasil maupun gagal.

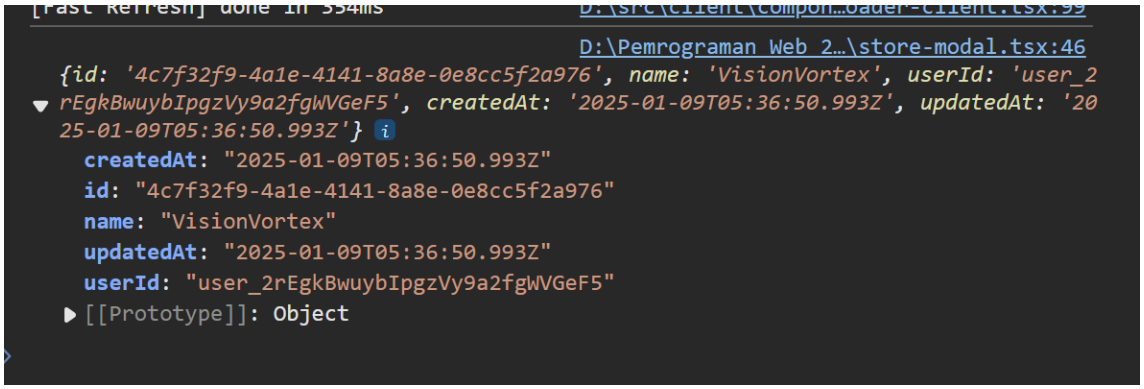
```
// Mendefinisikan komponen fungsional React bernama "StoreModal".
export const StoreModal = () => {
    const [loading, setLoading] = useState(false)

    // Mengakses state dan fungsi dari store modal menggunakan hook `useStoreModal`.
    const storeModal = useStoreModal();

    const form = useForm<z.infer<typeof formSchema>>({
        resolver: zodResolver(formSchema),
        defaultValues: {
            name: "",
        },
    });

    const onSubmit = async (values: z.infer<typeof formSchema>) => {
        try {
            setLoading(true)
            const response = await axios.post('/api/stores', values)
            console.log(response.data)
        } catch (error) {
            console.log(error)
        } finally{
            setLoading(false)
        }

    }
```

Menambhakan property disabled

```
                                <FormLabel>Name</FormLabel>
                                <FormControl>
                                    <Input
                                    disabled={loading}
                                    placeholder="E-Commerce"
                                    {...field}/>
                                </FormControl>
                                <FormMessage />
                            </FormItem>
                        )}
                    />
                    <div className="pt-6 space-x-2 flex items-center justify-end w-full">
                        <Button
                            disabled={loading}
                            variant="outline"
                            onClick={storeModal.onClose}>Cancel
                        </Button>
                        <Button disabled={loading} type="submit">Continue</Button>
                    </div>
                </form>
            </Form>
```

14. tampilan ketika program berhasil dijalnkan

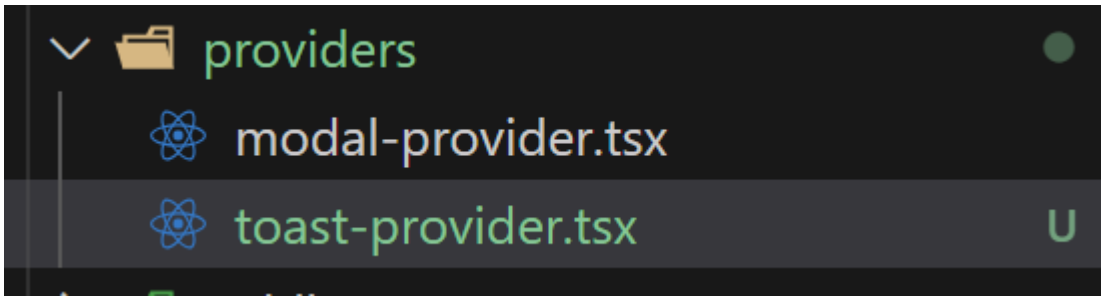




15. Menginstal react-hot-toast untuk menambahkan notifikasi toast ke aplikasi React dengan mudah dan intuitif

```
npm install react-hot-toast
```

16. menambahkan file toast-provider.tsx pada folder providers



Menambahkan program pada file toast-provider.tsx

```tsx
"use client"

import {Toaster} from "react-hot-toast"

export const ToasterProvider = () =>{
    return <Toaster/>;
};
```

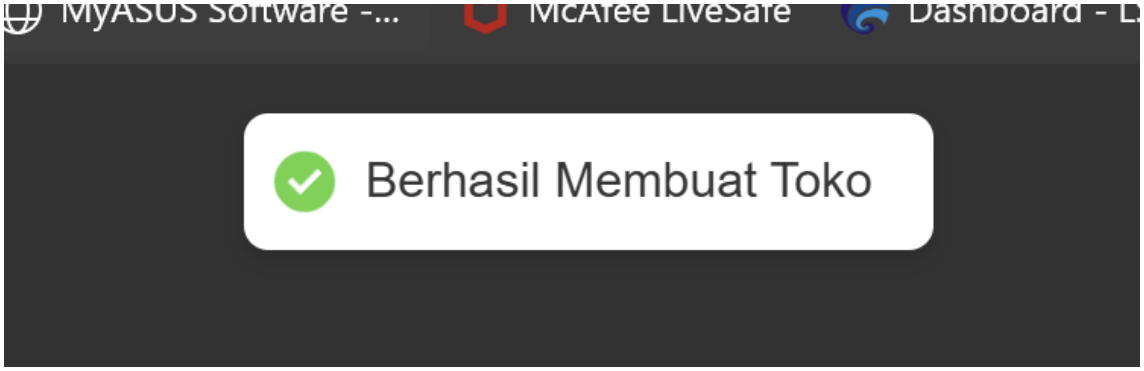
17. Memanggil ToastProvider pada file layout.tsx

```tsx
return (
  <Clerk (property) HTMLAttributes<T>.lang?: string | undefined
  <html lang="en">
    <body className={`${geistSans.variable} ${geistMono.variable} antialiased`}>
      <ToasterProvider/>
      <ModalProvider/>
      {children}
    </body>
  </html>
  </ClerkProvider>
);
}
```

18. Menambahkan program untuk menampilkan notifikasi berhasil dan tidak berhasil menggunakan toast pada file store-modal.tsx

```tsx
const onSubmit = async (values: z.infer<typeof formSchema>) => {
    try {
        setLoading(true)
        const response = await axios.post('/api/stores', values)
        console.log(response.data)
        toast.success("Berhasil Membuat Toko")
    } catch (error) {
        toast.error("Gagal Membuat Toko")
    } finally{
        setLoading(false)
    }

}
```

19.Jika berhasil Maka tampilannya akan seperti berikut jika berhasil menambhakn toko:



Data Table (Admin)

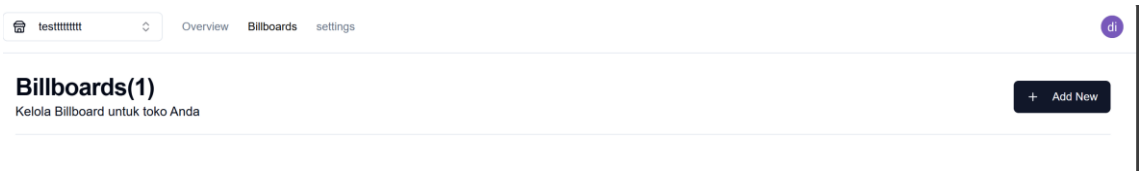
1. menambahkan program pada folder billboards file page.tsx

```tsx
import db from "@/lib/db";
import { BillboardClient } from "./components/client";

const BillboardsPage = async ({
    params
}: {
    params: {storeId: string}
}) => {
    const billboards= await db.billboard.findMany({
        where: {
            storeId: params.storeId
        },
        orderBy: {
            createdAt: 'desc'
        }
    })
    return (
        <div className="flex-col">
            <div className="flex-1 space-y-4 p-8 pt-6">
                <BillboardClient data={billboards}/>
            </div>
        </div>
    );
}

export default BillboardsPage;
```

