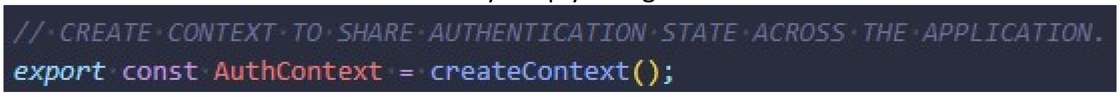
FRONTEND: AuthContext

martedì 28 novembre 2023 21:47

AUTH CONTEXT

After we create the route path for your chat app, we want to use **createContext** to make the User Authentication to be available in the whole app.
So what we do next is create a context by simply doing this:



```
// CREATE CONTEXT TO SHARE AUTHENTICATION STATE ACROSS THE APPLICATION.
export const AuthContext = createContext();
```

What now? Once we've created our **AuthContext** state, we need to set up the data that we want to share across the whole application, so our next step is create another component called **AuthContextProvider.** This component will return **AuthContext.Provider** and have as **value** all the things we want to share to our **{children}** (In this case, **AuthContextProvider** will wrap the children (component) **<App />**) in the **main.jsx** file.

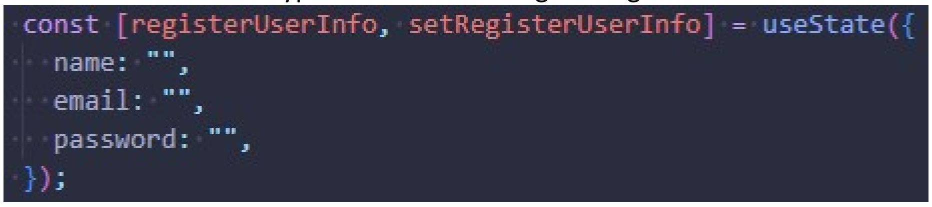
1. REGISTRATION PROCESS

What do we need to register a user?

1. **registerUserOnSubmit (useCallback),** this will send a post request to the server with the data we typed in the form during the registration. If everything proceeds correctly, the server will respond with an object containing the user data. We will set the current user with the user data we have received from the server.
2. **registerUserInfo (useState),** this state will contain an object with all the data that the user has given during the registration.
3. **updateRegisterUserInfo (useCallback),** this function will update the user data object (registerUserInfo) with the current values that the user is typing in the form during the registration.
4. **registerError** (**useState),** this state will manage the errors during the registration.
5. **isRegisterLoading (useState),** this state will update you about the status of the registration.

Now let's take a closer look of how it works.

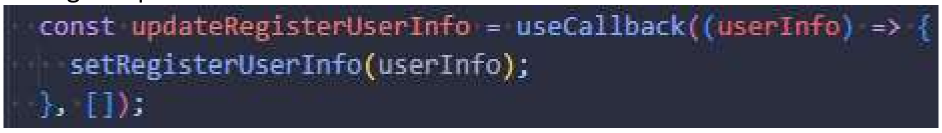
First of all, let's create a **useState** called **"registerUserInfo".** This will contain an object with all the data that the user will type in the form during the registration.



```
const [registerUserInfo, setRegisterUserInfo] = useState({
    name: "",
    email: "",
    password: "",
});
```

How are we going to update this object?

We're going to create **a useCallback** function called **"updateRegisterUserInfo"** that will help us during this process.

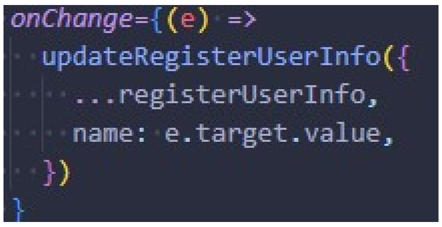


```
const updateRegisterUserInfo = useCallback((userInfo) => {
    setRegisterUserInfo(userInfo);
}, []);
```

How is it working?

