

Nama : Muhamad Dimas Stiyawan

NPM : 22312087

Environment 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

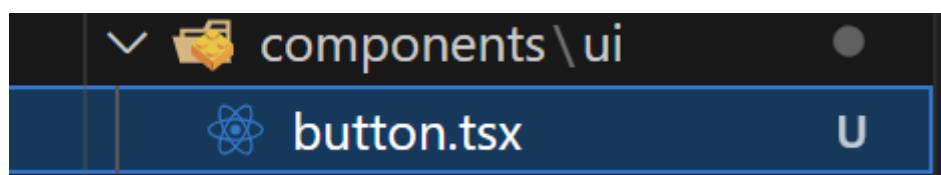
```
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex> npx shadcn@latest init
✓ Preflight checks.
✓ Verifying framework. Found Next.js.
✓ Validating Tailwind CSS.
✓ Validating import alias.
✓ Which style would you like to use? » Default
✓ Which color would you like to use as the base 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.
```

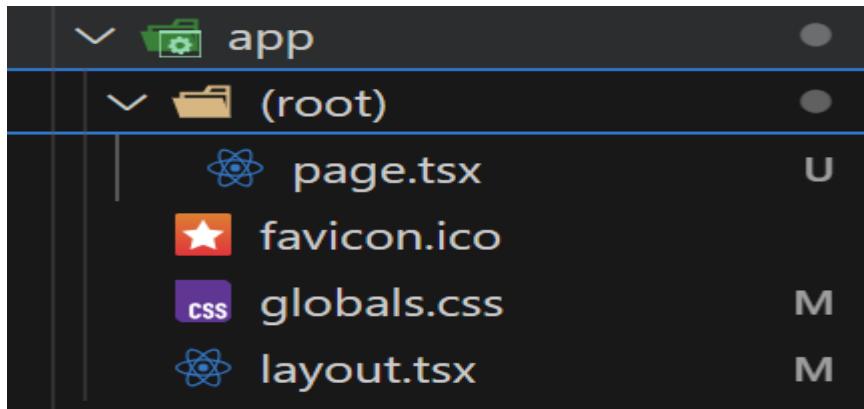
- jika sudah akan muncul folder components yg berisi file button.tsx



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
12 }
```

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

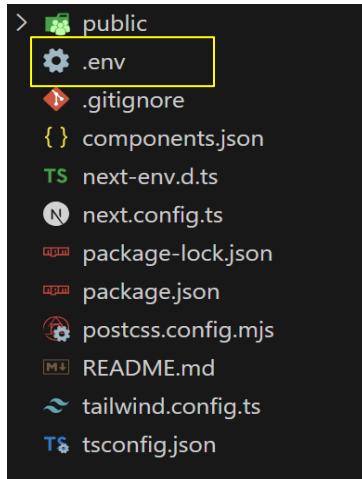


Clerk Authentication (Admin)

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

The image shows two side-by-side screenshots. On the left, the 'Let's build your <SignIn />' configuration screen for an application named 'VisionVortex'. It lists sign-in options: Email (selected), Phone number, Username, Google (selected), Facebook, Apple, and GitHub. A 'Create application' button is at the bottom. On the right, a preview of the 'Sign into VisionVortex' page. It features a 'Continue with Google' button and a 'Continue' button for email. Below these are links for 'Sign up' and 'Don't have an account?'. The text 'Secured by clerk' and 'This is a preview' are also visible.

2. buat file .env pada project



3. memasukkan API Keys clerk ke dalam file .env

```
ecommerce-visionvortex > .env
1 NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_test_bmVlZGVkLW1hcm1vc2V0LTU3LmNsZXJrLmFjY291bnRzLmRldiQ
2 CLERK_SECRET_KEY=sk_test_1G9pQJB04QybNOWUCCyXgNzxXFQ4cdyYDj42PifhU
3
```

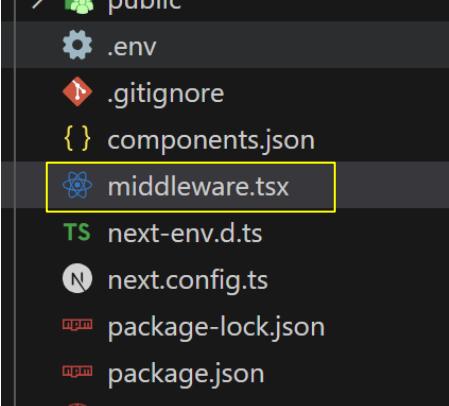
4. menginstal clerk pada project

```
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex> npm install @clerk/nextjs
added 25 packages, and audited 174 packages in 21s
37 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS D:\Pemrograman Web 2\ecommerce_VisionVortex\ecommerce-visionvortex>
```

5. mengimport clerkProvider pada file layout.tsx

```
1 import type { Metadata } from "next";
2 import { Geist, Geist_Mono } from "next/font/google";
3 import { ClerkProvider } from "@clerk/nextjs";
4 import "./globals.css";
5
6 > const geistSans = Geist({
7   weight: "normal",
8 });
9
10 > const geistMono = Geist_Mono({
11   weight: "normal",
12 });
13
14 export const metadata: Metadata = {
15   title: "Admin Dashboard",
16   description: "Admin Dashboard",
17 };
18
19 export default function RootLayout({
20   children,
21 }: Readonly<{
22   children: React.ReactNode;
23 }>) {
24   return (
25     <html lang="en">
26       <body
27         className={`${geistSans.variable} ${geistMono.variable} antialiased`}>
28           {children}
29         </body>
30       </html>
31     </ClerkProvider>
32   );
33 }
34
35 </ClerkProvider>
36
37 }
```

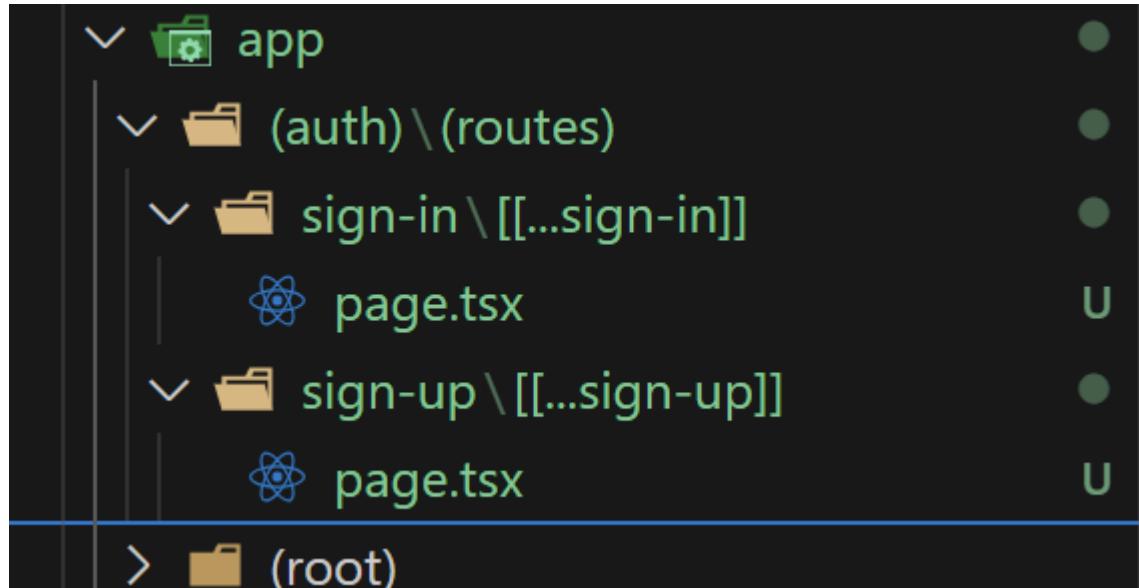
6. membuat file middleware.tsx dan menambahkan program config clerk authentication



```
ecommerce-visionvortex > 📂 middleware.tsx
1 import { clerkMiddleware } from "@clerk/nextjs/server";
2
3 export default clerkMiddleware();
4
5 <export const config = {
6   matcher: [
7     // Skip Next.js internals and all static files, unless found in search params
8     '/(?!_next|[^\\.]*\\.\\.(?:html|css|js)(?:\\?on|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*',
9     // Always run for API routes
10    '/(api|trpc)(.*)',
11  ],
12};
```

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
1 import { SignIn } from '@clerk/nextjs'
2
3 export default function Page() {
4   return <SignIn />
5 }
```

