Objetivo:

- React Router;
- React Router Hooks
- localStorage.

Observação: Para fazer os exercícios é necessário ter instalado o VS Code, npm e create-react-app.

Passos para criar um aplicativo React

Acesse o local de sua preferência do computador no CMD e digite o comando a seguir. Como exemplo o aplicativo terá o nome de codigo:

```
create-react-app codigo
```

A biblioteca react-router-dom abstrai a lógica para facilitar a navegação entre os componentes no aplicativo. Acesse a pasta codigo e instale a biblioteca react-router-dom e axios:

```
npm i react-router-dom axios
```

Para estilizar o código utilizaremos o ESLint. Adicione a biblioteca eslint como dependência de desenvolvimento:

```
npm i eslint -D
```

Digite o comando a seguir para criar o arquivo .eslintrc.json. Nesse arquivo estará a nossa configuração para o ESLint:

```
npx eslint --init ou yarn run eslint --init
```

A seguir tem-se uma sugestão de conteúdo para o arquivo .eslintrc.json:

```
{
    "env": {
        "browser": true,
        "es2021": true
    },
    "extends": [
        "eslint:recommended",
        "plugin:react/recommended"
    ],
    "parserOptions": {
        "ecmaFeatures": {
            "jsx": true
        },
        "ecmaVersion": 12,
        "sourceType": "module"
    },
    "plugins": [
        "react"
    ],
    "rules": {
        "indent": [
            "error",
            "tab"
        ],
        "linebreak-style": [
```

```
            "error",
            "unix"
        ],
        "quotes": [
            "error",
            "double"
        ],
        "semi": [
            "error",
            "always"
        ]
    }
}
```

Crie o arquivo .editorconfig no VS Code e coloque as seguintes regras de formatação do código:

```
root = true

[*]
indent_style = tab
end_of_line = lf
charset = utf-8
trim_trailing_whitespace = true
insert_final_newline = true
```

Para o .editorconfig e .eslintrc funcionarem o seu VS Code é necessário ter instalado as extensões/complementos ESLint e Prettier.

i. React Router

O React Router (npm i react-router-dom) é uma biblioteca padrão para roteamento no React. O Router permite a navegação entre visualizações de componentes em um aplicativo React, permite alterar a URL do navegador mantendo a interface de usuário sincronizada com a URL. Em outras palavras, permite criar uma URL/rota para um componente.

No React Router os seguintes componentes podem ser usados parar criar a estrutura de navegação nos componentes do seu aplicativo React:

- <BrowserRouter> define o local onde as rotas devem ser definidas. Geralmente colocamos o <BrowserRouter> no componente root <App>. O componente < BrowserRouter> recebe o componente filho <Switch>:

```
<BrowserRouter>
  <Switch>
  </Switch>
</ BrowserRouter>
```

O <Router> (https://reactrouter.com/web/api/Router) é uma altenativa ao <BrowserRouter>

```
<Router>
  <Switch>
  </Switch>
</Router>
```

Diferença entre BrowserRouter e Router:

- BrowserRouter usa a history API do HTML5 para manter a interface de usuário sincronizada com a URL;
- Router é uma interface comum de baixo nível para outros componentes roteadores, por exemplo, o BrowserRouter.

- <Switch> procura nos elementos filhos <Route> um caminho que corresponde a URL e renderiza o componente definido na <Route> ignorando os demais <Route>s. No exemplo a seguir, o <Switch> poderá renderizar os componentes Usuario, Produto ou Venda. Ao receber a URL http://localhost:3000/product o <Switch> renderizará somente o componente Produto na interface do usuário:

```
< BrowserRouter>
  <Switch>
    <Route exact path="/user" component={Usuario} />
    <Route exact path="/product" component={Produto} />
    <Route exact path="/sale" component={Venda} />
  </Switch>
</ BrowserRouter>
```

- <Route> define a rota para um componente.

O componente <Link> é usado para criar um hiperlink para um componente. No exemplo <Link to="/user">Usuário</Link> será criado um hiperlink para o componente Usuario.