2. menambahakan program pada file client.tsx untuk menampilkan banyak data yang di upload

```tsx
"use client"

import { Plus } from "lucide-react"
import { useParams, useRouter } from "next/navigation"

import { Button } from "@/components/ui/button"
import { Heading } from "@/components/ui/heading"
import { Separator } from "@/components/ui/separator"
import { Billboard } from "@prisma/client"

interface BillboardClientProps{
    data: Billboard[]
}

export const BillboardClient: React.FC<BillboardClientProps> = ({
    data
}) => {
    const router = useRouter();
    const params = useParams();


    return (
        <>
            <div className="flex items-center justify-between">
                <Heading
                    title={`Billboards(${data.length})`}
                    description="Kelola Billboard untuk toko Anda"
                />
                <Button onClick={() => router.push(`/${params.storeId}/billboards/new`)}>
                    <Plus className="mr-2 h-4 w-4"/>
                    Add New
                </Button>
            </div>
            <Separator />
        </>
    )
}
```

Tampilan ketika program dijalankan



3. menginstall components data table dari shadcnui.com

```
> npx shadcn@latest add table
```

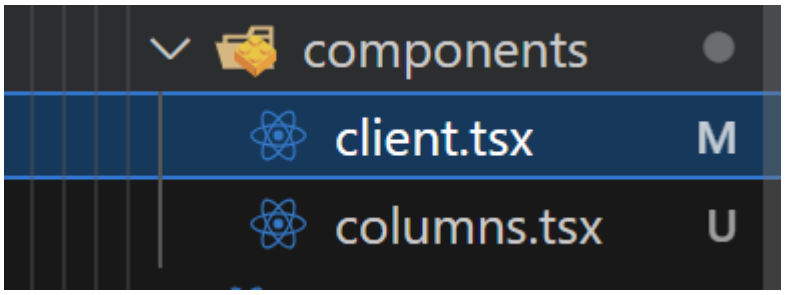
4. menginstal package tanstck

```
npm install @tanstack/react-table
```

5. menginstall package date

```
npm i date-fns
```

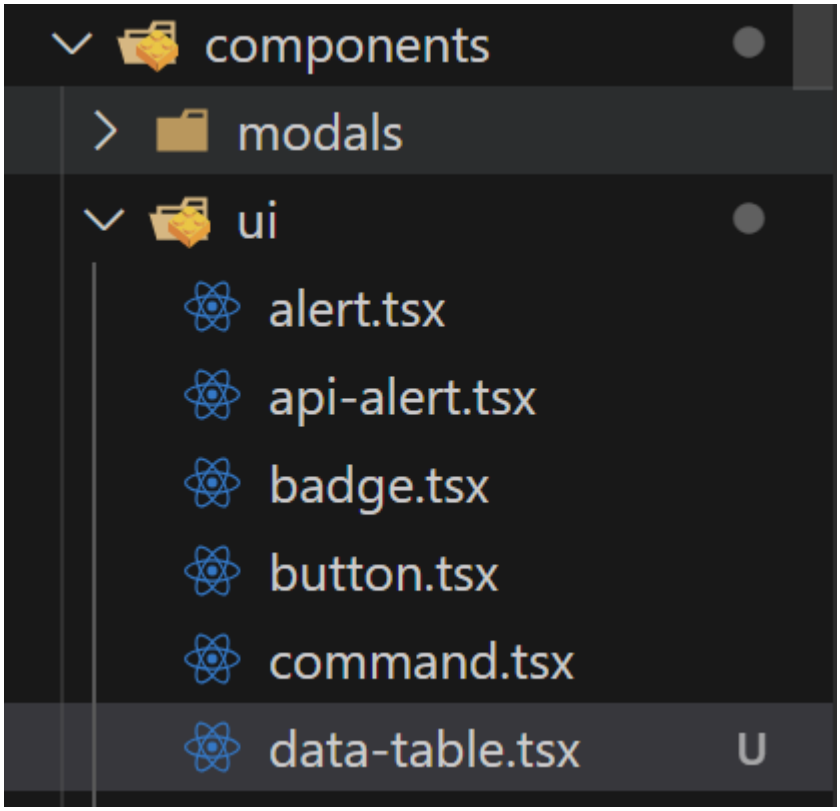
6. membuat file column.tsx pada folder components billboards



7. menambhakan program untuk membuat column pada file column.tsx

```tsx
"use client"

import { ColumnDef } from "@tanstack/react-table"

// This type is used to define the shape of our data.
// You can use a Zod schema here if you want.
export type BillboardColumn = {
  id: string
  label: string
  createdAt: string
}

export const columns: ColumnDef<BillboardColumn>[] = [
  {
    accessorKey: "label",
    header: "Label",
  },
  {
    accessorKey: "createdAt",
    header: "Date",
  },
]
```

8. membuat file data-table.tsx pada folder components/ui



9. menambhakn program pada file data-table.tsx

```tsx
"use client"

import {···

interface DataTableProps<TData, TValue> {
  columns: ColumnDef<TData, TValue>[]
  data: TData[]
}

export function DataTable<TData, TValue>({
  columns,
  data,
}: DataTableProps<TData, TValue>) {
  const table = useReactTable({
    data,
    columns,
    getCoreRowModel: getCoreRowModel(),
  })

  return (
    <div className="rounded-md border">
      <Table>
        <TableHeader>
          {table.getHeaderGroups().map((headerGroup) => (
            <TableRow key={headerGroup.id}>
              {headerGroup.headers.map((header) => {
                return (
                  <TableHead key={header.id}>
                    {header.isPlaceholder
                      ? null
                      : flexRender(
                          header.column.columnDef.header,
                          header.getContext()
                        )}
                  </TableHead>
                )
              })}
            </TableRow>
```

10. menambahkan program dalam file data-table.tsx untuk menampilkan pagination pada table

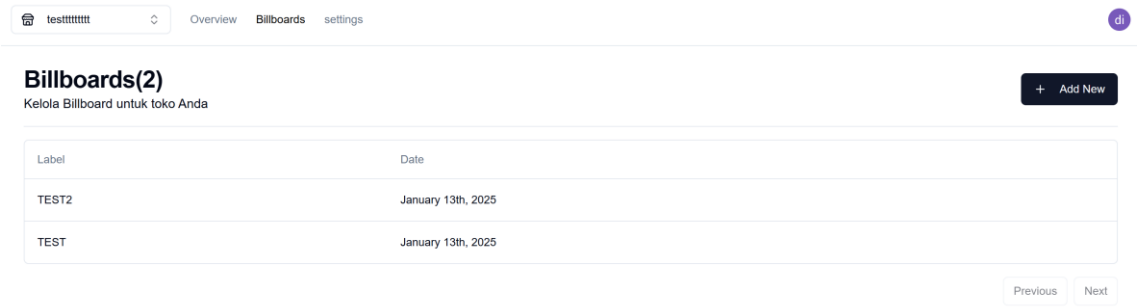
```tsx
import { Button } from "@/components/ui/button"
```

```tsx
import {
    ColumnDef,
    flexRender,
    getCoreRowModel,
    getPaginationRowModel,
    useReactTable,
} from "@tanstack/react-table"
```

```tsx
export function DataTable<TData, TValue>({
  columns,
  data,
}: DataTableProps<TData, TValue>) {
  const table = useReactTable({
    data,
    columns,
    getCoreRowModel: getCoreRowModel(),
    getPaginationRowModel: getPaginationRowModel(),
  })
```

```tsx
      </div>
      <div className="flex items-center justify-end space-x-2 py-4">
          <Button
              variant="outline"
              size="sm"
              onClick={() => table.previousPage()}
              disabled={!table.getCanPreviousPage()}
              >
              Previous
          </Button>
          <Button
              variant="outline"
              size="sm"
              onClick={() => table.nextPage()}
              disabled={!table.getCanNextPage()}
              >
              Next
          </Button>
      </div>
```