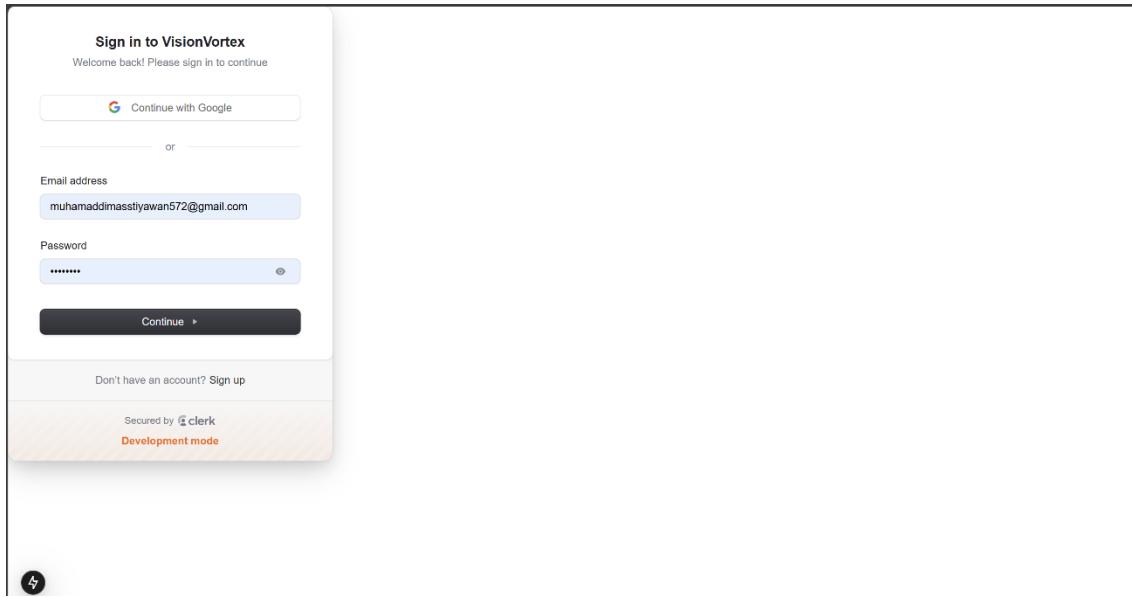
3. memasukkan program sign-up pada file page.tsx dalam folder sign-up

```
ecommerce-visionvortex > app > (auth) > (routes) > sign-up > [[...sign-up]] > ⚡ page.tsx > 📄 Page
1 import { SignUp } from '@clerk/nextjs'
2
3 export default function Page() {
4   return <SignUp />
5 }
```

4. menambahkan program pada file .env

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

5. tampilan setelah program dijalankan



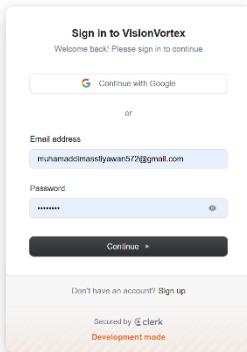
6. membuat file layout.tsx pada folder (routes)



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

```
ecommerce-visionvortex > app > (auth) > (routes) > layout.tsx > AuthLayout
1  export default function AuthLayout({ 
2    children
3  }: { 
4    children: React.ReactNode
5  }) {
6    return (
7      <div className="flex items-center justify-center min-h-screen ">{children}</div>
8    )
9 }
```

8. tampilan setelah program dijalankan



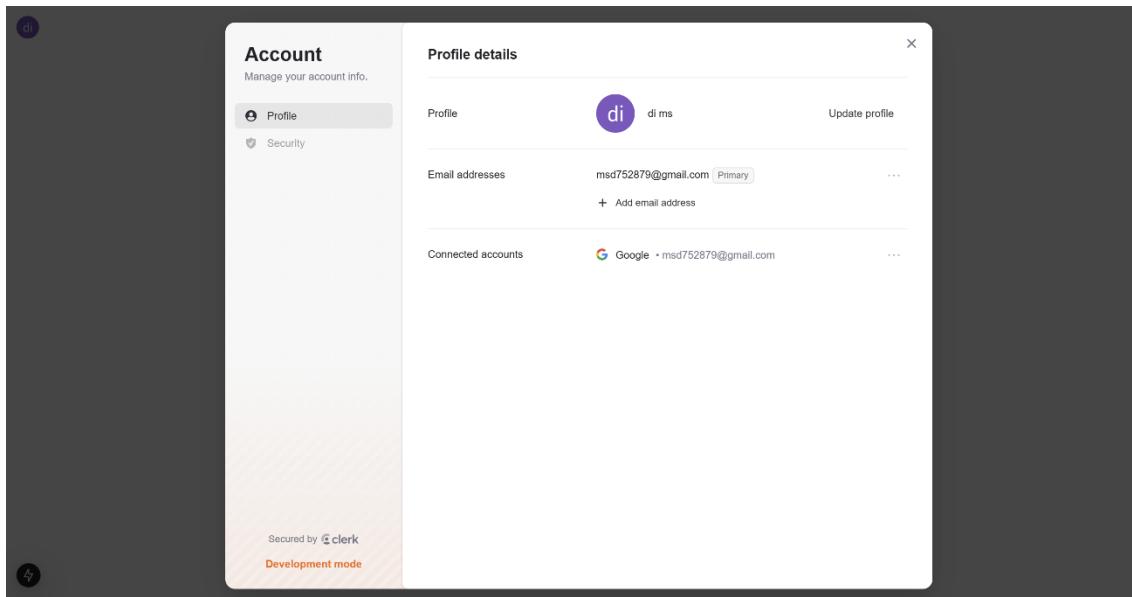
9. tampilan setelah sign-in menggunakan google



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

```
ecommerce-visionvortex > app > (root) > ⚙️ page.tsx > ...
1 import { Button } from "@/components/ui/button";
2 import { UserButton } from "@clerk/nextjs";
3 import Image from "next/image";
4
5 const setUpPage = () => {
6   return (
7     <div className="p-4">
8       <UserButton></UserButton>
9     </div>
10  );
11}
12
13
14 export default setUpPage
15
```

