در بخش فرانت اند از nextjs استفاده کردیم .

به همراه axios برای کال کردن api ها .

و از react-query برای نشون دادن یا post , delete , put .

که البته در فرانت صرفا register / login / logout رو داریم .

نصب و راه اندازی : (install)

با دستور زیر پروژه nextjs خودمون رو نصب می کنیم .

```

npx create-next-app@latest

```

نصب وابسته گی ها (dependencies) :

ماژول های که نیاز داریم برای این پروژه به صورت زیر هستند .

که در فایل package.json قابل مشاهده هستند :

```javascript

  "dependencies": {

    "axios": "^1.6.1",

    "next": "14.0.2",

    "react": "^18",

    "react-dom": "^18",

    "react-icons": "^4.11.0",

    "react-query": "^3.39.3"

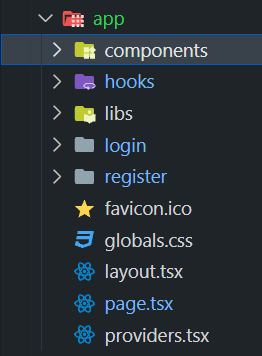
  },

```

فایل بند پروژه ( structure) :

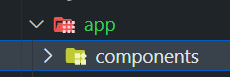
فولدر اصلی پروژه app هستش :

عکس :



**یک فولدر به اسم components داریم :**

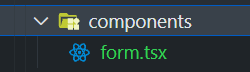
عکس



که درون فولدر components میام کامپونت های مورد نیازمون رو تعریف می کنیم .

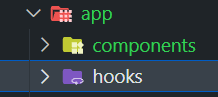
که در این پروژه یک کامپونت form درونش قرار داره .

عکس



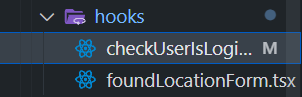
**Hook folder :**

عکس



هوک های مورد نیازمون رو درون این فولدر تعریف می کنیم .

که برای این پروژه 2 تا هوک داریم :



یکی از هوک ها وظیفه چک کردن کاربر رو داره که ایا کاربر لاگین هست یا خیر

دومین هوک وظیفه active کردن input مربوط به confrimPassword هست با توجه

به path که کامپونت form درونش امده رندر شده

**Providers.tsx :**



یک فایل به اسم providers داریم که درونش react-query قرار گرفته .

که بتونیم در layout اصلی استفاده کنیم .

به این دلیل که layout اصلی سمت سرور اجرا میشه .

و react-query سمت کلاینت برای اینکه layout اصلی رو client نکنیم

این provider رو تعریف کردیم .

کد درون providers.tsx به صورت زیر هستش :

```javascript

"use client";

import { QueryClient, QueryClientProvider } from "react-query";

import { useState } from "react";

export const Providers = ({ children }: { children: React.ReactNode }) => {

  const [client] = useState(new QueryClient());

  return <QueryClientProvider client={client}>{children}</QueryClientProvider>;

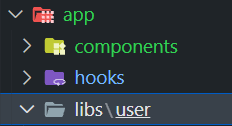
};

```

**Libs Folder :**

فولدر بعدی libs هست که درون این فولدر میایم api ها رو کال می کنیم .

عکس :

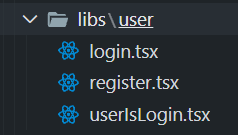


همونطور که در عکس مشخص هست یک فولدر دیگه به اسم user داریم .

که در فولدر user میایم api های مربوط به user رو فراخوانی می کنیم .

به کمک axios .

عکس :



در عکس بالا 3 تا فایل به اسم های login / register / userIsLogin مشاهده می کنید

که همونطور از اسم شون پیداست وظیفه ای به این فانکشن ها داده شده .

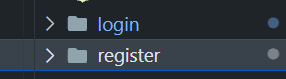
در login.tsx میایم api مربوط به login شدن کاربر کال می کنیم .

در register.tsx میایم api مربوط به register شدن کاربر رو کال می کنیم .