Tampilan ketika program dijalankan



11. Menambhakan program di file data-table.tsx untuk menampilkan filter

Mengimport input dari components/ui/input

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

Mengimport ColumnFilterState, dan getFilteredRowModel dari shadcnui.com

```
import {
  ColumnDef,
  ColumnFiltersState,
  flexRender,
  getCoreRowModel,
  getFilteredRowModel,
  getPaginationRowModel,
  useReactTable,
} from "@tanstack/react-table"
```

```
import { useState } from "react"
```

Membuat fungsi searchkey untuk searching

```
searchKey: string;
```

membuat fungsi columnFilters dan setColumnFilters menggunakan usestate

```
  data,
  searchKey,
}: DataTableProps<TData, TValue>) {
  const [columnFilters, setColumnFilters] = useState<ColumnFiltersState>(
    []
  )
```

memanggil columnFilters pada table

```
state: {
    columnFilters,
},
```

Membuat tampilan dan fungsi filter

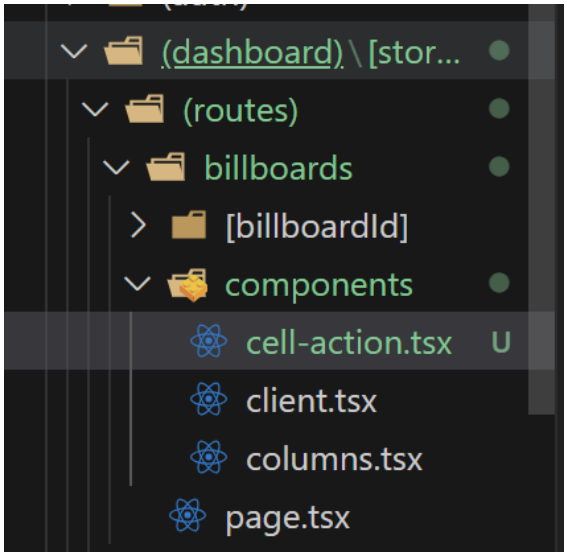
```
<div className="flex items-center py-4">
<Input
  placeholder="Search"
  value={(table.getColumn(searchKey)?.getFilterValue() as string) ?? ""}
  onChange={(event) =>
    table.getColumn(searchKey)?.setFilterValue(event.target.value)
  }
  className="max-w-sm"
/>
</div>
```

Memanggil fungsi searchKey pada client.tsx

```
<Separator />
<DataTable searchKey="label" columns={columns} data={data}/>
</>
```

Tampilan ketika program dijalankan

Membuat file baru yang bernama cell-action.tsx pada folder billboards/components



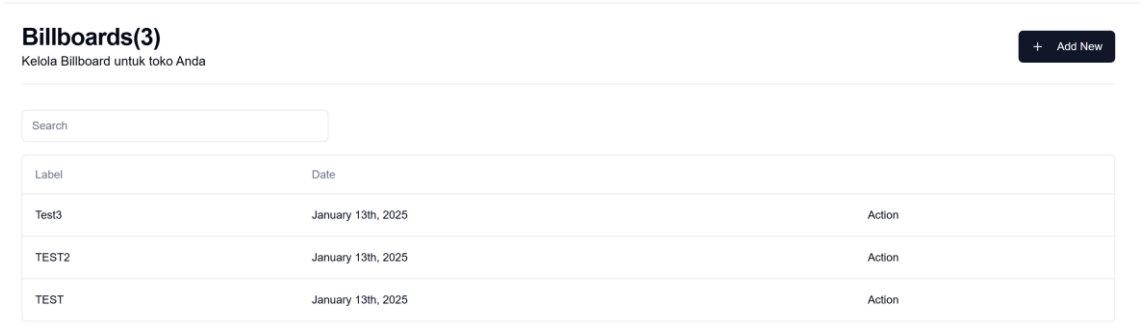
Menambhakan program pada file cell-action yang berupa fungsi CellAction yang didalamnya terdapat div Action

```
app > (dashboard) > [storeId] > (routes) > billboards > components > cell-action.tsx > ...
1   export const CellAction = () =>  {
2       return(
3           <div>Action</div>
4       );
5   };
```

Menambhakan program untuk memanggil fungsi CellAction yang tadi dibuat pada file column.tsx

```
{
  id: "Actions",
  cell: () => <CellAction/>
}
```

Tampilan ketika program dijalankan



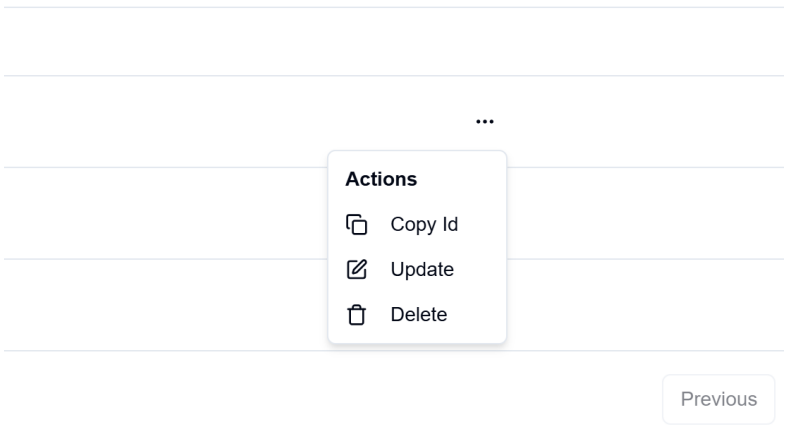
Menginstal components dropdown menu dari shadcn ui

```
npx shadcn@latest add dropdown-menu
```

Mengedit Program Action dengan menambhakan program untuk membuat Dropdown yang telah diinstal diatas

```
return(
    <DropdownMenu>
        <DropdownMenuTrigger asChild>
            <Button variant="ghost" className="h-8 w-8 p-0">
                <span className="sr-only"> Open Menu</span>
                <MoreHorizontal className="h-4 w-4"/>
            </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent align="end">
            <DropdownMenuLabel>
                Actions
            </DropdownMenuLabel>
            <DropdownMenuItem>
                <Copy className="mr-2 h-4 w-4"/>
                Copy Id
            </DropdownMenuItem>
            <DropdownMenuItem>
                <Edit className="mr-2 h-4 w-4"/>
                Update
            </DropdownMenuItem>
            <DropdownMenuItem>
                <Trash className="mr-2 h-4 w-4"/>
                Delete
            </DropdownMenuItem>
        </DropdownMenuContent>
    </DropdownMenu>
);
```

Tampilan dropdown ketika program dijalankan



Menambahakan program Fungsi copy pada file cell-action.tsx

```
}) =>  {
    const onCopy = (id: string) => {
        navigator.clipboard.writeText(id);
        toast.success("Billboard Id copied to  the clipboard")
     };
```

Lalu memanggilnya pada dropdown bagian copy

```
<DropdownMenuItem onClick={()=> onCopy(data.id)}>
    <Copy className="mr-2 h-4 w-4"/>
    Copy Id
</DropdownMenuItem>
```

Untuk Membuat fungsi update

Membuat fungsi router yang diimport dari fungsi useRouter dan membuat fungsi params yang diimport dari fungsi useParams

```
    const router= useRouter();
    const params= useParams();
```

```
import { useParams, useRouter } from "next/navigation";
```

Tambhakan fungsi onClick pada bagian update data dengan mengambil data/push dari database selanjutnya edit pda halaman billboards

```
<DropdownMenuItem onClick={()=> router.push(`/${params.storeId}/billboards/${data.id}`)}>
    <Edit className="mr-2 h-4 w-4"/>
    Update
</DropdownMenuItem>
```

Membuat program fungsi delete yang akan digunakan untuk menhapus data pada tabel

```
    const [loading, setLoading] = useState(false);
    const [Open, setOpen] = useState(false);
```

```
const onDelete = async () => {
    try {
        setLoading(true)
        await axios.delete(`/api/${params.storeId}/billboards/${params.billboardId}`);
        router.refresh();
        router.push("/")
        toast.success("Billboard deleted.")
    } catch (error) {
        toast.error("Make sure you removed all categories using this billboard first.");
    }finally{
        setLoading(false)
        setOpen(false)
    }
}
```

Lalu pada bagian return memanggil fungsi AlertModal dengan property isOpen,onClose,onConfirm dan loading.