11. tampilan setelah program dijalankan

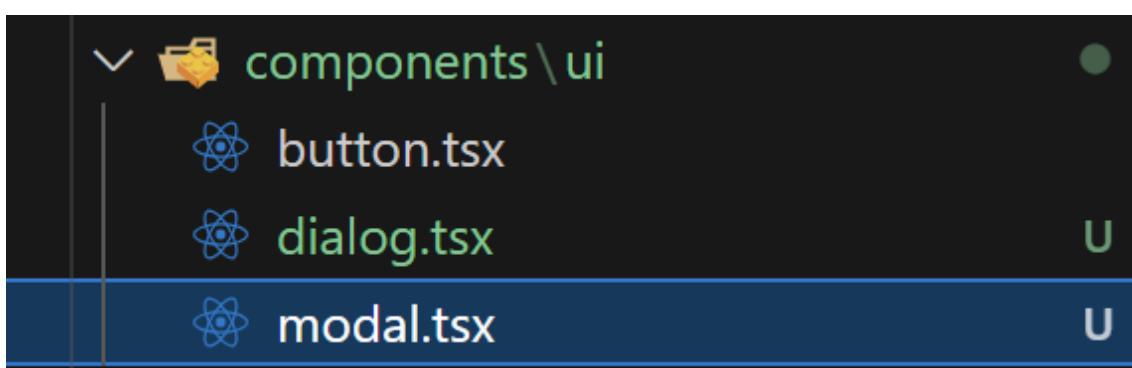


Modal Components

1. menginstal ui dialog dari Shadcnui.com

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

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



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

```
"use client"

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

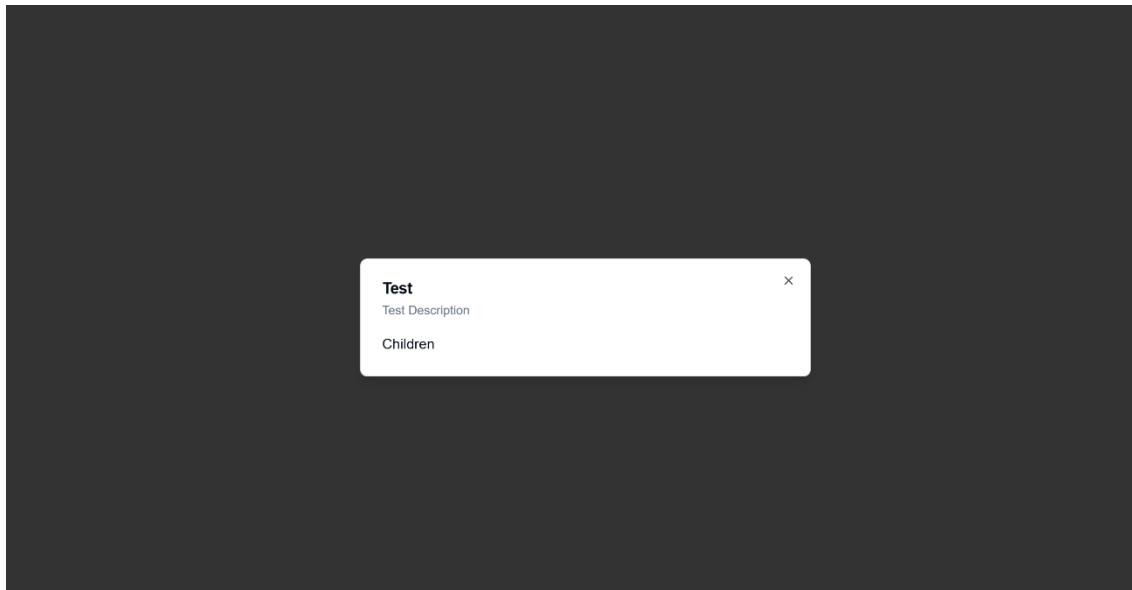
// Interface untuk mendefinisikan properti yang akan diterima oleh komponen Modal
interface ModalProps{
    title: string;
    description: string;
    isOpen: boolean;
    onClose: () => void;
    children?: React.ReactNode;
}

// 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>
                <DialogTitle>{title}</DialogTitle>
                <DialogDescription>
                    {description}
                </DialogDescription>
            </DialogTitle>
            <div>
                {children}
            </div>
        </DialogContent>
    </Dialog>
};
```

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

```
1 "use client"
2 // Mengimpor komponen `Modal` dari path `@/components/ui/modal`.
3 import { Modal } from "@/components/ui/modal";
4
5
6 // Fungsi `setUpPage` adalah komponen fungsional React yang digunakan untuk merender halaman.
7 const setUpPage = () => {
8     return (
9         <div className="p-4">
10            <Modal title="Test" description="Test Description" isOpen onClose={() => {}}>
11                Children
12            </Modal>
13        </div>
14
15    );
16 }
17
18 // Mengekspor `setUpPage` sebagai default, sehingga komponen ini dapat digunakan di file lain.
19 export default setUpPage
20
```

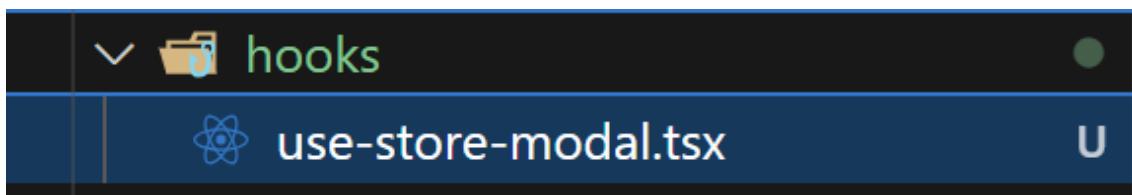
tampilan ketika program dijalankan



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

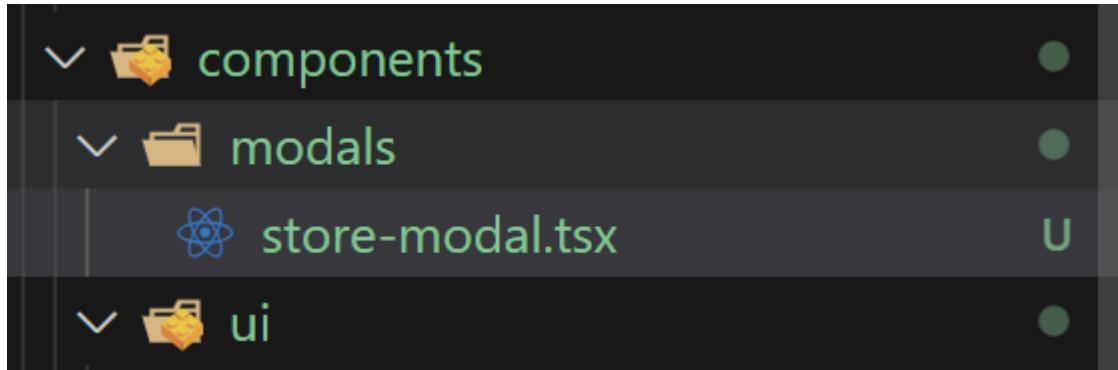


7. menambahkan program pada file use-store-modal.tsx berfungsi untuk mengelola status modal secara global menggunakan pustaka Zustand.

```
1 // Mengimpor fungsi 'create' dari pustaka Zustand untuk membuat store global pada aplikasi React.
2 import { create } from "zustand";
3
4 // Mendefinisikan tipe data untuk store yang mengelola status modal.
5 interface useStoreModalStore{
6     isOpen: boolean;
7     onOpen: () => void;
8     onClose: () => void;
9 };
10
11 // Membuat store dengan Zustand dan mendefinisikan status dan fungsi pengelolaan modal.
12 export const useStoreModal = create<useStoreModalStore>((set) => {
13     isOpen: false,
14     onOpen: () => set({isOpen: true}),
15     onClose: () => set({isOpen: false}),
16 });

```

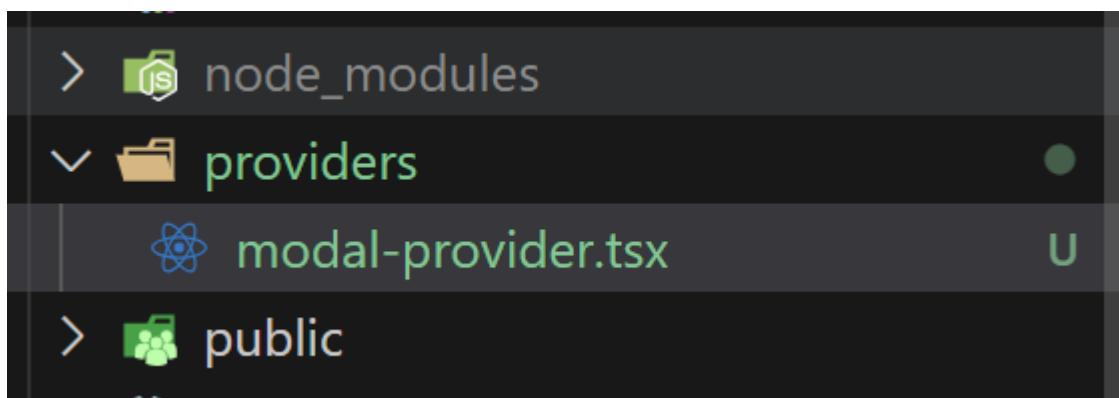
8. membuat folder modals dalam folder components, kemudian membuat file store-modal.tsx dalam folder modals



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`.

```
1  "use client"
2
3  // Mengimpor hook `useStoreModal` yang digunakan untuk mengakses state global modal dari Zustand.
4  import { useStoreModal } from "@/hooks/use-store-modal"
5  // Mengimpor komponen `Modal` dari path lokal. Komponen ini bertugas untuk menampilkan dialog/modal.
6  import { Modal } from "../ui/modal";
7
8  // Mendefinisikan komponen fungsional React bernama `StoreModal`.
9  export const StoreModal = () => {
0    // Mengakses state dan fungsi dari store modal menggunakan hook `useStoreModal`.
1    const storeModal = useStoreModal();
2
3    return(
4      <Modal
5        title="create store"
6        description="Add a new store to manage products and categories"
7        isOpen={storeModal.isOpen}
8        onClose={storeModal.onClose}
9      >
10        future Create Sore Form
11        {/* Konten modal yang akan muncul di dalamnya. Saat ini berupa placeholder teks,
12           tetapi akan digantikan dengan form atau elemen lain di masa mendatang. */}
13    </Modal>
14  }
15 }
```

10. membuat folder providers kemudian didalamnya 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.

```

ecommerce-visionvortex / renders / ... / modal-provider.tsx / ...
1  "use client"
2  // Mengimpor komponen `StoreModal` yang bertugas untuk menampilkan modal untuk menambahkan toko.
3  import { StoreModal } from "@components/modals/store-modal";
4  // Mengimpor hook `useEffect` dan `useState` dari React untuk mengelola efek samping dan state lokal.
5  import { useEffect, useState } from "react"
6
7  // `ModalProvider` yang bertugas mengelola render modal di sisi Klien.
8  export const ModalProvider = () => [
9      // State `isMounted` untuk mengecek apakah komponen sudah ter-mount.
10     const [isMounted, setIsMounted] = useState(false);
11
12     useEffect(() => {
13         // Mengubah `isMounted` menjadi `true` setelah komponen ter-mount di klien.
14         setIsMounted(true)
15     }, []);
16
17     if(!isMounted){
18         // Tidak merender komponen hingga ter-mount di klien.
19         return null;
20     }
21
22     return(
23         <>
24             {/* Render `StoreModal` setelah komponen ter-mount. */}
25             <StoreModal/>
26         </>
27     );
28 ];