در userIsLogin.tsx میایم api مربوط به profile کاربر رو کال می کنیم .

Folder login / register :

عکس



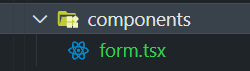
یک صحفه login داریم که در فولدر login کدش زده شده

یک صفحه register داریم که در فولدر register کدش زده شده .

هر دو صحفه یک کامپونت Form رندر می کند

که درون فولدر components ساخته شده

عکس :



**Start code :**

اول از همه بعد از نصب react-qeury نیاز داریم یک providers ایجاد کنیم .

که در بالا صحبت کردیم :



و کدی که درونش قرار داره :

```javascript

"use client";

import { QueryClient, QueryClientProvider } from "react-query";

import { useState } from "react";

export const Providers = ({ children }: { children: React.ReactNode }) => {

  const [client] = useState(new QueryClient());

  return <QueryClientProvider client={client}>{children}</QueryClientProvider>;

};

```

**استفاده از providers.tsx :**

میایم درون layout.tsx ازش استفاده می کنیم

عکس



اول import می کنیم .

```javascript

import { Providers } from "./providers";

```

حالا میایم children رو درونش wrap می کنیم :

```javascript

export default function RootLayout({

  children,

}: {

  children: React.ReactNode;

}) {

  return (

    <html lang="en">

      <body className={inter.className}>

        <Providers>{children}</Providers>

      </body>

    </html>

  );

}

```

الان هر جا بخویم از react-query استفاده کنیم در بقیه page های که داریم .

به راحتی می تونیم دسترسی داشته باشیم و اینکارو انجام بدیم .

**ایجاد Form برای register و login :**

درون فولدر components یک فایل به اسم form.tsx ایجاد می کنیم .



درون form.tsx از ‘use client’استفاده می کنیم .

به این دلیل که می خوایم از state استفاده کنیم برای ذخیره سازی value های input های که داریم .

در ادامه import های که درون form.tsx نیاز داریم رو انجام میدیم .

نکته : فانکشن های register و login که از libs ایمپورت شدن جلو تر بهشون میرسیم همینوطور hook که ایجاد کردیم برای تشخیص ادرس path اسمش useFoundPath هست .

```javascript

import { useFoundPath } from "@/app/hooks/foundLocationForm";

import { register } from "@/app/libs/user/register";

import { login } from "@/app/libs/user/login";

import React, { useState } from "react";

import { useMutation, useQueryClient } from "react-query";

import { useRouter } from "next/navigation";

```

خب یک فانکشن Form داریم

```javascript

const Form = () => { {

```

درون این فانکشن موارد زیر رو استفاده کردیم :

3 تا input داریم برای email , password, confrimPassword

یک state داریم برای گرفتن مقادیر input های که داریم

```javascript

  const [values, setValues] = useState({

    email: "",

    password: "",

    confirmPassword: "",

  });

```

از useRouter استفاده کردیم برای اینکه بعد از لاگین شدن کاربر به صحفه اصلی بره

اول import شده :

```javascript

import { useRouter } from "next/navigation";

```

```javascript

  const route = useRouter();

```

از useQueryClient استفاده کردیم که برای استفاده از react-query استفاده میشه

```javascript

import { useMutation, useQueryClient } from "react-query";

```

از useMutation هم استفاده کردیم .

زمانی که بخوایم post , delete , update رو انجام بدیم از useMutation نیاز هست

استفاده کنیم که از react-query میاد ایمپورت میشه .

```javascript

  const queryCL = useQueryClient();

```

2 تا mutation نیاز داریم یکی برای login یکی برای register .

برای login و register از متد POST استفاده می کنیم .

برای register از mutation زیر داریم استفاده می کنیم :

```javascript

  const mutationRegisterUser = useMutation(register, {

    onSuccess: () => {

      queryCL.invalidateQueries("users");

    },

  });

```

برای login شدن از mutation زیر استفاده می کنیم :

```javascript

  const mutationLoginUser = useMutation(login, {

    onSuccess: () => {

      queryCL.invalidateQueries("users");

    },

  });

```

2 تا فانکشن داریم درون کامپونت form یکی onChange هست .