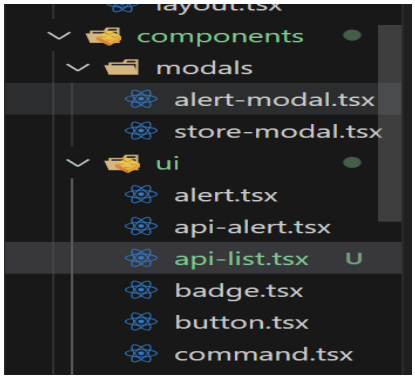
```
<AlertModal
isOpen={Open}
onClose={()=> setOpen(false)}
onConfirm={onDelete}
loading={loading}
/>
```

Setelah itu pada dropdown delete tambahakan fungsi onClick dengan memnggil setOpen yang nilainya true

```
<DropdownMenuItem onClick={() => setOpen(true)}>
    <Trash className="mr-2 h-4 w-4"/>
    Delete
</DropdownMenuItem>
```

Setelah program dijalankan maka data pada tabel sudah bisa dihapus.

Membuat file baru yang bernama api-list.tsx pada folder componnets/ui



```tsx
interface ApiListProps{
    entityName: string;
    entityIdName: string;
}
```

Interface ini mendefinisikan properti yang harus diterima oleh komponen `ApiList`.

- **`entityName`**: Nama entitas yang menjadi target endpoint API (contoh: `products`, `users`, dll.).
- **`entityIdName`**: Nama parameter ID yang digunakan untuk mengidentifikasi entitas tertentu (contoh: `productId`, `userId`, dll.).

```tsx
export const ApiList: React.FC<ApiListProps> =({
    entityName,
    entityIdName
}) => {

    const params = useParams();
    const origin = useOrigin();

    const baseUrl = `${origin}/ap/${params.storeId}`;
```

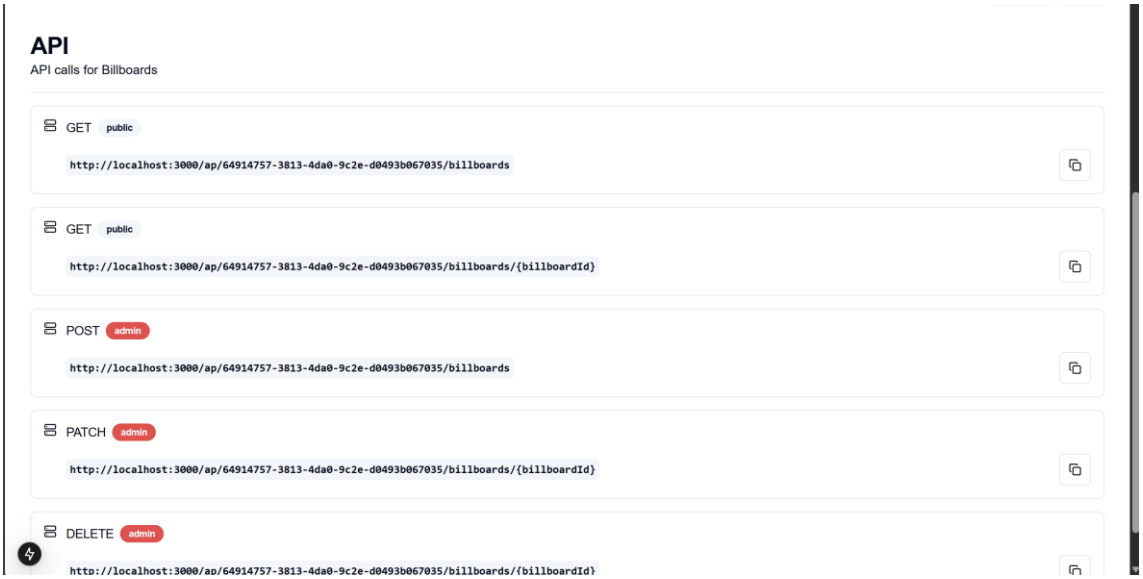
- **`params`**: Mengambil parameter URL, seperti `storeId` dari path dinamis `store/[storeId]`.
- **`origin`**: Mengambil URL dasar aplikasi
- **`baseUrl`**: Membentuk URL dasar API berdasarkan `origin` dan `storeId`,

```tsx
  return(
     <>
      <ApiAlert
      title="GET"
      variant="public"
      description={`${baseUrl}/${entityName}`}
      />
```

- **`title`**: Menunjukkan metode HTTP yang digunakan untuk endpoint (GET, POST, PATCH, DELETE).
- **`variant`**: Menentukan aksesibilitas endpoint:

    - `public`: Dapat diakses oleh semua pengguna.
    - `admin`: Hanya dapat diakses oleh admin.

- **`description`**: URL endpoint API baseUrl/entityname



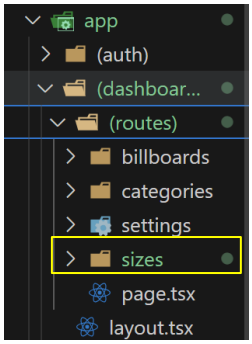
Tampilan ketika program dijalankan.

SIZE Entity(ADMIN)

```
54
55  model Size{
56    id String @id @default(uuid())
57    storeId String
58    store Store @relation("StoreToSize", fields: [storeId], references: [id])
59    name String
60    value String
61    createdAt DateTime @default(now())
62    updatedAt DateTime @updatedAt
63
64    @@index([storeId])
65  }
```

Membuat Model schema Prisma yang mendefinisikan tabel **Size** untuk menyimpan informasi ukuran produk dengan penjelasan berikut:

- **id**: Primary key dengan UUID unik.
- **storeId**: Foreign key yang menghubungkan Size ke model Store.
- **Relasi (store)**: Menghubungkan model Size ke Store menggunakan storeId.
- **name** & **value**: Menyimpan nama dan nilai ukuran.
- **createdAt** & **updatedAt**: Menyimpan waktu pembuatan dan pembaruan data secara otomatis.
- **@@index([storeId])**: Membuat indeks pada kolom storeId untuk meningkatkan performa query.



membuat folder sizes pada folder app/dashboard/routes dengan cara mencopy folder billboards kemudian ganti nama dengan sizes

```
    {
        href: `/${params.storeId}/sizes`,
        label: 'Sizes',
        active: pathname === `/${params.
        storeId}/sizes`,
    },
```

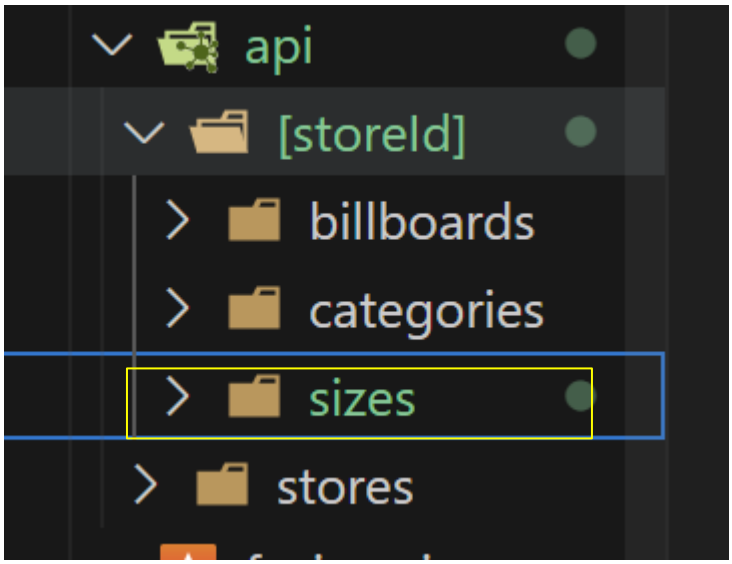
Menambahkan program pada fil main-nav.tsx untuk menampilkan halaman sizes

```
"use client"

import { ColumnDef } from "@tanstack/react-table"
import { CellAction } from "./cell-action"

// This type is used to define the shape of our data.
// You can use a Zod schema here if you want.
export type SizeColumn = {
  id: string
  name: string
  value: string
  createdAt: string
}

export const columns: ColumnDef<SizeColumn>[] = [
  {
    accessorKey: "name",
    header: "Name",
  },
  {
    accessorKey: "value",
    header: "Value",
  },
  {
    accessorKey: "createdAt",
    header: "Date",
  },
  {
    id: "Actions",
    cell: ({row}) => <CellAction data={row.original}/>
  }
]
```