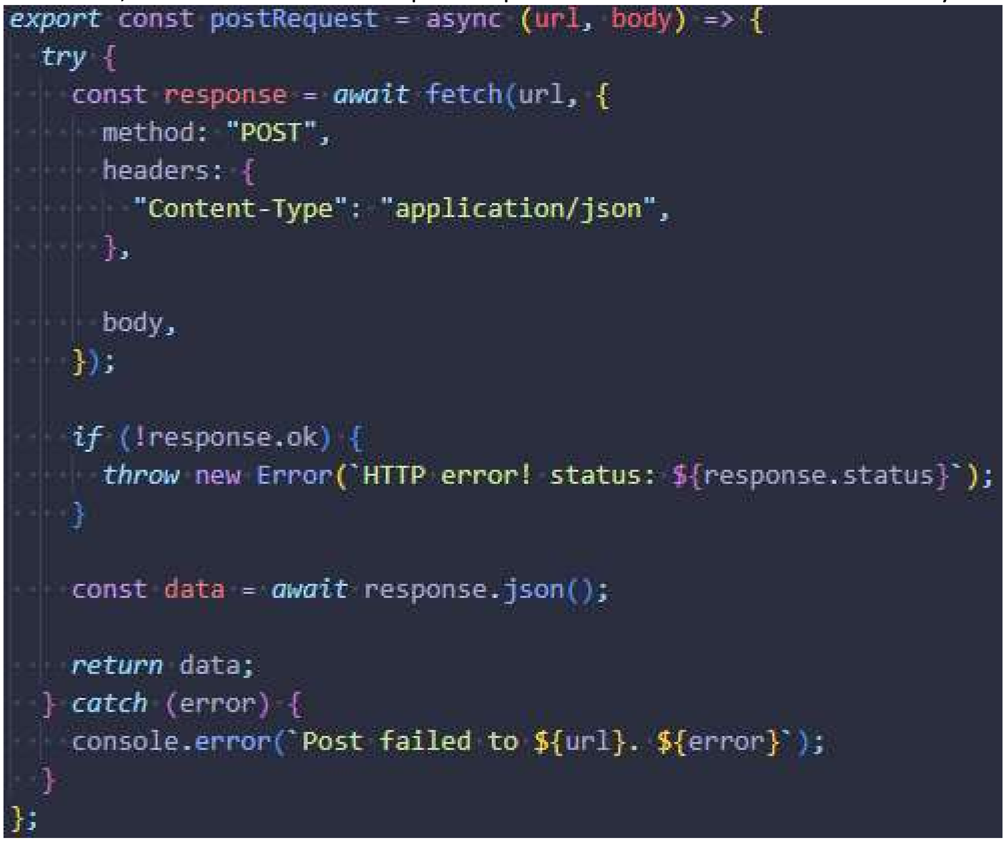
Let's say we're passing this function to the **onChange** event of the **form name input,** this means that every time the user types something in this name field, the **onChange** event will trigger the **updateRegisterUserInfo** function that will update the **registerUserInfo** state with the name of the user.

So, all we need to do is: in the **"name input field"** of the form, we're going to pass the **updateRegisterUserInfo** to the **onChange event.** We'll spread the other info that the user typed, and add the new ones with **name: e.target.value**



```
onChange={(e) =>
  updateRegisterUserInfo({
    ...registerUserInfo,
    name: e.target.value,
  })
}
```

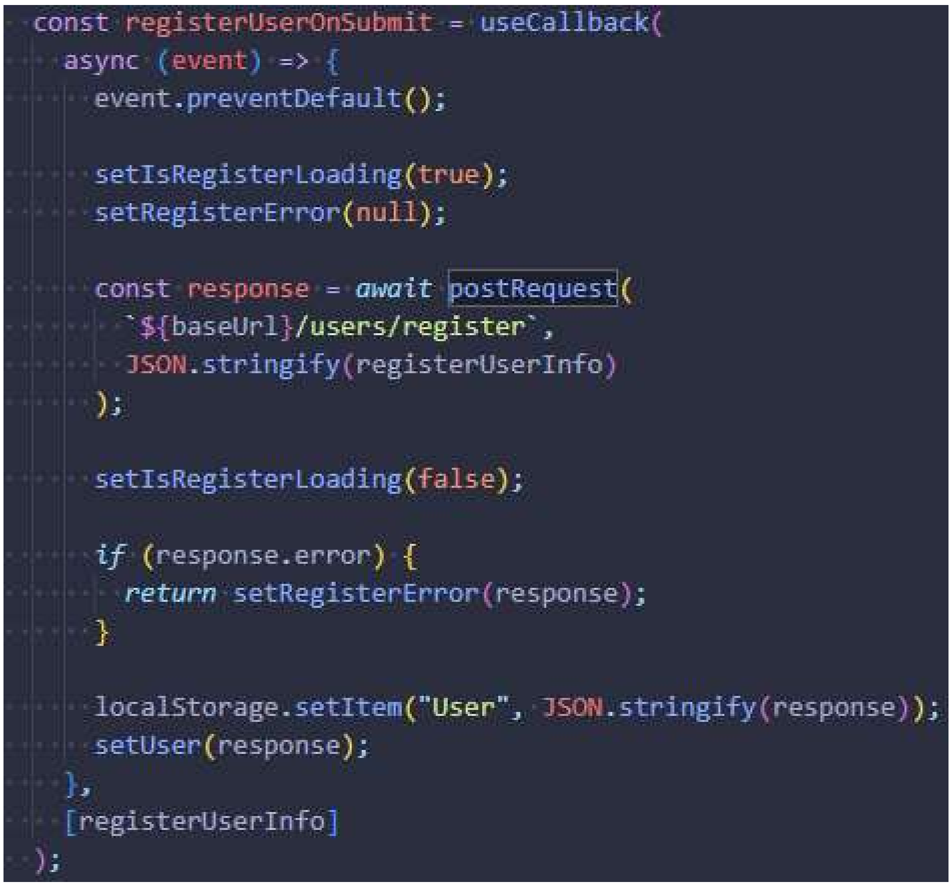
How are we going to send these info to the server?
First of all, let's create our service post request function that we can use for any occasion.



```
export const postRequest = async (url, body) => {
  try {
    const response = await fetch(url, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },

      body,
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();

    return data;
  } catch (error) {
    console.error(`Post failed to ${url}. ${error}`);
  }
};
```