و فانکشن دیگه که داریم onSubmitHandlere هستش .

که فانکشنonchangeHandler وظیفه گرفتن value های input ها رو داره .

و این value ها رو درون state میاد که داشتیم میاد تعریف می کنه .

```javascript

  const onChangeHandler = (e: any) => {

    setValues({ ...values, [e.target.name]: e.target.value });

  };

```

فانکشن onSubmitHandlere میاد مقادیر state رو ارسال می کنه به فانکشن های که مربوط به api هستند .

```javascript

  const onSubmitHandlere = (e: any) => {

    e.preventDefault();

    console.log(values);

    // call api send value to back end

    active

      ? mutationRegisterUser.mutate({ ...values })

      : mutationLoginUser.mutate({ ...values });

    route.push("/");

  };

```

در اینجا میبینید که از mutation های که بالا تر تعریف کردیم داره استفاده میشه .

و یک mutate صدا زده شده این mutate در اصل همون فانکشن های مربوط به api هستند

```javascript

  //\* user register mutation

  const mutationRegisterUser = useMutation(register, {

    onSuccess: () => {

      queryCL.invalidateQueries("users");

    },

  });

```

در کد بالا ما داریم mutation مربوط به register رو می بینیم .

که اگر دقت کنید یک register بهش پاس داده شده .

که این register در واقع فانکشن هستش .

که این فانکشن register در فولدر lib داره api مربوط به register شدن کاربر رو از

سمت بک اند کال می کنه .

و حالا درون فانکشن onSubmitHandlere وقتی گفته شده

mutationRegisterUser.mutate این mutate در واقع اشاره داره

به همون فانکشن register که در فولدر lib داره اجرا میشه

برای mutationLoginUser هم همین موضوع صدق می کنه .

دار فانکشن login رو که در فولدر lib مربوط به api لاگین شدن کاربر هست

رو کال می کنه از سمت بک اند .

یک بار دیگه onSubmitHandlere رو ببینیم :

```javascript

  const onSubmitHandlere = (e: any) => {

    e.preventDefault();

    console.log(values);

    // call api send value to back end

    active

      ? mutationRegisterUser.mutate({ ...values })

      : mutationLoginUser.mutate({ ...values });

    route.push("/");

  };

```

یک active مشاهده می کنیم که در اصل از یک hook میاد .

```javascript

    active

      ? mutationRegisterUser.mutate({ ...values })

      : mutationLoginUser.mutate({ ...values });

    route.push("/");

```

که این hook وظیفه اش چک کردن ادرسی هست که form داره درونش رندر میشه .

چک می کنه اگر داخل صحفه register بود بیاد mutation مربوط به register اتفاق

بیوفته اگر داخل صحفه login بود mutation مربوط به لاگین اتفاق بیوفته

در ادامه اگر کاربر لاگین شد به وسیله useRouter کاربر به صحفه اصلی هدایت شه

```javascript

    route.push("/");

```

درون return مربوط به ui کامپونت موارد زیر رو داریم :

```javascript

        <button> {active ? "reigser" : "login"}</button>

```

این دکمه برای submit شدن form تعریف شده .

که با توجه به path صحفه میاد اسم درونش login یا register نوشته میشه .

اگر active مقدارش true باشه به این معنی هستش که داخل صحفه register هستیم در غیراینصورت درون صحفه لاگین هستیم .

یاداوری : active از یک hook میاد که داره path رو چک می کنه که ببینه داخل چه صحفه ای هستیم برای form اگر register بود این active مقدارش true میشه و اگر نباشه مقدارش false هست .

یک جای دیگه درون return یک input داریم برای confrimPassword .

که امدیم گفتیم اگر فقط در صحفه register بودیم این فیلد input بیاد رندر شه .