Ubah program pada column.tsx yang awalnya bernama BillboardColumn menjadi SizeColumn, kemudian tambahkan property id, name dan value dan ubah accesorkey menjadi name dan value seperti diprogram.

```tsx
interface SizesClientProps{
    data: SizeColumn[]
}

export const SizesClient: React.FC<SizesClientProps> = ({
    data
}) => {
    const router = useRouter();
    const params = useParams();


    return (
        <>
            <div className="flex items-center justify-between">
                <Heading
                    title={`Sizes(${data.length})`}
                    description="Kelola Ukuran untuk toko Anda"
                />
                <Button onClick={() => router.push(`/${params.storeId}/sizes/new`)}>
                    <Plus className="mr-2 h-4 w-4"/>
                    Add New
                </Button>
            </div>
            <Separator />
            <DataTable searchKey="name" columns={columns} data={data}/>
            <Heading title="API" description="API calls for Sizes"/>
            <Separator/>
            <ApiList entityName="sizes" entityIdName="sizeId"/>
        </>
    )
}
```

Ubah juga pada program client.tsx yang awalnya billboard menjadi Sizes seperti yang ditandaikan dalam kotak.

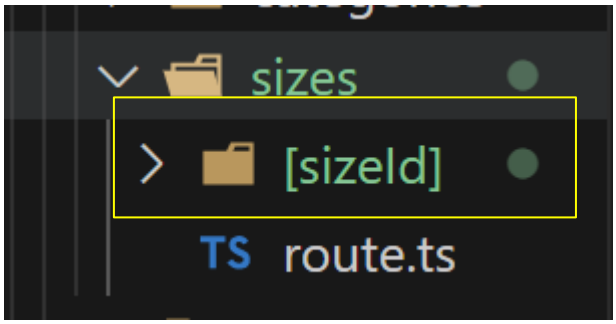


Copy folder billboards kemudian pastekan dalam folder [storeId] lalu ubah nama folder copy menjadi sizes

```
export async function POST(
    req: Request,
    {params}: {params: {storeId: string}}
) {
    try{
        const { userId} = await auth()
        const body = await req.json();

        const {name, value} = body

        if (!userId){
            return new NextResponse("Unauthenticated", {status: 401})
        }

        if (!name){
            return new NextResponse("Nama Toko Perlu diinput", {status: 400})
        }

        if (!value){
            return new NextResponse("Value Perlu diinput", {status: 400})
        }

        if (!params.storeId) {
            return new NextResponse("Id Toko Perlu diinput", {status: 400})
        }

        const storeByUserId =  await db.store.findFirst({
            where: {
                id: params.storeId,
                userId
            }
        });
```

```
    const size = await db.size.create({
        data: {
            name,
            value,
            storeId: params.storeId
        },
    });
return NextResponse.json(size);
}catch (error){
    console.log("[SIZES_POST]", error)
    return new NextResponse("Internal error", {status: 500})
}


xport async function GET(
  req: Request,
  {params}: {params: {storeId: string}}
{
  try{
    if (!params.storeId) {
        return new NextResponse("Id Toko Perlu diinput", {status: 400})
    }

    const size = await db.size.findMany({
        where: {
            storeId: params.storeId,
        }
    });
return NextResponse.json(size);
}catch (error){
    console.log("[SIZES_GET]", error)
```

Dalam folder sizes terdapat file route.tsx kemudian ubah program menjadi seperti yang dikotakkan



Ubah nama folder billboardId menjadi sizeId

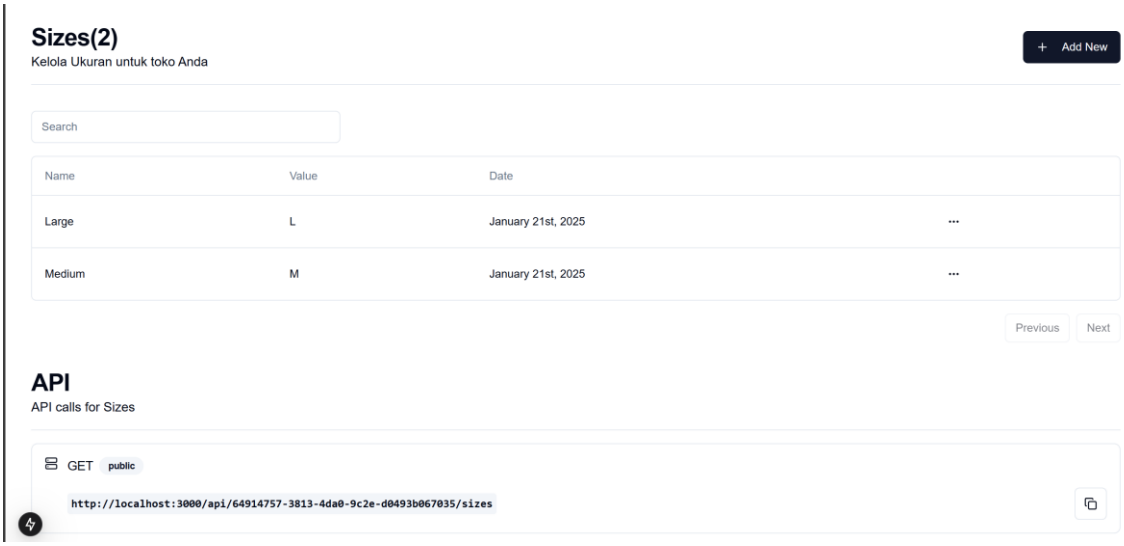
```ts
import db from "@/lib/db";
import { auth } from "@clerk/nextjs/server";
import { NextResponse } from "next/server";


export async function GET(
    req: Request,
    { params }: { params: { sizeId: string } }
  ) {
    try {
      if (!params.sizeId) {
        return new NextResponse("ukuran id dibutuhkan", { status: 400 });
      }

      const size = await db.size.findUnique({
        where: {
          id: params.sizeId,
        },
      });

      return NextResponse.json(size);
    } catch (error) {
      console.log("[SIZE_GET]", error);
      return new NextResponse("Internal error", { status: 500 });
    }
  }
```

```ts
export async function PATCH(
  req: Request,
  { params }: { params: { storeId: string, sizeId: string } }
) {
  try {
    const { userId } = await auth();
    const body = await req.json();

    const { name, value } = body;

    if (!userId) {
      return new NextResponse("Unauthenticated", { status: 401 });
    }
    if (!name) {
      return new NextResponse(" menginput nama", { status: 400 });
    }
    if (!value) {
        return new NextResponse("menginput Value", { status: 400 });
      }
    if (!params.sizeId) {
      return new NextResponse("Size Diinputkan", { status: 400 });
    }
```

```
    const storeByUserId =  await db.store.findFirst({
        where: {
            id: params.storeId,
            userId
        }
    });

    if (!storeByUserId) {
        return new NextResponse("Unauthorized", {status: 403});
    }

    const size = await db.size.updateMany({
      where: {
        id: params.sizeId,
      },
      data: {
        name,
        value
      },
    });

    return NextResponse.json(size);
  } catch (error) {
    console.log("[SIZE_PATCH]", error);
    return new NextResponse("Internal error", { status: 500 });
  }
}
```

```
export async function DELETE(
    req: Request,
    { params }: { params: { storeId: string, sizeId: string } }
  ) {
    try {
      const { userId } = await auth();


      if (!userId) {
        return new NextResponse("Unauthenticated", { status: 401 });
      }

      if (!params.sizeId) {
        return new NextResponse("id ukuran dibutuhkan", { status: 400 });
      }

      const storeByUserId =  await db.store.findFirst({
        where: {
            id: params.storeId,
            userId
        }
    });

    if (!storeByUserId) {
        return new NextResponse("Unauthorized", {status: 403});
    }

      const size = await db.size.deleteMany({
        where: {
          id: params.sizeId,
        },
      });
```

Ubah program pada file route.ts dalam folder sizeId menjadi seperti diatas



Tampilan Keetika program dijalankan.