```

12. memanggil ModalProvider pada halaman layout.tsx

```

ecommerce-visionvortex / app / ... / layout.tsx / ...
1  import type { Metadata } from "next";
2  import { Geist, Geist_Mono } from "next/font/google";
3  import { ClerkProvider } from "@clerk/nextjs";
4
5  import { ModalProvider } from "@providers/modal-provider";
6  import "./globals.css";
7
8
9  > const geistSans = Geist({
10    });
11
12 > const geistMono = Geist_Mono({
13    });
14
15 export const metadata: Metadata = {
16     title: "Admin Dashboard",
17     description: "Admin Dashboard",
18 };
19
20 export default function RootLayout({
21     children,
22 }: Readonly<{
23     children: React.ReactNode;
24 }>) {
25     return (
26         <ClerkProvider>
27             <html lang="en">
28                 <body className={`${geistSans.variable} ${geistMono.variable} antialiased`}>
29                     <ModalProvider/>
30                     {children}
31                 </body>
32             </html>
33         </ClerkProvider>
34     );
35 }
36
37
38
39

```

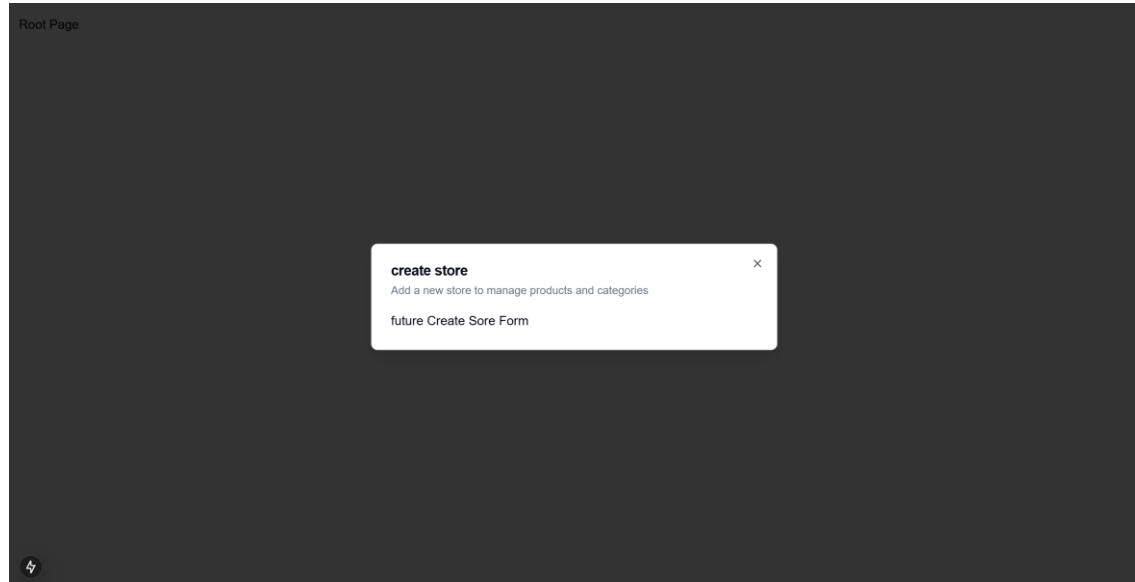
13. Mengubah dan menambahkan program pada file page.tsx yang terdapat pada folder (root).

```

ecommerce-visionvortex > app > (root) > page.tsx > [o] setUpPage
1  "use client"
2
3 // Mengimpor hook `useStoreModal` untuk mengakses store modal.
4 import { useStoreModal } from "@/hooks/use-store-modal";
5
6 import { useEffect } from "react";
7
8
9 // Fungsi `setUpPage` adalah komponen fungsional React yang digunakan untuk merender halaman.
10 const setUpPage = () => {
11
12 // Mengambil fungsi `onOpen` dan status `isOpen` dari store `useStoreModal`.
13 const onOpen = useStoreModal((state) => state.onOpen);
14 const isOpen = useStoreModal((state) => state.isOpen);
15
16 // Menggunakan `useEffect` untuk membuka modal jika `isOpen` bernilai false.
17 useEffect(() => {
18   if(!isOpen){
19     onOpen();
20   }
21 }, [isOpen,onOpen]);
22
23
24 return [
25   <div className="p-4">
26     {/* Konten dari halaman, menampilkan teks "Root Page". */}
27     Root Page
28   </div>
29 ]
30
31 }
32
33 // Mengekspor `setUpPage` sebagai default, sehingga komponen ini dapat digunakan di file lain.
34 export default setUpPage

```

14. Tampilan ketika program dijalankan



BACKEND PRISMA, NEON DAN PostgreSQL

1. menginstal prisma

```
npm install -D prisma
```

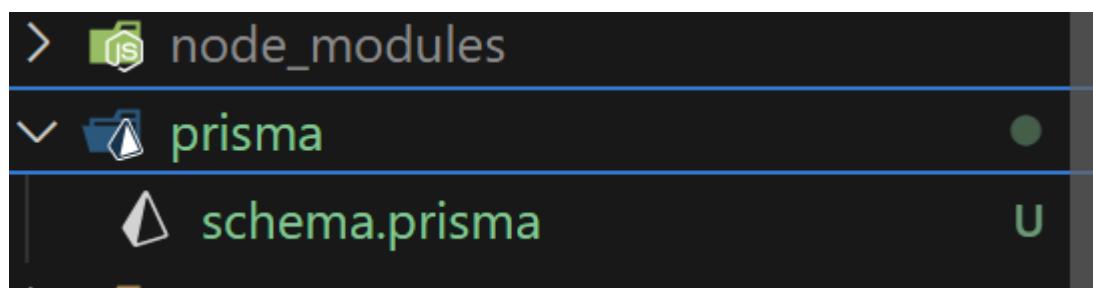
2. menginstal prisma client

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

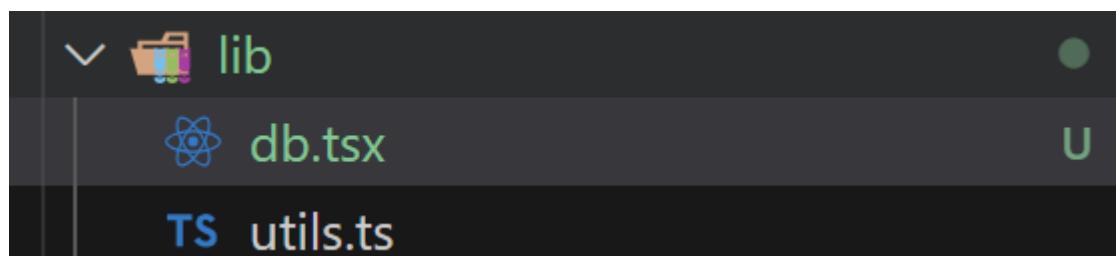
3. menginstal prisma init

```
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

```
e-commerce-visionvortex > lib > db.tsx > ...
1 import {PrismaClient} from "@prisma/client"
2
3 // Mendeklarasikan variabel global bernama `prisma`
4 declare global{
5   var prisma: PrismaClient | undefined
6 }
7
8 // Membuat instans PrismaClient
9 const db = global.globalThis.prisma || new PrismaClient();
10 // Menyimpan instans PrismaClient ke `globalThis.prisma` |
11 if (process.env.NODE_ENV !== "production") globalThis.prisma = db
12
13 export default db;
14
```

6. membuat akun dan database di neon.tech

The screenshot shows the Neon.tech web interface for creating a new project. The page has a dark background with white text and light gray input fields.

Project name: VisionVortex

Postgres version: 17

Database name: website-admin

Cloud Service Provider: AWS (selected)

Region: US East (Ohio)

Note: Select the region closest to your application.

Advanced options: A collapsed section indicated by a downward arrow.

Create project button at the bottom left.

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

Connect to your database

```
schema.prisma .env
```

```
DATABASE_URL="postgresql://website-admin_owner:*****@ep-floral-math-a51kpte1-pooler.us-east-2.aws.neon.tech/website-admin?sslmode=require"
# uncomment next line if you use Prisma <5.10
# DATABASE_URL_UNPOOLED="postgresql://website-admin_owner:*****@ep-floral-math-a51kpte1.us-east-2.aws.neon.tech/website-admin?sslmode=require"
```

Show password Copy snippet