در غیر اینصورت این اتفاق نیوفته که درون صحفه login این فیلد confrimPassword

رو نخواهیم داشت .

```javascript

        {active && (

          <input

            onChange={onChangeHandler}

            type="text"

            name="confirmPassword"

            id=""

            placeholder="confirmPassword"

          />

        )}

```

از فانکشن onChangeHandlere درون تمامی input ها استفاده کردیم .

```javascript

 onChange={onChangeHandler}

```

به این دلیل که قرار هست مقدار value شون رو بگیریم .

**نکته :** name که برای input ها تعریف میشه باید با مقدار key که در state تعریف شده یکی باشند .

```javascript

  name="password"

```

```javascript

   name="confirmPassword"

```

```javascript

name="email"

```

و یه نگاهی به state داشته باشیم برای یاداوری :

```javascript

  const [values, setValues] = useState({

    email: "",

    password: "",

    confirmPassword: "",

  });

```

اگر اشتباه تایپی باشه کد به مشکل می خوره و value مد نظر ذخیره نخواهد شد .

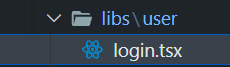
**Lib/user/:**

**login.tsx**

درون فلورد lib فانکشن login وظیفه لاگین شدن کاربر رو داره .

از این فانکشن درون کامپونت فورم استفاده کردیم .

عکس



از axios استفاده می کنیم برای کال کردن api .

```javascript

import axios from "axios";

```

نیاز به ادرس api داریم که ادرس Api ما به صورت زیر هستش و در یک متغیر url قرار گرفته

```javascript

const url = "http://localhost:2023/user";

```

یک type داریم به اسم loginParam که تایپ param های فانکشن

لاگین رو مشخص می کنه :

```javascript

type loginParam = {

  email: string;

  password: string;

  confirmPassword: string;

};

```

فانکشن login رو به صورت زیر داریم که هم export شده و هم نیاز هست از نوع async باشه .

نکته : تمامی فانکشن های مربوط به api اکسپورت شدن و از نوع async هستند .

```javascript

export const login = async ({ email, password }: loginParam) => {

  const user = await axios.post(

    `${url}/login`,

    { email, password },

    { headers: { "Content-Type": "application/json" }, withCredentials: true }

  );

  localStorage.setItem("token", JSON.stringify(user.data.data.token));

  console.log("user is login data", { data: user.data.data.token });

  return user;

};

```

اول از همه از axois.post استفاده کردیم برای فرستادن .

مقدایر email , password رو داریم از مقادیر value در کامپونت form میگیریم .

در ادامه نکته مهمی که هست این فانکشن بعد از کال شدن یک token به ما میده

که نیاز هست این توکن رو ذخیره کنیم .

چون قرار هست به وسیله این token متوجه شیم کاربر ما لاگین هست یا خیر .

تا بتونیم شرط گذاری های مربوط به لاگین بودن کاربر رو انجام بدیم .

این توکن رو در این پروژه درون local.stroge ذخیره کردیم :

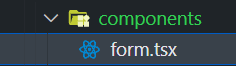
```javascript

  localStorage.setItem("token", JSON.stringify(user.data.data.token));

```

از این فانکشن login درون form.tsx که درون فولدر components هست استفاد

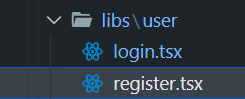
عکس :



کردیم که درون mutationLoginUser استفاده شده

**register.tsx :**

عکس

****

باز axiosرو نیاز داریم و همینطور api برای register کاربر :

```javascript

import axios from "axios";

const url = "http://localhost:2023/user";

// axios.defaults.withCredentials = true;

```

یک type داریم به اسم registerParam که داره تایپ param ها رو مشخص میکنه :

```javascript

type registerParam = {

  email: string;

  password: string;

  confirmPassword: string;

};

```

خوده فانکشن register رو داریم که از نوع async هست و همینطور export شده

```javascript

export const register = async ({

  email,

  password,

  confirmPassword,

}: registerParam) => {};

```

پارامتر های email , password , confrimPassword از کامپونت from.tsx که درون

فولدر components هست داره گرفته میشه :

عکس :



در ادامه axios.post رو داریم به همراه await .

```javascript

  const user = await axios.post(

    `${url}/register`,

    { email, password, confirmPassword },

    { headers: { "Content-Type": "application/json" } , }

  );

```

در ادامه مقادیر email , password , confrimPassword داره ارسال میشه سمت بک اند .