COLOR Entity(ADMIN)

Membuat model schema prisma untuk warna seperti berikut:

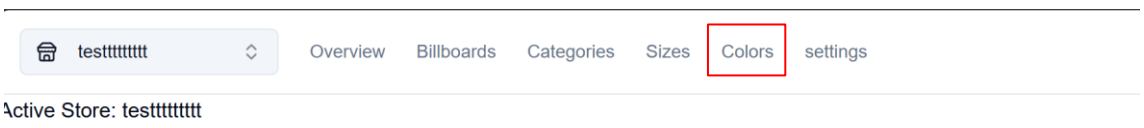
```
model Color{
  id String @id @default(uuid())
  storeId String
  store Store @relation("StoreToColor", fields: [storeId], references: [id])
  name String
  value String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@index([storeId])
}
```

Tambahakan color untuk relasi pada model store sebagai berikut:

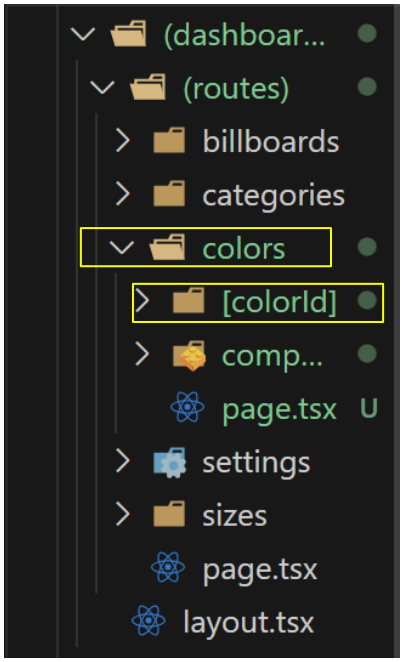
```
model Store{
    id String @id @default(uuid())
    name String
    userId String
    billboards Billboard[] @relation("StoreToBillboard")
    categories Category[] @relation("StoreToCategory")
    sizes Size[] @relation("StoreToSize")
    colors Color[] @relation("StoreToColor")
    createdAt DateTime @default(now())
    updatedAt  DateTime @updatedAt
  }
```

Tambahakn program untuk menampilkan halaman warna pada navigasi dalam file main-nav.tsx sebagai berikut:

```
    {
        href: `/${params.storeId}/colors`,
        label: 'Colors',
        active: pathname === `/${params.storeId}/colors`,
    },
```



Copy folder **size** yang terdapat pada **dashboard/[storeId]/(routes)/..** ubah nama folder menjadi **colors** kemudian dalam folder colors terdapat folder yang bernama [sizeId] ubah menjadi [colorId]. Seperti pada gambar :



Ubah program pada file page.tsx dalam folder size menjadi sepert berikut:

app > (dashboard) > [storeId] > (routes) > colors > page.tsx > ColorsPage

```tsx
import { format, formatDate } from "date-fns"

import db from "@/lib/db";
import { SizesClient } from "./components/client";
import { SizeColumn } from "./components/columns";

const ColorsPage = async ({
    params
}: {
    params: {storeId: string}
}) => {
    const colors= await db.color.findMany({
        where: {
            storeId: params.storeId
        },
        orderBy: {
            createdAt: 'desc'
        }
    });

    const formattedColors: SizeColumn[] = colors.map((item) => ({
        id: item.id,
        name: item.name,
        value: item.value,
        createdAt: formatDate(item.createdAt,"MMMM do, yyyy" )
    }));

    return (
        <div className="flex-col">
            <div className="flex-1 space-y-4 p-8 pt-6">
                <SizesClient data={formattedColors}/>
            </div>
        </div>
    );
}

export default ColorsPage;
```

Lalu pada folder components dalam folder color ubah program pada file columns.tsx menjadi seperti berikut :

```tsx
"use client"

import { ColumnDef } from "@tanstack/react-table"
import { CellAction } from "./cell-action"

// This type is used to define the shape of our data.
// You can use a Zod schema here if you want.
export type ColorColumn = {
  id: string
  name: string
  value: string
  createdAt: string
}

export const columns: ColumnDef<ColorColumn>[] = [
  {
    accessorKey: "name",
    header: "Name",
  },
  {
    accessorKey: "value",
    header: "Value",
    cell: ({ row }) => (
      <div className="flex items-center gap-x-2">
        {row.original.value}
        <div className="h-6 w-6 rounded-full border" style={{backgroundColor: row.original.value}}/>
      </div>
    )
  },
  {
    accessorKey: "createdAt",
    header: "Date",
  },
  {
    id: "Actions",
    cell: ({row}) => <CellAction data={row.original}/>
  }
]
```

ubah SizeColumn menjadi ColorColumn pada file page.tsx sebelumnya :

```tsx
import { format, formatDate } from "date-fns"

import db from "@/lib/db";
import { SizesClient } from "./components/client";
import { ColorColumn } from "./components/columns";

const ColorsPage = async ({
    params
}: {
    params: {storeId: string}
}) => {
    const colors = await db.color.findMany({
        where: {
            storeId: params.storeId
        },
        orderBy: {
            createdAt: 'desc'
        }
    });

    const formattedColors: ColorColumn[] = colors.map((item) => ({
        id: item.id,
        name: item.name,
        value: item.value,
        createdAt: formatDate(item.createdAt,"MMMM do, yyyy" )
    }));

    return (
        <div className="flex-col">
            <div className="flex-1 space-y-4 p-8 pt-6">
                <SizesClient data={formattedColors}/>
            </div>
        </div>
    );
}

export default ColorsPage;
```

(property) orderBy: { createdAt: "desc"; }

Ubah juga program client.tsx pada folder component Colors seperti berikut :

```tsx
import { ColorColumn, columns } from "./columns"
import { DataTable } from "@/components/ui/data-table"
import { ApiList } from "@/components/ui/api-list"

interface ColorsClientProps{
    data: ColorColumn[]
}

export const ColorsClient: React.FC<ColorsClientProps> = ({
    data
}) => {
    const router = useRouter();
    const params = useParams();


    return (
        <>
            <div className="flex items-center justify-between">
                <Heading
                    title={`Colors(${data.length})`}
                    description="Kelola Warna untuk toko Anda"
                />
                <Button onClick={() => router.push(`/${params.storeId}/colors/new`)}>
                    <Plus className="mr-2 h-4 w-4"/>
                    Add New
                </Button>
            </div>
            <Separator />
            <DataTable searchKey="name" columns={columns} data={data}/>
            <Heading title="API" description="API calls for Colors"/>
            <Separator/>
            <ApiList entityName="colors" entityIdName="colorId"/>
        </>
    )
}
```