```
e-commerce-visionorrex ✘ ✘ .env
```

```
1 NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_test_bmVlZGVkLW1hcm1vc2V0LTU3LmNsZXJrLmFjY291bnRzLmRldiQ
2 CLERK_SECRET_KEY=sk_test_lG9pJBU4QybNOWUCcyXgNzxXFQ4cdUYDJj42PifhU
3 NEXT_PUBLIC_CLERK_SIGN_IN_URL=/sign-in
4 NEXT_PUBLIC_CLERK_SIGN_UP_URL=/sign-up
5
6 # This was inserted by `prisma init`:
7 # Environment variables declared in this file are automatically made available to Prisma.
8 # See the documentation for more detail: https://pris.ly/d/prisma-schema#accessing-environment-variables-from-the-schema
9
10 # Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL Server, MongoDB and CockroachDB.
11 # See the documentation for all the connection string options: https://pris.ly/d/connection-strings
12
13 DATABASE_URL="postgresql://website-admin_owner:9XUcLImZH7aA@ep-floral-math-a51kpte1-pooler.us-east-2.aws.neon.tech/website-admin?sslmode=require"
14
15 # uncomment next line if you use Prisma <5.10
16 # DATABASE_URL_UNPOOLED="postgresql://website-admin_owner:9XUcLImZH7aA@ep-floral-math-a51kpte1.us-east-2.aws.neon.tech/website-admin?sslmode=require"
```

8. Membuat model database pada schema.prisma

```
5
6   ↴ model Store{
7     id String @id @default(uuid())
8     name String
9     userId String
10    createdAt DateTime @default(now())
11    updatedAt DateTime @updatedAt
12  }
```

Lalu ketik npx prisma generate pada terminal

```
npx prisma generate
```

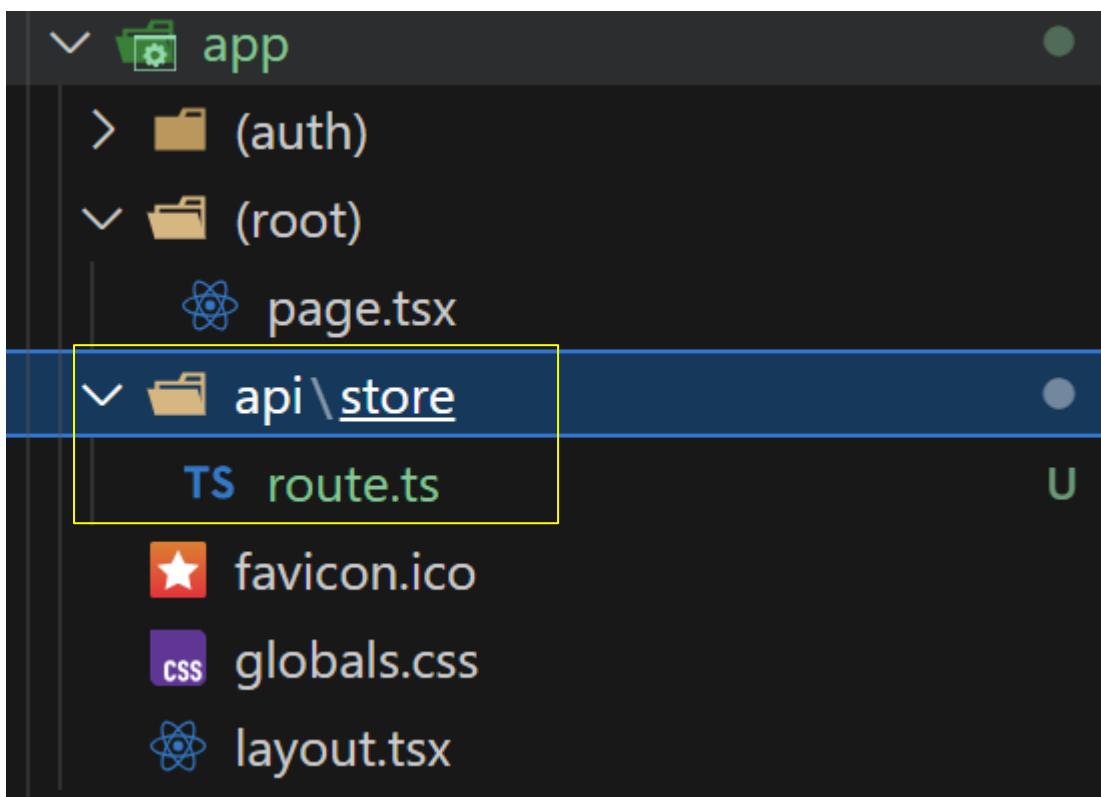
Push schema ke dalam database

```
npx prisma db push
```

Jika berhasil maka didalam web neon akan seperti ini

The screenshot shows the di.ms VisionVortex application interface. On the left, there's a sidebar with sections for PROJECT (Dashboard, Branches, SQL Editor, Restore, Monitoring, Integrations, Authorize, Settings, Quickstart), BRANCH (Overview, Tables), and Feedback. The 'Tables' section is currently selected. In the main area, there's a 'Tables' header with various filters and a search bar. Below it, there's a list of tables: 'website-admin' (selected), 'public', and 'Store'. The 'Store' table is highlighted with a yellow box. At the bottom right, it says 'No rows limit 50 offset 0'.

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
1 import { auth } from "@clerk/nextjs/server"
2 import { NextResponse } from "next/server"
3
4 export async function POST(req: Request){
5     try{
6         const { userId} = await auth()
7         const body = await req.json();
8
9         const {name} = body
10
11     }catch (error){
12         console.log("[STORES_POST]", error)
13         return new NextResponse("Internal error", {status: 500})
14     }
15 }
```

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

```
// 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

```
// 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.

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

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

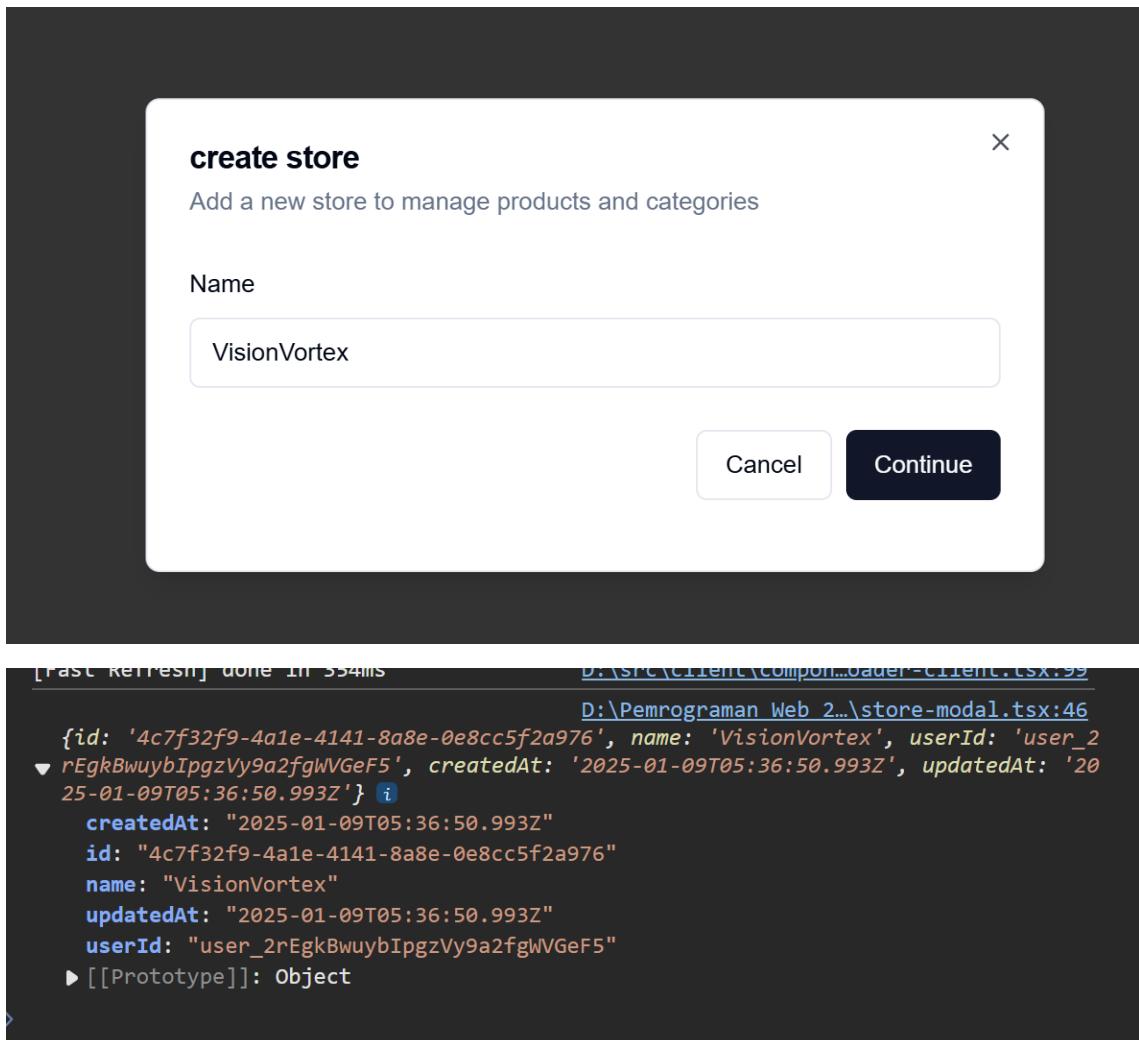
  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      name: "",
    },
  });