Para mais detalhes acesse https://reactrouter.com/web/guides/primary-components

Exemplo 1

Colar o código a seguir no componente App do aplicativo. O código cria os componentes App, Menu, Usuario, Produto e Venda. Os componentes Route criam as rotas http://localhost:3000/user, http://localhost:3000/product e http://localhost:3000/sale para os componentes Usuario, Produto e Venda, respectivamente. O componente BrowserRouter é usado para navegar entre esses componentes, ou seja, será exibido apenas um componente a cada momento e a rota http://localhost:3000 não possui conteúdo para ser mostrado no navegador. O componente <Menu /> será exibido o tempo todo por ser filho do componente <BrowserRouter>, mas não estar dentro da marcação de seleção <Switch>.

```
import React from "react";
import { BrowserRouter, Switch, Route, Link } from "react-router-dom";

export default function App() {
  return (
    <BrowserRouter>
      <Menu />
```

```
      <Switch>
        <Route exact path="/user" component={Usuario} />
        <Route exact path="/product" component={Produto} />
        <Route exact path="/sale" component={Venda} />
      </Switch>
    </BrowserRouter>
  );
}

const Usuario = () => <h4>Componente usuário</h4>;

const Produto = () => <h4>Componente produto</h4>;

const Venda = () => <h4>Componente venda</h4>;

const Menu = () => (
  <div style={{"background":"yellow"}}>
    <Link to="/user">Usuário</Link>
    <Link to="/product">Produto</Link>
    <Link to="/sale">Venda</Link>
  </div>);
```

ii. React Router Hooks

Os hooks são funções que permitem acessar o estado (state) do router e fazer a navegação usando código JavaScript. Os hooks nos provê alguns objetos:

- History (https://reactrouter.com/web/api/history): este objeto nos permite acessar o histórico de navegação do navegador e navegar usando uma URL;
- Location: este objeto possui as propriedades atuais do navegador, a URL atual.

No Exemplo 1 usamos a marcação <Link> para criar um hiperlink, no hooks podemos usar a função push do objeto history para criar um hiperlink no JavaScript, assim como é mostrado no Exemplo 2.

Para mais detalhes acesse https://reactrouter.com/web/api/Hooks

Exemplo 2

Colar o código a seguir no componente App do aplicativo. A diferença entre o código do Exemplo 1 e 2 está no uso do objeto history para navegar de um componente para outro. Usamos o objeto location para obter o caminho atual do componente que está sendo exibido pelo <Switch>.

```
import React from "react";
import { BrowserRouter, Switch, Route, useHistory, useLocation }
from "react-router-dom";

export default function App() {
  return (
    <BrowserRouter>
```

```
      <Menu />
      <Switch>
        <Route exact path="/user" component={Usuario} />
        <Route exact path="/product" component={Produto} />
        <Route exact path="/sale" component={Venda} />
      </Switch>
    </BrowserRouter>
  );
}

const Usuario = () => <h4>Componente usuário</h4>;

const Produto = () => <h4>Componente produto</h4>;

const Venda = () => <h4>Componente venda</h4>;

const Menu = () => {
    const history = useHistory();
    const location = useLocation();

    return (
        <div style={{"background":"yellow"}}>
            <div>Caminho: {location.pathname}</div>
            <button onClick={()=>history.push("/user")}>Usuário</button>
            <button onClick={()=>history.push("/product")}>Produto</button>
            <button onClick={()=>history.push("/sale")}>Venda</button>
        </div>
    );
};
```

iii. localStorage

A API Web Storage fornece acesso ao armazenamento de sessão ou armazenamento local para um domínio específico, ou seja, somente a própria aplicação consegue ler/atualizar/excluir os dados armazenados.

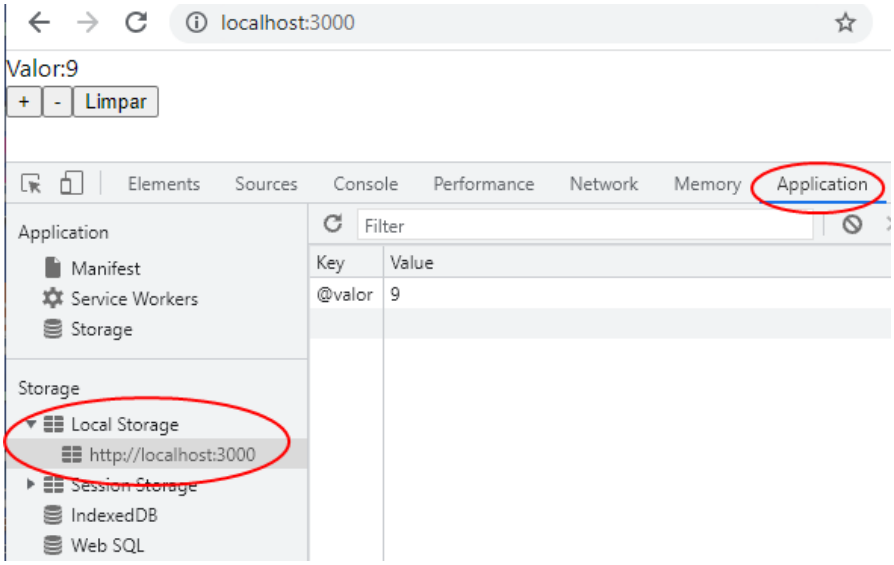
A propriedade localStorage permite acessar um objeto Storage local através dos métodos:

- setItem é usado para criar/atualizar o valor da propriedade armazenada;
- getItem é usado para ler o valor da propriedade armazenada;
- removeItem é usado para remover a propriedade armazenada.

Para mais detalhes acesse https://developer.mozilla.org/pt-BR/docs/Web/API/Storage

Exemplo 3

Colar o código a seguir no componente App do aplicativo. Ao testar o código no navegador, abra a interface do desenvolvedor no navegador para acompanhar as alterações no Local Storage.




```
import React, { useState, useEffect } from "react";

export default function App() {
    const [valor,setValor] = useState(0);

    //chamado após montar o componente
    useEffect(() =>{
        const storage = localStorage.getItem("@valor");
        if( storage ){
            setValor(JSON.parse(storage));
        }
    },[]);

    const update = (nro) => {
        setValor(nro);
        localStorage.setItem("@valor",JSON.stringify(nro));
    };

    const clear = () => {
        localStorage.removeItem("@valor");
    };

    return (
        <>
            <div>
                <div>Valor:{valor}</div>
                <div>
                    <button onClick={()=>update(valor+1)}>+</button>
                    <button onClick={()=>update(valor-1)}>-</button>
                    <button onClick={()=>clear()}>Limpar</button>
                </div>
            </div>
        </>
    );
```

```
}
```

Exemplo 4 – Fazer a conexão com o servidor da Atividade para praticar 6

Colar o código a seguir no componente App do aplicativo.

Observação: é necessário rodar o servidor criado na Atividade para praticar 6.

A aplicação possui os componentes App, Menu, Login e Gasto. As propriedades token e setToken, definidas no componente App, estão disponíveis na árvore de componentes através do objeto Context.

No componente Login é feita a requisição HTTP para o servidor, o token recebido é colocado na propriedade token (através do Context) e, na sequência, redirecionado para o componente Gasto através da rota URL /gasto.

A renderização do componente Gasto está vinculada a existência de valor na propriedade token.

Ao clicar no botão Logout a propriedade token será limpa.

```
import React, { useState, createContext, useContext } from "react";
import { BrowserRouter, Switch, Route, useHistory } from "react-router-dom";
import axios from "axios";

const Contexto = createContext();

const api = axios.create({
    baseURL: "http://localhost:3100",
});

export default function App() {
    const [token, setToken] = useState("");
    return (
        <Contexto.Provider value={{token,setToken}}>
            <BrowserRouter>
                <Menu />
                <Switch>
                    <Route exact path="/login" component={Login} />
                    <Route exact path="/gasto" component={Gasto} />
                </Switch>
            </BrowserRouter>
        </Contexto.Provider>
    );
}

const Login = () => {
    const [mail,setMail] = useState("");
    const [senha,setSenha] = useState("");
    const { setToken } = useContext(Contexto);
    const history = useHistory();

    const submeter = async (e) => {
```

```
        e.preventDefault();
        try{
            const {data} = await api.post("/api/usuario/login", { mail, senha });
            setToken(data.token);
            history.push("/gasto");
        }
        catch(e){
            alert(e.response.data.error[0]);
        }
    };

    return (
        <>
            <h4>Login</h4>
            <div>
                <label>e-mail</label>
                <input value={mail} onChange={(e)=>setMail(e.target.value)} />
            </div>
            <div>
                <label>Senha</label>
                <input type="password" value={senha}
                   onChange={(e)=>setSenha(e.target.value)} />
            </div>
            <div>
                <button onClick={submeter}>Logar</button>
            </div>
        </>
    );
};

const Gasto = () => {
    const { token } = useContext(Contexto);
    const history = useHistory();
    if( !token ){
        history.push("/login");
    }
    return (
        <>
            <h4>Componente gasto</h4>
            <div>Token: {token}</div>
        </>);
};

const Menu = () => {
    const history = useHistory();
    const { setToken } = useContext(Contexto);

    const logout = () => {
        setToken("");
```

```
    };

    return (
        <div style={{"background":"yellow"}}>
            <button onClick={()=>history.push("/login")}>Login</button>
            <button onClick={()=>history.push("/gasto")}>Gasto</button>
            <button onClick={()=>logout()}>Logout</button>
        </div>
    );
};
```