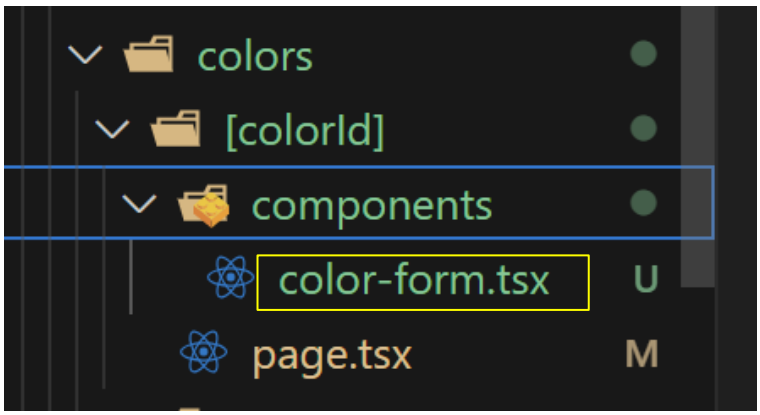
Lalu ubah SizesClient menjadi ColorsClient pada file page.tsx seperti ini :

```tsx
import db from "@/lib/db";
import { ColorsClient } from "./components/client";
import { ColorColumn } from "./components/columns";

const ColorsPage = async ({
    params
}: {
    params: {storeId: string}
}) => {
    const colors= await db.color.findMany({
        where: {
            storeId: params.storeId
        },
        orderBy: {
            createdAt: 'desc'
        }
    });

    const formattedColors: ColorColumn[] = colors.map((item) => ({
        id: item.id,
        name: item.name,
        value: item.value,
        createdAt: formatDate(item.createdAt,"MMMM do, yyyy" )
    }));

    return (
        <div className="flex-col">
            <div className="flex-1 space-y-4 p-8 pt-6">
                <ColorsClient data={formattedColors}/>
            </div>
        </div>
    );
}

export default ColorsPage;
```

Ubah page.tsx pada folder [colorId] menjadi seperti berikut :

```tsx
import db from "@/lib/db";
import { SizeForm } from "./components/size-form";

const ColorPage = async ({ params }: { params: { colorId: string } }) => {
    // Tunggu `params` sebelum mengakses properti `sizeId`
    const { colorId } = await params;

    const color = await db.color.findUnique({
        where: {
            id: colorId, // Gunakan sizeId setelah params di-await
        },
    });

    return (
        <div className="flex-col">
            <div className="flex-1 space-y-4 p-8 pt-6">
                <SizeForm initialData={color} />
            </div>
        </div>
    );
};

export default ColorPage;
```

Ubah folder size-form pada folder components colorId menjadi color-form.tsx



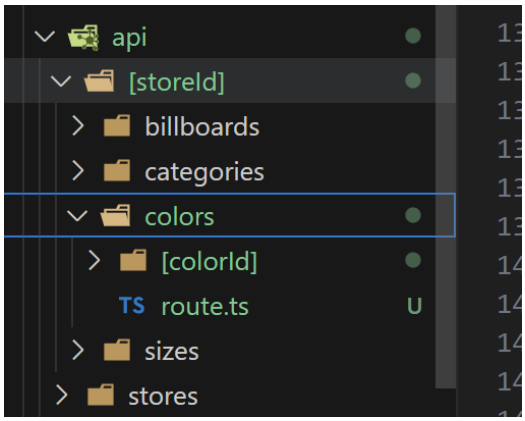
Ubah program pada file color-form.tsx menjadi seperti berikut :

```tsx
const formSchema = z. object ({
    name: z.string().min(1),
    value: z.string().min(4).regex(/^#/, {
        message: 'String must be a valid hex code'
    })
});

type ColorFormValues = z. infer<typeof formSchema>;

interface ColorFromProps {
    initialData: Color | null;
}

export const ColorForm: React.FC<ColorFromProps> = ({
    initialData
}) => {
const params = useParams();
const router = useRouter();


const [open,setOpen] = useState(false);
const [loading,setLoading] = useState(false);

const title = initialData ? "Edit color" : "Create color";
const description = initialData ? "Edit color" : "Add a new color";
const toastMessage = initialData ? "Color update" : "Color created";
const action = initialData ? "Save change" : "Create";

    const form = useForm<ColorFormValues>({
        resolver: zodResolver(formSchema),
        defaultValues: initialData || {
            name: '',
            value: '',
        }
    });
```

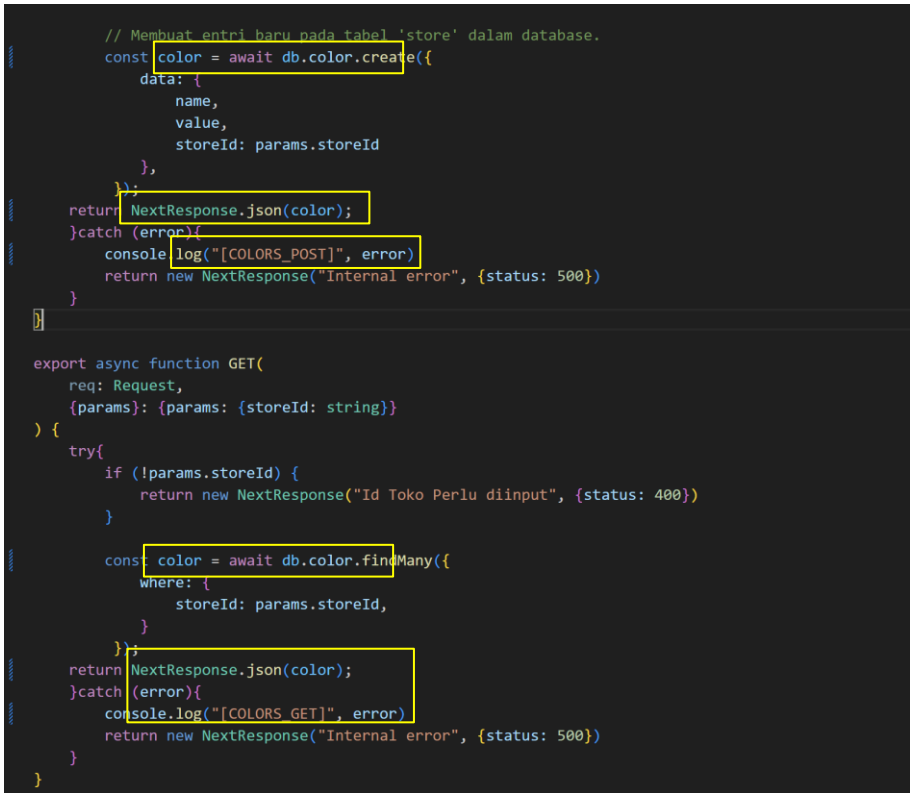
```tsx
const onSubmit = async (data: ColorFormValues) =>{
    try {
        setLoading(true);
        if (initialData) {
            await axios.patch(`/api/${params.storeId}/colors/${params.colorId}`,data);
        } else {
            await axios.post(`/api/${params.storeId}/colors`,data);
        }
        router.refresh();
        router.push(`/${params.storeId}/colors`)
        toast.success(toastMessage);
    } catch (error) {
        toast.error("Something went wrong.");
    }finally{
        setLoading(false);
    }
};

const onDelete = async () => {
    try {
        setLoading(true)
        await axios.delete(`/api/${params.storeId}/colors/${params.colorId}`);
        router.refresh();
        router.push(`/${params.storeId}/colors`)
        toast.success("Color deleted.")
    } catch (error) {
        toast.error("Make sure you removed all products using this color first.");
    }finally{
        setLoading(false)
        setOpen(false)
    }
}
```

```
</div>
<Separator />
<Form {...form}>
    <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-8 w-full">
        <div className="grid grid-cols-3 gap-8">
         <FormField
         control={form.control}
         name="name"
         render={({field}) => (
            <FormItem>
                <FormLabel>Name</FormLabel>
                <FormControl>
                    <Input disabled={loading} placeholder="Color name"{...field} />
                </FormControl>
                <FormMessage />
            </FormItem>
         )}
        />
         <FormField
         control={form.control}
         name="value"
         render={({field}) => (
            <FormItem>
                <FormLabel>Value</FormLabel>
                <FormControl>
                    <div className="flex items-center gap-x-4">
                    <Input disabled={loading} placeholder="Color value"{...field} />
                    <div
                    className=" border p-4 rounded-full"
                    style={{backgroundColor: field.value}}/>
                    </div>
                </FormControl>
                <FormMessage />
            </FormItem>
         )}
        />
        </div>
```