  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)
    }
  }
}
```

Menambahkan property disabled

```
<FormItem>
  <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>
```

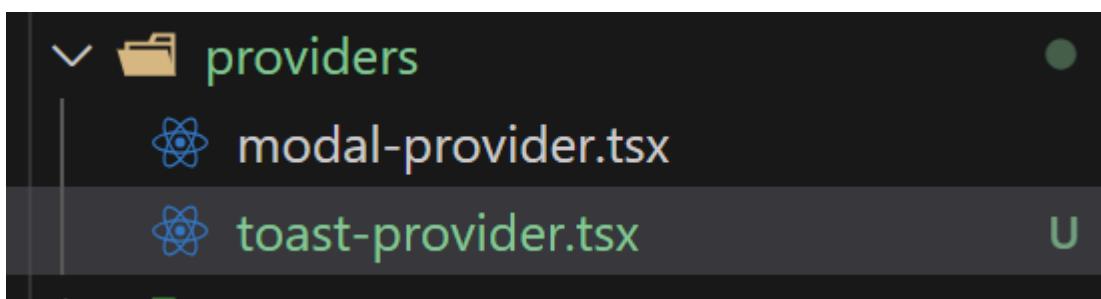
14. tampilan ketika program berhasil dijalankan



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

```
1  "use client"
2
3  import {Toaster} from "react-hot-toast"
4
5  export const ToasterProvider = () =>{
6      return <Toaster/>;
7  };
```

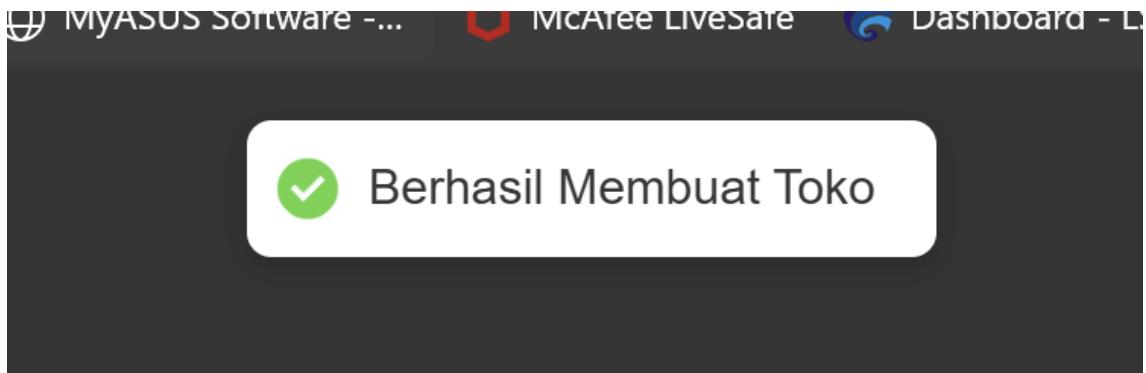
17. Memanggil ToastProvider pada file layout.tsx

```
return (
  <Clerk<T> (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

```
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 menambahkn toko:



Data Table (Admin)

1. menambahkan program pada folder billboards file page.tsx

```
    <div className="flex-col">
      <div className="flex-1 space-y-4 p-8 pt-6">
        <BillboardClient data={billboards}/>
      </div>
    </div>
  );
}

export default BillboardsPage;
```

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

```

"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"

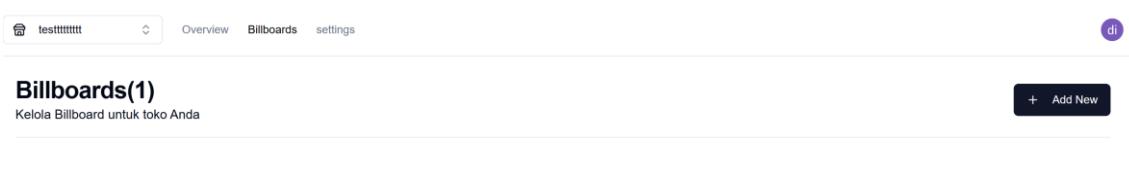
interface BillboardClientProps{
    data: Billboard[]
}

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
```

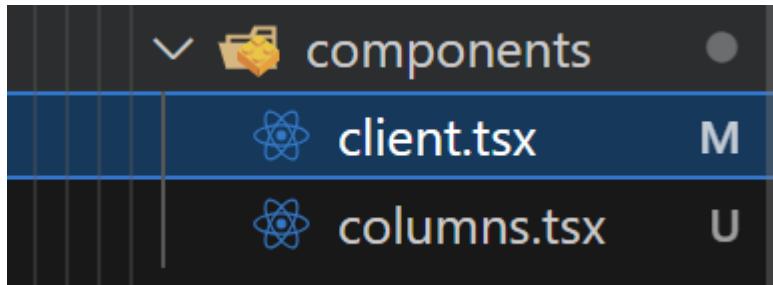
4. menginstal package tanstck

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

5. menginstall package date

```
npm i date-fns
```

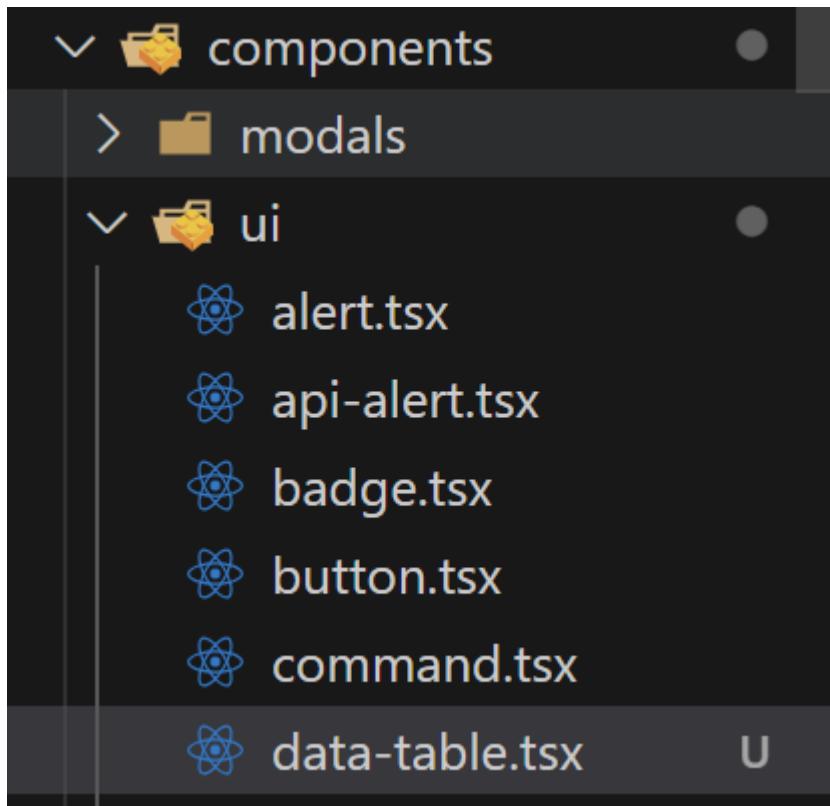
6. membuat file column.tsx pada folder components billboards



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

```
1  "use client"
2
3  import { ColumnDef } from "@tanstack/react-table"
4
5  // This type is used to define the shape of our data.
6  // You can use a Zod schema here if you want.
7  export type BillboardColumn = {
8    id: string
9    label: string
10   createdAt: string
11 }
12
13 export const columns: ColumnDef<BillboardColumn>[] = [
14  {
15    accessorKey: "label",
16    header: "Label",
17  },
18  {
19    accessorKey: "createdAt",
20    header: "Date",
21  },
22]
23
```

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



9. menambahkan program pada file data-table.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>
        ))
      </TableHeader>
      <TableBody>
        {table.getRowModel().rows.map((row) => (
          <TableRow key={row.id}>
            {row.cells.map((cell) => (
              <TableCell key={cell.id}>
                {cell.getRenderValue()}
              </TableCell>
            ))}
          </TableRow>
        ))
      </TableBody>
    </Table>
  </div>
)
```