Exemplo 5 – Fazer a conexão com o servidor da Atividade para praticar 6 e manter as operações de login e logout separadas do código

Colar o código a seguir no componente App do aplicativo.

Observação: é necessário instalar a biblioteca history (https://www.npmjs.com/package/history). Ela facilita o uso da sessão history em qualquer parte do código. Neste exemplo migramos as operações de login e logout para a função hooks. Como a função hooks está fora da árvore de componentes, então não conseguimos usar o objeto history do react-router-dom.

No código substituímos o componente BrowserRouter pelo Router. A mudança foi necessária pelo fato de ao usar history.push("/gasto") a URL ser atualizada, mas a interface de usuário não ser atualizada. Esse problema não existe ao usar o componente Router.

A função createBrowserHistory é usada para construir um objeto history.

Na chamada do componente Router passamos o objeto history para os componentes filhos.

```
import React, { useState, useContext, createContext } from "react";
import { Router, Switch, Route } from "react-router-dom";
import axios from "axios";
const Context = createContext();

import { createBrowserHistory } from "history";
const history = createBrowserHistory();

export default function App() {
    const {login, logout, logado, setLogado} = Hooks();
    return (
        <Context.Provider value={{login, logout, logado, setLogado}}>
            <Router history={history}>
                <Menu />
                <Switch>
                    <Route exact path="/login" component={Login} />
                    <Route exact path="/gasto" component={Gasto} />
                </Switch>
            </Router>
        </Context.Provider>
```

```
    );
}

const api = axios.create({
    baseURL: "http://localhost:3100",
});

const Hooks = () => {
    const [logado,setLogado] = useState(false);
    const login = async (mail, senha) => {
        try{
            const {data} = await api.post("/api/usuario/login", { mail, senha });
            api.defaults.headers.Authorization = `Bearer ${data.token}`;
            setLogado(true);
            history.push("/gasto");
        }
        catch(e){
            console.log(e.message);
            alert(e.response.data.error[0]);
        }
    };

    const logout = () => {
        api.defaults.headers.Authorization = undefined;
        setLogado(false);
        history.push("/login");
    };

    return {login, logout, logado, setLogado};
};


const Login = () => {
    const [mail,setMail] = useState("");
    const [senha,setSenha] = useState("");
    const { login } = useContext(Context);

    const handle = (e) => {
        e.preventDefault();
        login(mail,senha);
    };

    return (
        <>
            <h4>Login</h4>
            <div>
                <label>e-mail</label>
                <input value={mail} onChange={(e)=>setMail(e.target.value)} />
            </div>
```

```
            <div>
                <label>Senha</label>
                <input type="password" value={senha}
                        onChange={(e)=>setSenha(e.target.value)} />
            </div>
            <div>
                <button onClick={handle}>Logar</button>
            </div>
        </>
    );
};

const Gasto = () => {
    const { logado } = useContext(Context);
    if( !logado ){
        history.push("/login");
    }
    return (
        <>
            <h4>Componente gasto</h4>
        </>);
};

const Menu = () => {
    const { logout } = useContext(Context);

    return (
        <div style={{"background":"yellow"}}>
            <button onClick={()=>history.push("/login")}>Login</button>
            <button onClick={()=>history.push("/gasto")}>Gasto</button>
            <button onClick={()=>logout()}>Logout</button>
        </div>
    );
};
```

Exemplo 6 – Alterar o Exemplo 5 para customizar as rotas e validar se o usuário está logado