Buka folder app/api/[storeId] kemudian copy folder sizes pastekan dalam folder [storeId] lalu ubah nama menjadi colors dan sizeId menjadi colorId seperti berikut :

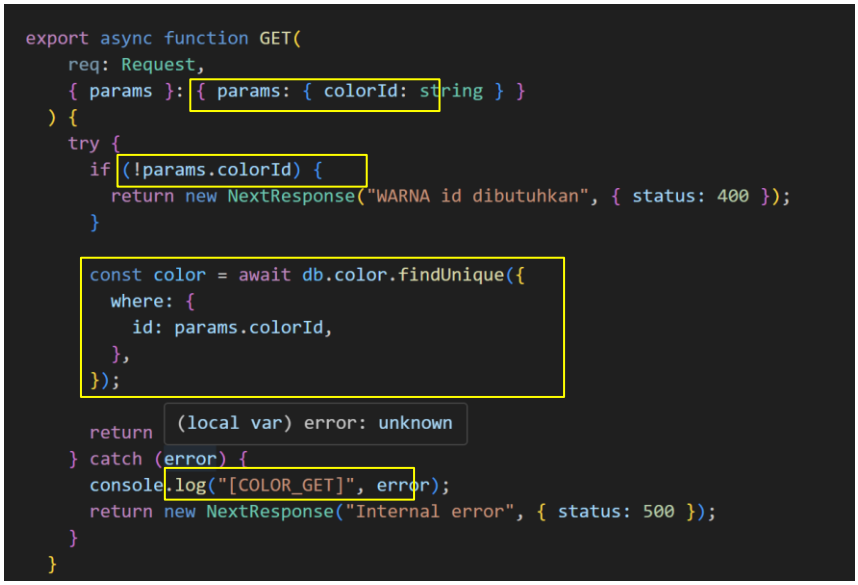


Ubah program pada bagian berikut di dalam file route.ts menjadi seperti berikut:



```
        // Membuat entri baru pada tabel 'store' dalam database.
        const color = await db.color.create({
            data: {
                name,
                value,
                storeId: params.storeId
            },
        });
    return NextResponse.json(color);
    }catch (error){
        console.log("[COLORS_POST]", error)
        return new NextResponse("Internal error", {status: 500})
    }
}

export async function GET(
    req: Request,
    {params}: {params: {storeId: string}}
) {
    try{
        if (!params.storeId) {
            return new NextResponse("Id Toko Perlu diinput", {status: 400})
        }

        const color = await db.color.findMany({
            where: {
                storeId: params.storeId,
            }
        });
    return NextResponse.json(color);
    }catch (error){
        console.log("[COLORS_GET]", error)
        return new NextResponse("Internal error", {status: 500})
    }
}
```

UBAH juga program pada file route.ts yang terdapat dalam folder colorId seperti berikut:



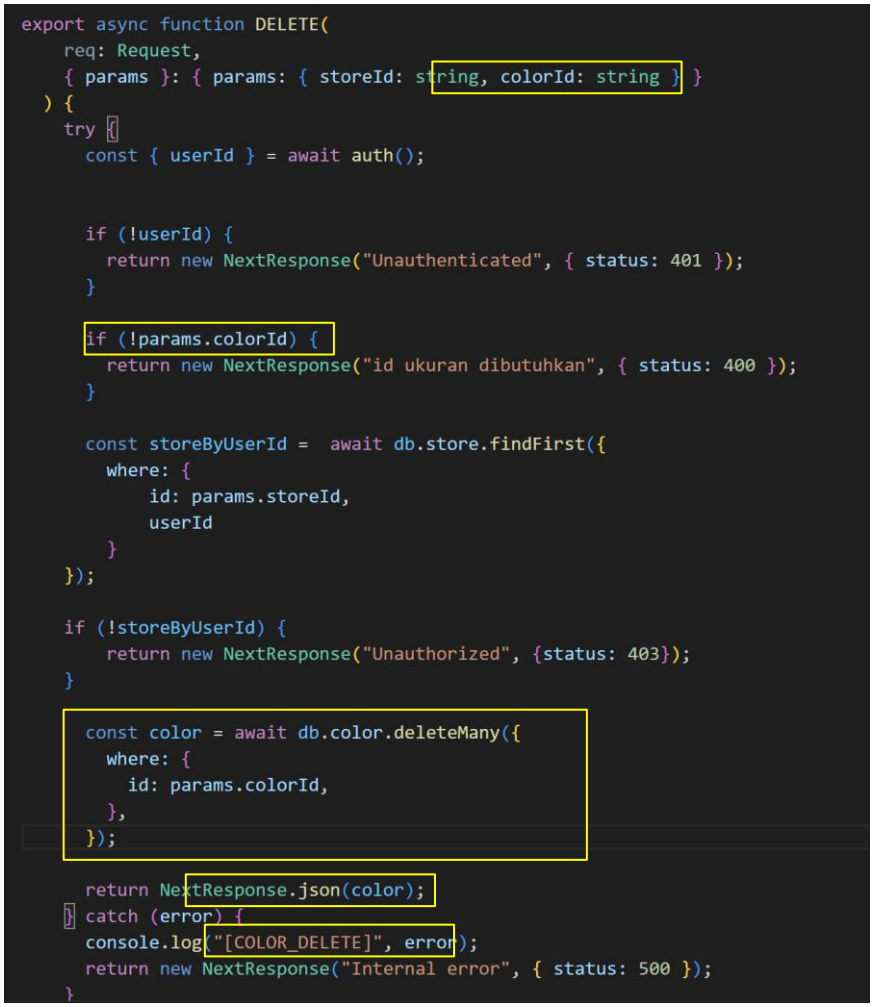
```
export async function GET(
    req: Request,
    { params }: { params: { colorId: string } }
  ) {
    try {
      if (!params.colorId) {
        return new NextResponse("WARNA id dibutuhkan", { status: 400 });
      }

      const color = await db.color.findUnique({
        where: {
          id: params.colorId,
        },
      });

      return
    } catch (error) {
      console.log("[COLOR_GET]", error);
      return new NextResponse("Internal error", { status: 500 });
    }
  }
```

(local var) error: unknown

```ts
export async function PATCH(
  req: Request,
  { params }: { params: { storeId: string, colorId: string } }
) {
  try {
    const { userId } = await auth();
    const body = await req.json();

    const { name, value } = body;

    if (!userId) {
      return new NextResponse("Unauthenticated", { status: 401 });
    }
    if (!name) {
      return new NextResponse(" menginput nama", { status: 400 });
    }
    if (!value) {
        return new NextResponse("menginput Value", { status: 400 });
      }
    if (!params.colorId) {
      return new NextResponse("Warna id Diinputkan", { status: 400 });
    }

    const storeByUserId =  await db.store.findFirst({
        where: {
            id: params.storeId,
            userId
        }
    });
```

(property) ResponseInit.

```ts
    if (!storeByUserId) {
        return new NextResponse("Unauthorized", {status: 403});
    }

    const color = await db.color.updateMany({
      where: {
        id: params.colorId,
      },
      data: {
        name,
        value
      },
    });

    return NextResponse.json(color);
  } catch (error) {
    console.log("[COLOR_PATCH]", error);
    return new NextResponse("Internal error", { status: 500 });
  }
}
```



```
export async function DELETE(
    req: Request,
    { params }: { params: { storeId: string, colorId: string } }
  ) {
    try {
      const { userId } = await auth();


      if (!userId) {
        return new NextResponse("Unauthenticated", { status: 401 });
      }

      if (!params.colorId) {
        return new NextResponse("id ukuran dibutuhkan", { status: 400 });
      }

      const storeByUserId =  await db.store.findFirst({
        where: {
            id: params.storeId,
            userId
        }
    });

    if (!storeByUserId) {
        return new NextResponse("Unauthorized", {status: 403});
    }

      const color = await db.color.deleteMany({
        where: {
          id: params.colorId,
        },
      });

      return NextResponse.json(color);
    } catch (error) {
      console.log("[COLOR_DELETE]", error);
      return new NextResponse("Internal error", { status: 500 });
    }
```