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

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

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

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

```
  |> </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. Menambahkan program di file data-table.tsx untuk menampilkan filter

Mengimport input dari components/ui/input

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

Mengimport ColumnFiltersState, dan getFilteredRowModel dari shadcnui.com

```
5  import {
6    ColumnDef,
7    ColumnFiltersState,
8    flexRender,
9    getCoreRowModel,
10   getFilteredRowModel,
11   getPaginationRowModel,
12   useReactTable,
13 } from "@tanstack/react-table"
14
```

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

Membuat fungsi searchkey untuk searching

```
7  data: TData[],
8  searchKey: string;
```

membuat fungsi columnFilters dan setColumnFilters menggunakan useState

```
33  data,
34  searchKey,
35  ): DataTableProps<TData, TValue> {
36  const [columnFilters, setColumnFilters] = useState<ColumnFiltersState>(
37    []
38  )
39
```

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}/>  
</>
```

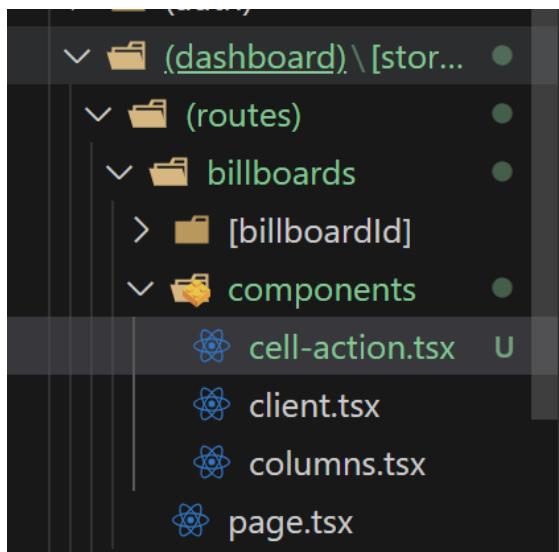
Tampilan ketika program dijalankan

Billboards(3)
Kelola Billboard untuk toko Anda

Label	Date
Test3	January 13th, 2025

Previous Next

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



Menambahkan 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  };
```

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

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

Tampilan ketika program dijalankan

The screenshot shows a web application interface titled 'Billboards(3)'. At the top right is a button labeled '+ Add New'. Below the title is a search bar with the placeholder 'Search'. A table follows, with columns for 'Label', 'Date', and 'Action'. The data rows are:

Label	Date	Action
Test3	January 13th, 2025	Action
TEST2	January 13th, 2025	Action
TEST	January 13th, 2025	Action

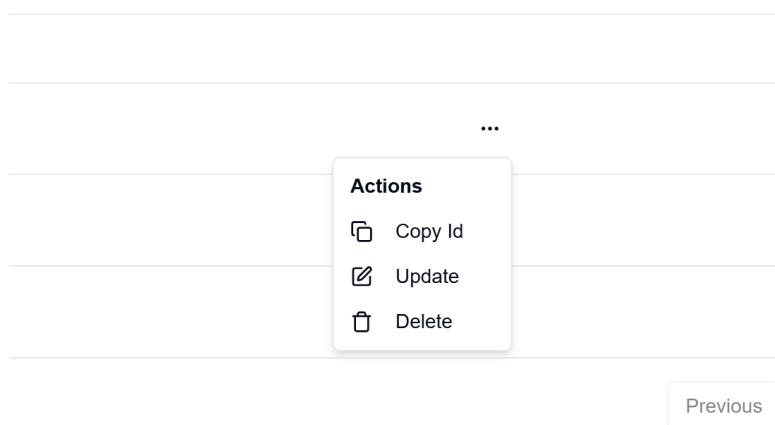
Menginstal components dropdown menu dari shadcn ui

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

Mengedit Program Action dengan menambahkan 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")
  };
}

export {
```

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

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

```
<| import DropdownMenuItem
<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 tambahkan fungsi onClick dengan memnggil setOpen yang nilainya true

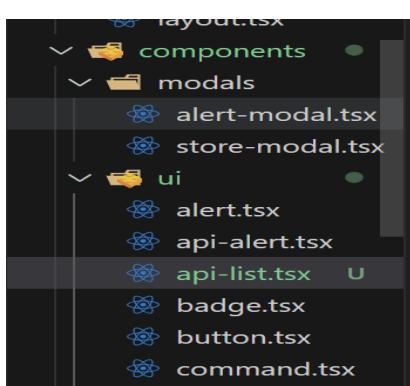
```

<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



```
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.).

```
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`,

```
    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`

The screenshot shows a Swagger UI interface for an API. The title is "API" and the subtitle is "API calls for Billboards". There are five listed endpoints:

- GET public: http://localhost:3000/ap/64914757-3813-4da0-9c2e-d0493b067035/billboards
- GET public: http://localhost:3000/ap/64914757-3813-4da0-9c2e-d0493b067035/billboards/{billboardId}
- POST admin: http://localhost:3000/ap/64914757-3813-4da0-9c2e-d0493b067035/billboards
- PATCH admin: http://localhost:3000/ap/64914757-3813-4da0-9c2e-d0493b067035/billboards/{billboardId}
- DELETE admin: http://localhost:3000/ap/64914757-3813-4da0-9c2e-d0493b067035/billboards/{billboardId}

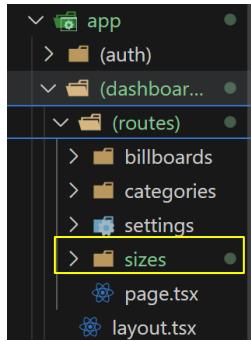
Tampilan ketika program dijalankan.

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
}

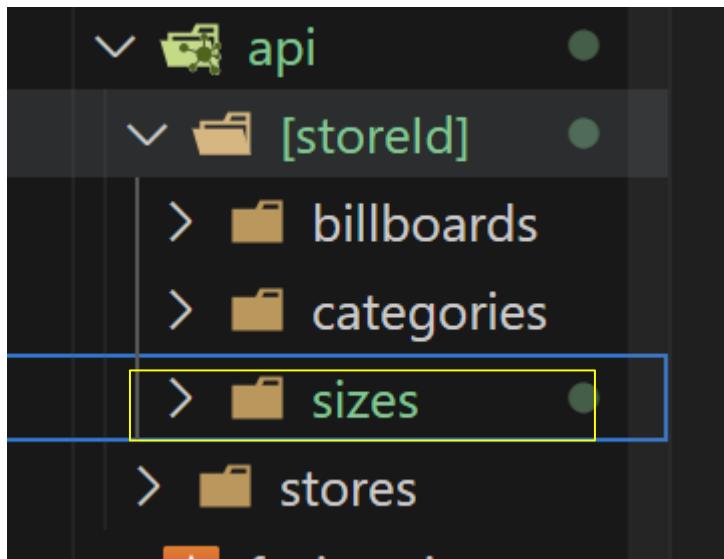
export const columns: ColumnDef<SizeColumn>[] = [
  {
    accessorKey: "name",
    header: "Name",
  },
  {
    accessorKey: "value",
    header: "Value",
  },
  {
    accessorKey: "createdAt",
    header: "Date",
  },
  {
    id: "Actions",
    cell: ({row}) => <CellAction data={row.original}>/>
  }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

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.