A diferença desse código para o Exemplo 5 está no fato de termos criado o componente CustomRoute. Esse componente funcional checa se a rota é privada e o usuário está logado. No Exemplo 5 essa checagem estava no componente Gasto, mas aqui ela foi movida para o componente CustomRoute.

```
import React, { useState, useContext, createContext } from "react";
import { Router, Switch, Route, Redirect } from "react-router-dom";
import axios from "axios";
const Context = createContext();

import { createBrowserHistory } from "history";
const history = createBrowserHistory();
```

```
export default function App() {
    const {login, logout, logado, setLogado} = Hooks();
    return (
        <Context.Provider value={{login, logout, logado, setLogado}}>
            <Router history={history}>
                <Menu />
                <Switch>
                    <CustomRoute exact path="/login" component={Login} />
                    <CustomRoute isPrivate exact path="/gasto" component={Gasto} />
                </Switch>
            </Router>
        </Context.Provider>
    );
}

const CustomRoute = ({isPrivate, ...rest}) => {
    const { logado } = useContext(Context);
    if( isPrivate && !logado ){
        return <Redirect to="/login" />;
    }
    return <Route {...rest} />;
};

const api = axios.create({
    baseURL: "http://localhost:3100",
});

const Hooks = () => {
    const [logado,setLogado] = useState(false);
    const login = async (mail, senha) => {
        try{
            const {data} = await api.post("/api/usuario/login", { mail, senha });
            api.defaults.headers.Authorization = `Bearer ${data.token}`;
            setLogado(true);
            history.push("/gasto");
        }
        catch(e){
            console.log(e.message);
            alert(e.response.data.error[0]);
        }
    };

    const logout = () => {
        api.defaults.headers.Authorization = undefined;
        setLogado(false);
        history.push("/login");
    };

    return {login, logout, logado, setLogado};
```

```
};


const Login = () => {
    const [mail,setMail] = useState("");
    const [senha,setSenha] = useState("");
    const { login } = useContext(Context);

    const handle = (e) => {
        e.preventDefault();
        login(mail,senha);
    };

    return (
        <>
            <h4>Login</h4>
            <div>
                <label>e-mail</label>
                <input value={mail} onChange={(e)=>setMail(e.target.value)} />
            </div>
            <div>
                <label>Senha</label>
                <input type="password" value={senha}
                    onChange={(e)=>setSenha(e.target.value)} />
            </div>
            <div>
                <button onClick={handle}>Logar</button>
            </div>
        </>
    );
};

const Gasto = () => {
    return (<h4>Componente gasto</h4>);
};

const Menu = () => {
    const { logout } = useContext(Context);

    return (
        <div style={{"background":"yellow"}}>
            <button onClick={()=>history.push("/login")}>Login</button>
            <button onClick={()=>history.push("/gasto")}>Gasto</button>
            <button onClick={()=>logout()}>Logout</button>
        </div>
    );
};
```

Exemplo 7 – Alterar o Exemplo 6 para manter o token no localStorage

As modificações ocorreram nas funções hooks e CustomRoute.

A função de efeito colateral useEffect será executada ao carregar a função Hooks, como o tempo de leitura do localStorage pode demorar, então foi necessário criar a propriedade isLoading para sinalizar que ainda não temos o resultado do token armazenado no localStorage ao fazer uso dele na função CustomRoute.

A propriedade isLoading é usada no componente CustomRoute para evitar que a condição if(isPrivate && !logado) seja testada antes de terminar a leitura do localStorage.