This function (**postRequest**) is post request function that takes two parameters: **URL** and **BODY.**
The **URL** is the URL of the server where we will send the request, and **BODY** will be the data that we will send to the server in JSON format.

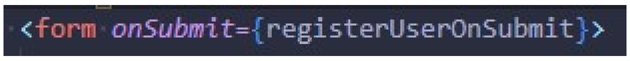
Once we've set this up, we're now going to create a post request for the registration.



```
const registerUserOnSubmit = useCallback(
  async (event) => {
    event.preventDefault();

    setIsRegisterLoading(true);
    setRegisterError(null);

    const response = await postRequest(
      `${baseUrl}/users/register`,
      JSON.stringify(registerUserInfo)
    );

    setIsRegisterLoading(false);

    if (response.error) {
      return setRegisterError(response);
    }

    localStorage.setItem("User", JSON.stringify(response));
    setUser(response);
  },
  [registerUserInfo]
);
```

The **"registerUserOnSubmit"** function orchestrates user registration, preventing page reload upon submission.
It sets initial **loading** and **errors** status**,** sending registration details to the server via **"postRequest".**
Afterward, it updates the loading and error status.
Upon successful registration, we're going to set the response of the server (the data of the registered user) in the local storage. This will keep the user logged in even if he refreshes the page. How? The app will retrieve the user info from the local storage.
Simultaneously, the current user state is updated with the data received from the server (user info), this will ensure a seamless login experience

The last thing we have to do is to pass the "**registerUserOnSubmit"** function to the **onSubmit** event of the form:



```
<form onSubmit={registerUserOnSubmit}>
```

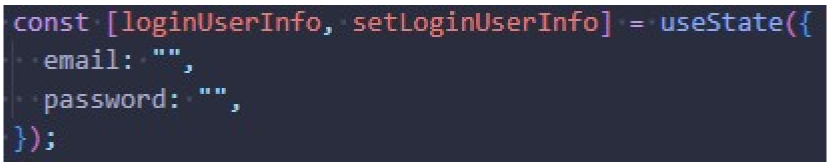
2. LOGIN PROCESS

What do we need to login a user?

1. **loginUserOnSubmit (useCallback),** this function will send a post request to the server with the data we typed in the form during the login process. If everything proceeds correctly and the user exist in the database, the server will respond with an object containing the user data. We will set the current user with the data we have received from the server.
2. **loginUserInfo (useState),** this state will contain an object with all the data that the user has given during the login process.
3. **updateLoginUserInfo (useCallback),** this function will update the user data object (loginUserInfo) with the current values that the user is typing in the form during the login process.
4. **loginError (useState),** this state will manage errors during the login.
5. **isLoginLoading (useState),** this state will update you about the status of the login.

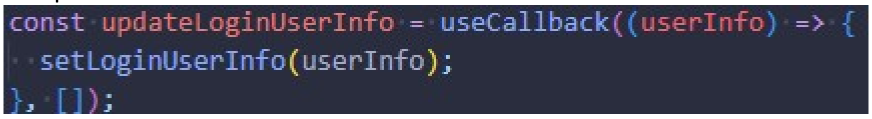
Now let's take a closer look of how it works

First of all, let's create a **useState** called **"loginUserInfo".** This will contain an object with all the data that the user will type in the form during the login process.



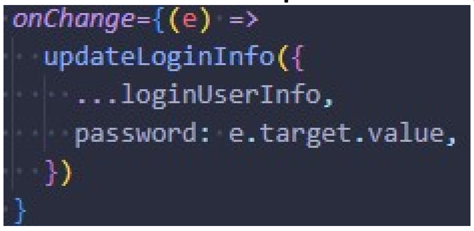
```
const [loginUserInfo, setLoginUserInfo] = useState({
  email: "",
  password: "",
});
```

How are we going to update this object?
We're going to create a **useCallback** function called **"updateLoginUserInfo",** that will help us during this process.