که در انتهای فانکشن کاربری که داره ساخته میشه یا به عبارتی دیگه متغییر user رو return شده .

```javascript

  return user;

```

یک بار به طور کلی به فانکشن register نگاه کنیم در فولدر lib/user :

```javascript

export const register = async ({

  email,

  password,

  confirmPassword,

}: registerParam) => {

  const user = await axios.post(

    `${url}/register`,

    { email, password, confirmPassword },

    { headers: { "Content-Type": "application/json" } , }

  );

  console.log("user data register", { user });

  return user;

};

```

از فانکشن register در کامپونت form استفاده شده که درون فولدر components هست

عکس :

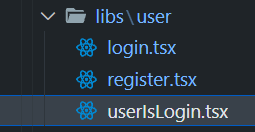


که بعد از import شدن فانکشن register در form.tsx .

درون mutationRegisterUser استفاده شده .

**userISLogin.tsx :**

عکس :

****

خب axios رو نیاز داریم و ادرس url مربوط به api .

```javascript

import axios from "axios";

const url = "http://localhost:2023/user";

```

در ادامه فانکشن userIsLogin رو داریم :

```javascript

export const userIsLogin = async () => {{

```

یک بلاک try / catch داریم .

درون بلاک try اولین خط کد گرفتن token از local stroge هست .

همون token که موقع login شدن گرفتیم و درون localstroge ذخیره کردیم .

```javascript

    const token = JSON.parse(localStorage.getItem("token") as string);

```

در ادامه یک شرط if داریم که token رو چک می کنه .

```javascript

    if (token) {

      const user = await axios.get(`${url}/profile`, {

        headers: {

          "Content-Type": "application/json",

          Authorization: `Bearer ${token}`,

        },

        // withCredentials: true, // Include this line to send the cookie

      });

      console.log("this user is login", { user });

      return user;

    }

```

اگر token وجود داشت ادامه کد اجرا شه .

اگر که این if رو نزاریم این فانکشن همچنان ران میشه و

در خواست به سمت بک اند می فرسته اگر توکنی وجود نداشته باشه

هم سمت بک هم سمت فرانت به مشکل می خوریم .

این token مهمه چون موقع کال کردن api بهش نیاز داریم .

**نیاز هست token داخل header قرار بگیره , سمت بک اند هم بهش نیاز هست .**

**و از همین headers اینجا بک اند به این token دسترسی داره .**

کدی که درون بلاک if داریم به صورت زیر هستش :

```javascript

      const user = await axios.get(`${url}/profile`, {

        headers: {

          "Content-Type": "application/json",

          Authorization: `Bearer ${token}`,

        },

        // withCredentials: true, // Include this line to send the cookie

      });

      console.log("this user is login", { user });

      return user;

```

در کد بالا داخل header یک Authoraztion داریم به اسم Bearer که توکنی که از

localstroge گرفتیم اونجا قرار داده شده .

در نهایت داریم return می کنیم خوده متغییر user رو :

```javascript

      return user;

```

یک نگاه کلی به کد فانکشن userIsLogin داشته باشیم :

```javascript

import axios from "axios";

const url = "http://localhost:2023/user";

// axios.defaults.withCredentials = true;

export const userIsLogin = async () => {

  try {

    const token = JSON.parse(localStorage.getItem("token") as string);

    console.log(token, "this token");

    if (token) {

      const user = await axios.get(`${url}/profile`, {

        headers: {

          "Content-Type": "application/json",

          Authorization: `Bearer ${token}`,

        },

        // withCredentials: true, // Include this line to send the cookie

      });

      console.log("this user is login", { user });

      return user;

    }

  } catch (err) {

    console.log(err);

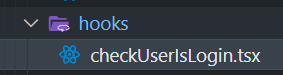
  }

};

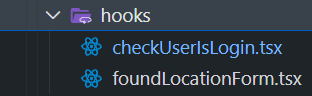
```

از فانکشن userIsLogin درون hook به اسم checkUserIsLogin.tsx استفاده کردیم

عکس :



Hooks folder :



همونطور که قبلا در بخش ساختار پروژه اشاره کردیم 2 تا هوک داریم .