```
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})
  }
}

```

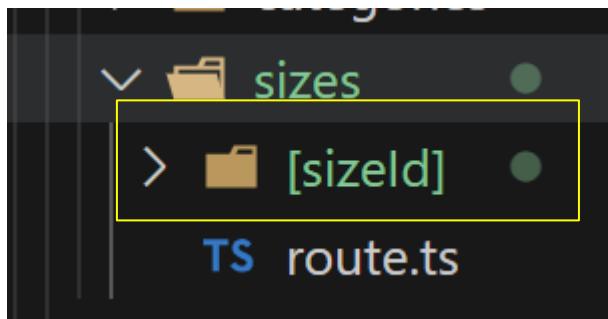
```

export 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

```
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 });
}
}
```

```
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 sizeld menjadi seperti diatas

The screenshot shows a web interface for managing sizes. At the top, there's a header "Sizes(2)" and a button "+ Add New". Below is a search bar and a table with columns: Name, Value, and Date. Two entries are listed: "Large" with value "L" and date "January 21st, 2025", and "Medium" with value "M" and date "January 21st, 2025". At the bottom right are "Previous" and "Next" buttons.

API

API calls for Sizes

GET public

<http://localhost:3000/api/64914757-3813-4da0-9c2e-d0493b067035/sizes>

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

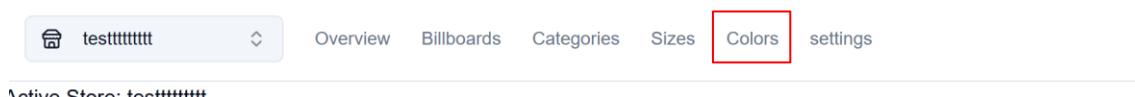
    @@index([storeId])
}
```

Tambahakan color untuk relasi pada model store sebagai berikut:

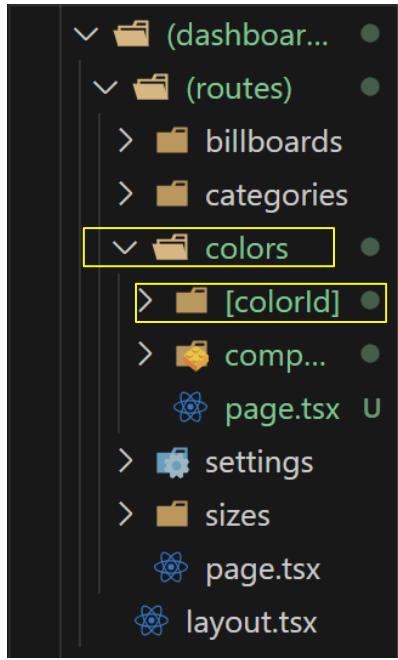
```
model Store{
    id String @id @default(uuid())
    name String
    userId String
    billboards Billboard[] @relation("StoreToBillboard")
    categories Category[] @relation("StoreToCategory")
    sizes Size[] @relation("StoreToSize")
    colors Color[] @relation("StoreToColor")
    createdAt DateTime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

Tambahkan 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 seperti berikut:

```
app > (dashboard) > (routes) > colors > [colorId] > page.tsx > ColorsPage
1  import { format, formatDate } from "date-fns"
2
3  import db from "@/lib/db";
4  import { SizesClient } from "./components/client";
5  import { SizeColumn } from "./components/columns";
6
7  const ColorsPage = async ({  
8    params  
9  }: {  
10    params: {storeId: string}  
11  }) => {  
12    const colors = await db.color.findMany({  
13      where: {  
14        storeId: params.storeId  
15      },  
16      orderBy: {  
17        createdAt: 'desc'  
18      }  
19    });  
20    const formattedColors: SizeColumn[] = colors.map((item) => ({  
21      id: item.id,  
22      name: item.name,  
23      value: item.value,  
24      createdAt: formatDate(item.createdAt, "MMMM do, yyyy")  
25    }));  
26  };  
27
28  return (  
29    <div className="flex-col">  
30      <div className="flex-1 space-y-4 p-8 pt-6">  
31        <SizesClient data={formattedColors}>/</SizesClient>  
32      </div>  
33    </div>  
34  );  
35}
36
37  export default ColorsPage;
```

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

```

1 "use client"
2
3 import { ColumnDef } from "@tanstack/react-table"
4 import { CellAction } from "./cell-action"
5
6 // This type is used to define the shape of our data.
7 // You can use a Zod schema here if you want.
8 export type ColorColumn = {
9   id: string
10  name: string
11  value: string
12  createdAt: string
13}
14
15 export const columns: ColumnDef<ColorColumn>[] = [
16  {
17    accessorKey: "name",
18    header: "Name",
19  },
20  {
21    accessorKey: "value",
22    header: "Value",
23    cell: ({ row }) => (
24      <div className="flex items-center gap-x-2">
25        {row.original.value}
26        <div className="h-6 w-6 rounded-full border" style={{backgroundColor: row.original.value}}/>
27      </div>
28    )
29  },
30  {
31    accessorKey: "createdAt",
32    header: "Date",
33  },
34  {
35    id: "Actions",
36    cell: ({row}) => <CellAction data={row.original}/>
37  }
38]

```

ubah SizeColumn menjadi ColorColumn pada file page.tsx sebelumnya :

```

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.colors.findMany({
    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;

```

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

```

9 import { ColorColumn, columns } from "./columns"
10 import { DataTable } from "@/components/ui/data-table"
11 import { ApiList } from "@/components/ui/api-list"
12
13 interface ColorsClientProps{
14   data: ColorColumn[]
15 }
16
17 export const ColorsClient: React.FC<ColorsClientProps> = ([
18   data
19 ]) => {
20   const router = useRouter();
21   const params = useParams();
22
23   return [
24     <>
25       <div className="flex items-center justify-between">
26         <Heading
27           title={`Colors(${data.length})`}
28           description="Ketola Warna untuk toko Anda"
29         />
30         <Button onClick={() => router.push(`/${params.storeId}/colors/new`)}>
31           <Plus className="mr-2 h-4 w-4"/>
32           Add New
33         </Button>
34       </div>
35       <Separator />
36       <DataTable searchKey="name" columns={columns} data={data}>
37         <Heading title="API" description="API calls for Colors"/>
38         <Separator/>
39         <ApiList entityName="colors" entityIdName="colorId"/>
40       </DataTable>
41     </>
42   ];
43 }

```

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

```

3 import db from "@lib/db";
4 import { ColorsClient } from "./components/client";
5 import { ColorColumn } from "./components/columns";
6
7 const ColorsPage = async ({
8   params
9 }: {
10   params: {storeId: string}
11 }) => {
12   const colors= await db.color.findMany({
13     where: {
14       storeId: params.storeId
15     },
16     orderBy: {
17       createdAt: 'desc'
18     }
19   });
20
21   const formattedColors: ColorColumn[] = colors.map((item) => ({
22     id: item.id,
23     name: item.name,
24     value: item.value,
25     createdAt: formatDate(item.createdAt,"MMMM do, yyyy" )
26   }));
27
28   return (
29     <div className="flex-col">
30       <div className="flex-1 space-y-4 p-8 pt-6">
31         <ColorsClient data={formattedColors}/>
32       </div>
33     </div>
34   );
35 }
36
37 export default ColorsPage;

```

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

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

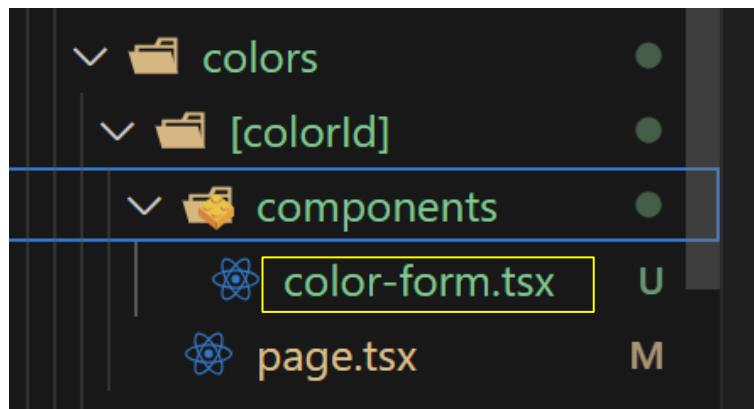
const ColorPage = async ({ params }: { params: { colorId: string } }) => {
    // tunggu params sebelum mengakses properti sizeId
    const { colorId } = await params;

    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 initialValue={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 :

```

const formSchema = z.object ({
  name: z.string(),
  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: '',
    }
  });
}

```

```

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)
  }
};

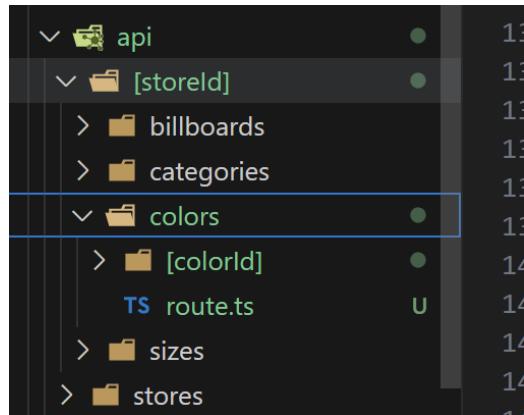
```

```

</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>
  </form>

```

Buka folder app/api/[storeId] kemudian copy folder sizes pastekan dalam folder [storeId] lalu ubah nama menjadi colors dan sizeld 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 (local var) error: unknown
    } catch (error) {
        console.log("[COLOR_GET]", error);
        return new NextResponse("Internal error", { status: 500 });
    }
}

```

```
export async function PATCH(
  req: Request,
  { 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
      }
    });
  
```

```
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 });
  }
}
```