Ative o console do navegador para ver o redirecionamento das rotas no componente CustomRoute. Neste exemplo foi usada a URL http://localhost:3000/gasto e existia o token no localStorage. Veja que o CustomRoute foi executado duas vezes, na 1ª vez o isLoading era true e na segunda vez foi renderizado o componente Gasto.

```
Lendo o storage. Destino: /gasto
Storage. Token:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJp
VYRlQQPbX69HSKLWjt7XpX6-rg"
Componente renderizado: /gasto
```

```
import React, { useState, useContext, createContext, useEffect } from "react";
import { Router, Switch, Route, Redirect } from "react-router-dom";
import axios from "axios";
const Context = createContext();

import { createBrowserHistory } from "history";
const history = createBrowserHistory();

export default function App() {
    const {login, logout, logado, isLoading} = Hooks();
    return (
        <Context.Provider value={{login, logout, logado, isLoading}}>
            <Router history={history}>
                <Menu />
                <Switch>
                    <CustomRoute exact path="/login" component={Login} />
                    <CustomRoute isPrivate exact path="/gasto" component={Gasto} />
                </Switch>
            </Router>
        </Context.Provider>
    );
}

const CustomRoute = ({isPrivate, ...rest}) => {
    const { logado, isLoading } = useContext(Context);

    if (isLoading) {
        console.log("Lendo o storage. Destino:", rest.path);
        return <h3>Carregando...</h3>;
    }
```

```
    if( isPrivate && !logado ){
        return <Redirect to="/login" />;
    }

    console.log("Componente renderizado:", rest.path);
    return <Route {...rest} />;
};

const api = axios.create({
    baseURL: "http://localhost:3100",
});

const Hooks = () => {
    const [logado,setLogado] = useState(false);
    const [isLoading, setIsLoading] = useState(true);

    //chamado após montar o componente
    useEffect(() =>{
        const storage = localStorage.getItem("@token");
        if( storage ){
            const token = JSON.parse(storage);
            api.defaults.headers.Authorization = `Bearer ${token}`;
            setLogado(true);
        }
        setIsLoading(false);
        console.log("Storage. Token:", storage);
    },[]);


    const login = async (mail, senha) => {
        try{
            const {data} = await api.post("/api/usuario/login", { mail, senha });
            localStorage.setItem("@token",JSON.stringify(data.token));
            api.defaults.headers.Authorization = `Bearer ${data.token}`;
            setLogado(true);
            history.push("/gasto");
        }
        catch(e){
            console.log(e.message);
            alert(e.response.data.error[0]);
        }
    };

    const logout = () => {
        setLogado(false);
        localStorage.removeItem("@token");
        api.defaults.headers.Authorization = undefined;
        history.push("/login");
    };
```

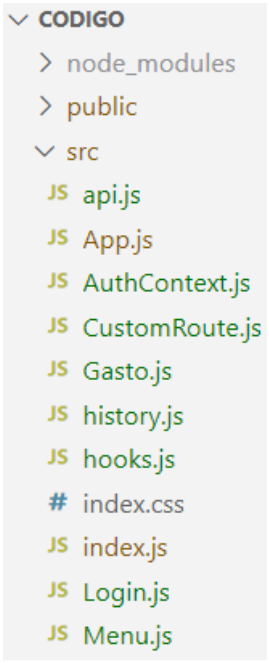
```
    return {login, logout, logado, setLogado,isLoading };
};


const Login = () => {
    const [mail,setMail] = useState("");
    const [senha,setSenha] = useState("");
    const { login } = useContext(Context);

    const handle = (e) => {
        e.preventDefault();
        login(mail,senha);
    };

    return (
        <>
            <h4>Login</h4>
            <div>
                <label>e-mail</label>
                <input value={mail} onChange={(e)=>setMail(e.target.value)} />
            </div>
            <div>
                <label>Senha</label>
                <input type="password" value={senha}
                      onChange={(e)=>setSenha(e.target.value)} />
            </div>
            <div>
                <button onClick={handle}>Logar</button>
            </div>
        </>
    );
};

const Gasto = () => {
    return (<h4>Componente gasto</h4>);
};

const Menu = () => {
    const { logout } = useContext(Context);

    return (
        <div style={{"background":"yellow"}}>
            <button onClick={()=>history.push("/login")}>Login</button>
            <button onClick={()=>history.push("/gasto")}>Gasto</button>
            <button onClick={()=>logout()}>Logout</button>
        </div>
    );
};
```

Exemplo 8 – Alterar o Exemplo 7 para organizar os componentes e funções em arquivos/módulos separados

No arquivo history.js colocamos o código para obter e exportar o objeto history. Centralizar esse código num módulo foi necessário para ele poder ser importado nos demais módulos.

Da mesma forma foi necessário criar o arquivo AuthContext para exportar o objeto Context e componente AuthProvider, que na prática é o Context.Provider que envolve a árvore de componentes do aplicativo. O objeto children possui os componentes filhos da marcação <AuthProvider> que está no módulo App.

A seguir tem-se o código dos arquivos.