که الان می خوایم هوک checkUserIsLogin.tsx رو بررسی کنیم .

در این هوک checkUserIsLogin.tsx 2 تا اتفاق می افته .

اولین اتفاق فانکشن userIsLogin اجرا میشه و مشخصات کاربر رو به ما میده .

دومین اتفاق یک فانکشن برای logout شدن داریم .

**نکته :** logout شدن در این پروژه صرفا سمت فرانت اند انجام میشه و سمت بک اند کدی نداره

از useQuery و useQueryClient استفاده شده .

نکته : مواردی که برای react-query استفاده میشه سمت کلاینت اتفاق می افته پس نیاز هست که از use client استفاده کنیم .

```javascript

"use client";

import { userIsLogin } from "../libs/user/userIsLogin";

import { useQuery, useQueryClient } from "react-query";

```

از useQuery برای GET کردن استفاده می کنیم .

از useQueryClinet برای clear کردن catch مربوط به کاربر ها که برای logout

استفاده میشه .

در ادامه یک فانکشن useIsUserLogin رو داریم که export شده .

```javascript

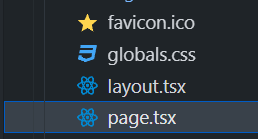
export const useIsUserLogin = () => {{

```

این فانکشن ما در اصل یک hook هست .

از این hook درون home page پروژه استفاده میشه .

عکس :



درون useIsLoginUser کد های زیر رو داریم

اول امدیم مشخصات کاربر رو به وسیله useQuery گرفتیم :

به فانکشن userIsLogin که درون libs نوشتیم نیاز داریم .

```javascript

import { userIsLogin } from "../libs/user/userIsLogin";

```

بعد به وسیله useQuery میایم از این فانکشن استفاده می کنیم .

و مشخصات کاربر رو درون data خواهیم داشت :

```javascript

  const { isLoading, isError, data, error } = useQuery({

    queryKey: ["users"],

    queryFn: userIsLogin,

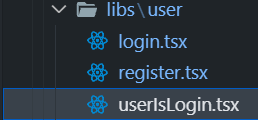
  });

```

در queryFn امدیم از userIsLogin رو قرار دادیم .

که userIsLogin فانکشن ما هست در فولدر libs/user

عکس :



در ادامه logout رو داریم که یک فانکشن هست :

اول useQueryClinet رو صدا می زنیم چون نیاز داریم که catch مربوط به user رو

بیایم clear کنیم در فانکشن logout

```javascript

const queryClient = useQueryClient();

```

در ادامه در فانکشن logout کافیه localstorge که حاوی token کاربر هست

رو بیایم remove کنیم و همین طور clear catch رو انجام بدیم

```javascript

  const logout = () => {

    // Clear the user data from the cache

    queryClient.setQueryData(["users"], null);

    localStorage.removeItem("token");

  };

```

در نهایت مواری رو که ممکنه نیازمون شه رو میایم return می کنیم :

```javascript

return { isLoading, isError, data, error, logout };