Código do arquivo src/api.js

```
import axios from "axios";

const api = axios.create({
    baseURL: "http://localhost:3100",
});

export default api;
```

Código do arquivo src/App.js

```
import React from "react";
import { Router, Switch } from "react-router-dom";
import { AuthProvider } from "./AuthContext";
import Menu from "./Menu";
import Login from "./Login";
import Gasto from "./Gasto";
import CustomRoute from "./CustomRoute";
import history from "./history";

export default function App() {
    return (
        <AuthProvider>
            <Router history={history}>
                <Menu />
                <Switch>
                    <CustomRoute exact path="/login" component={Login} />
                    <CustomRoute isPrivate exact path="/gasto" component={Gasto} />
```

```
                </Switch>
            </Router>
        </AuthProvider>
    );
}
```

Código do arquivo src/AuthContext.js

```
import React, { createContext } from "react";
import hooks from "./Hooks";
const Context = createContext();

function AuthProvider({ children }) {
    const {login, logout, logado, isLoading} = Hooks();
    return (
        <Context.Provider value={{login, logout, logado, isLoading}}>
            {children}
        </Context.Provider>
    );
}

export { Context, AuthProvider };
```

Código do arquivo src/CustomRoute.js

```
import React, { useContext } from "react";
import { Route, Redirect } from "react-router-dom";
import { Context } from "./AuthContext";

const CustomRoute = ({isPrivate, ...rest}) => {
    const { logado, isLoading } = useContext(Context);

    if (isLoading) {
        console.log("Lendo o storage. Destino:", rest.path);
        return <h3>Carregando...</h3>;
    }

    if( isPrivate && !logado ){
        return <Redirect to="/login" />;
    }

    console.log("Componente renderizado:", rest.path);
    return <Route {...rest} />;
};

export default CustomRoute;
```

Código do arquivo src/Gasto.js

```
import React from "react";

const Gasto = () => {
    return (<h4>Componente gasto</h4>);
};

export default Gasto;
```

Código do arquivo src/history.js

```
import { createBrowserHistory } from "history";

export default createBrowserHistory();
```

Código do arquivo src/Hooks.js

```
import { useState, useEffect } from "react";
import history from "./history";

const Hooks = () => {
    const [logado,setLogado] = useState(false);
    const [isLoading, setIsLoading] = useState(true);

    //chamado após montar o componente
    useEffect(() =>{
        const storage = localStorage.getItem("@token");
        if( storage ){
            const token = JSON.parse(storage);
            api.defaults.headers.Authorization = `Bearer ${token}`;
            setLogado(true);
        }
        setIsLoading(false);
        console.log("Storage. Token:", storage);
    },[]);


    const login = async (mail, senha) => {
        try{
            const {data} = await api.post("/api/usuario/login", { mail, senha });
            localStorage.setItem("@token",JSON.stringify(data.token));
            api.defaults.headers.Authorization = `Bearer ${data.token}`;
            setLogado(true);
            history.push("/gasto");
        }
        catch(e){
            console.log(e.message);
            alert(e.response.data.error[0]);
        }
    };
```

```
    const logout = () => {
        setLogado(false);
        localStorage.removeItem("@token");
        api.defaults.headers.Authorization = undefined;
        history.push("/login");
    };

    return {login, logout, logado, setLogado,isLoading };
};

export default Hooks;
```

Código do arquivo src/Login.js

```
import React, { useState, useContext } from "react";
import { Context } from "./AuthContext";

const Login = () => {
    const [mail,setMail] = useState("");
    const [senha,setSenha] = useState("");
    const { login } = useContext(Context);

    const handle = (e) => {
        e.preventDefault();
        login(mail,senha);
    };

    return (
        <>
            <h4>Login</h4>
            <div>
                <label>e-mail</label>
                <input value={mail} onChange={(e)=>setMail(e.target.value)} />
            </div>
            <div>
                <label>Senha</label>
                <input type="password" value={senha} onChange={(e)=>setSenha(e.target.value)} />
            </div>
            <div>
                <button onClick={handle}>Logar</button>
            </div>
        </>
    );
};

export default Login;
```

Código do arquivo src/Menu.js

```
import React, { useContext } from "react";
import { Context } from "./AuthContext";
import history from "./history";

const Menu = () => {
    const { logout } = useContext(Context);

    return (
        <div style={{"background":"yellow"}}>
            <button onClick={()=>history.push("/login")}>Login</button>
            <button onClick={()=>history.push("/gasto")}>Gasto</button>
            <button onClick={()=>logout()}>Logout</button>
        </div>
    );
};

export default Menu;
```

Exercício 1 – Alterar o componente Gasto do Exemplo 8 para fazer as operações de criar e listar gastos.