```

یک بار کل کد هوک useIsUserLoginرو با هم ببینیم :

```javascript

"use client";

import { userIsLogin } from "../libs/user/userIsLogin";

import { useQuery, useQueryClient } from "react-query";

export const useIsUserLogin = () => {

  const queryClient = useQueryClient();

  const { isLoading, isError, data, error } = useQuery({

    queryKey: ["users"],

    queryFn: userIsLogin,

  });

  console.log(data, "user is login data in hook");

  // اینو می تونیم درون لیبز قسمت ای پی ای مربوط به کاربر ها تعریف کنیم

  const logout = () => {

    // Clear the user data from the cache

    queryClient.setQueryData(["users"], null);

    localStorage.removeItem("token");

  };

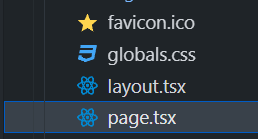
  return { isLoading, isError, data, error, logout };

};

```

استفاده : همونطور که گفتیم از این هوک درون home page استفاده میشه :

عکس :



نیاز هست اول import کنیم و اینکه نیاز هست که use client استفاده کنیم

```javascript

"use client";

import Link from "next/link";

import { useIsUserLogin } from "./hooks/checkUserIsLogin";

```

در نهایت هوک رو صدا می زنیم و مقادیر مورد نیاز رو destructuringمی کنیم :

```javascript

export default function Home() {

  const { data, logout } = useIsUserLogin();

}

```

فانکشن logout رو درون یک button که یک onClick نوشته شده استفاده می کنیم .

```javascript

onClick={() => {

        logout();

      }}

```

برای گرفتن مشخصات کاربر هم از data استفاده می کنیم

```javascript

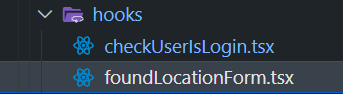
 <h1 className="text-lg">{data && data?.data?.email}</h1>

```

که داریم email کاربری که لاگین هست رو نشون میدیم .

هوک بعدی که داریم به اسم foundLocationForm هست .

عکس :



کارش این هست که میاد چک می کنه ببینه form که داریم .

در چه صحفه ای داره رندر میشه و با توجه به صحفه register یا login بودن صحفه

میاد یک state رو true / false می کنه .

که وسیله این state درون کامپونت form شرط گذاری ها انجام گرفته

برای نشون دادن input confrimPassword و همینطور برای text دکمه مربوط به

submit شدن فورم .

کد مربوط به این هوک به صورت زیر هستش :

```javascript

import { usePathname } from "next/navigation";

import { useEffect, useState } from "react";

export const useFoundPath = () => {

  const path = usePathname();

  const [active, setActive] = useState(false);

  useEffect(() => {

    if (path === "/register") {

      console.log("we in register page");

      setActive(true);

    }

  }, []);

  console.log(active, "active in useFound");

  return { active };

};

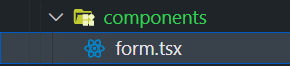
```

نکته : نیاز هست هوک ما در جای استفاده شه که use clinet باشه .

چون از useState و useEffect داریم درونش استفاده می کنیم .

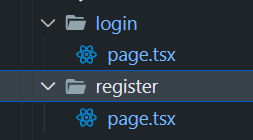
این هوک درون form.tsx استفاده شده .

عکس :



**صحفات login / register :**

عکس :



در این صحفات صحفه لاگین شدن کاربر و ثبت نام کاربر رو داریم .

که هر دو کامپونت form رو رندر می کند :

```javascript

import Form from "@/app/components/form";

import React from "react";

const Login = () => {

  return (

    <div>

      login page

      <Form />

    </div>

  );

};

export default Login;

```

برای register کد زیر رو داریم :

```javascript

import Form from "@/app/components/form";

import React from "react";

const Register = () => {

  return (

    <div>

      Register page

      <Form />

    </div>

  );

};

export default Register;

```

**خلاصه :**

2 تا هوک داشتیم که یکی وظیفه logout و گرفتن مشخصات کاربر لاگین شده رو داشت

هوک بعدی فهمیدن route که فورم درونش قرار هست رندر شه .

3 تا صحفه داشتیم register / login / home .

درون فولدر libs/user کال کردن api مربوط به user رو داشتیم به وسیله axios

درون home page و form.tsx از فانکشن های که درون libs/user نوشته بودیم

به وسیله react-query استفاده کردیم .

**پایان داکیومنت فرانت اند**