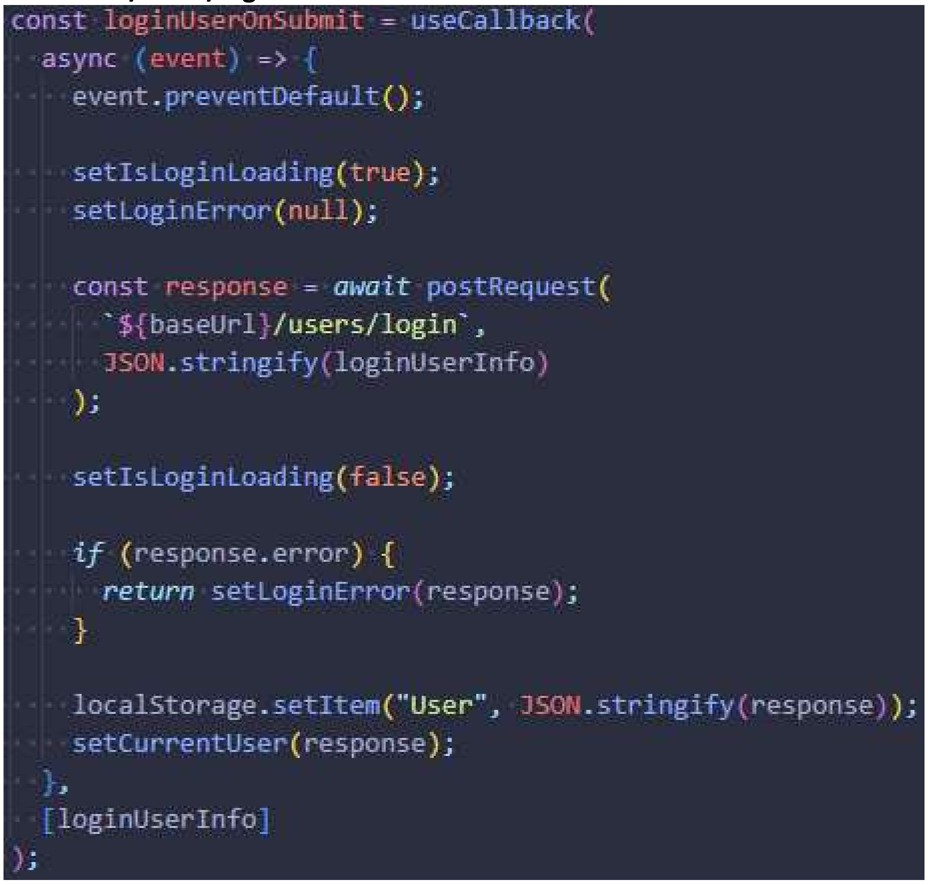
```
const updateLoginUserInfo = useCallback((userInfo) => {
  setLoginUserInfo(userInfo);
}, []);
```

How is it working?
Let's say we're passing this function to the **onChange** event of the **form password input,** this means that every time the user is about to login, and types something in this password field, the **onChange** event will trigger the **updateLoginUserInfo** function that will update the **loginUserInfo** state with the password of the user.
So, all we need to do is: in the **"password input field"** of the form, we're going to pass the **updateLoginUserInfo** to the **onChange event.** We'll spread the other info that the user typed, and add the new ones with **password: e.target.value**



```
onChange={(e) =>
  updateLoginInfo({
    ...loginUserInfo,
    password: e.target.value,
  })
}
```

How are we going to make the login request to the server?
We'll be using the same postRequest function model as before, but we're just going to change the URL to which we're going to make the request for login.
When registering a new user, we used the baseUrl/users/register but now we're going to change it to **baseUrl/users/login**

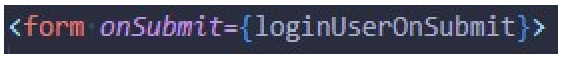


```
const loginUserOnSubmit = useCallback(
  async (event) => {
    event.preventDefault();

    setIsLoginLoading(true);
    setLoginError(null);

    const response = await postRequest(
      `${baseUrl}/users/login`,
      JSON.stringify(loginUserInfo)
    );

    setIsLoginLoading(false);

    if (response.error) {
      return setLoginError(response);
    }

    localStorage.setItem("User", JSON.stringify(response));
    setCurrentUser(response);
  },
  [loginUserInfo]
);
```

The **"loginUserOnSubmit"** function orchestrates user login, preventing page reload upon

submission.
It sets initial **loading** and **errors** status, sending login details to the server via **"postRequest".**
Afterward, it updates the loading and error status.
Upon successful login, we're going to set the response of the server (the data of the existing user) in the local storage. This will keep the user logged in even if he refreshes the page. How? The app will retried the user info from the local storage.
Simultaneously, the current user state is updated with the data received from the server (user info), this will ensure a seamless login experience.

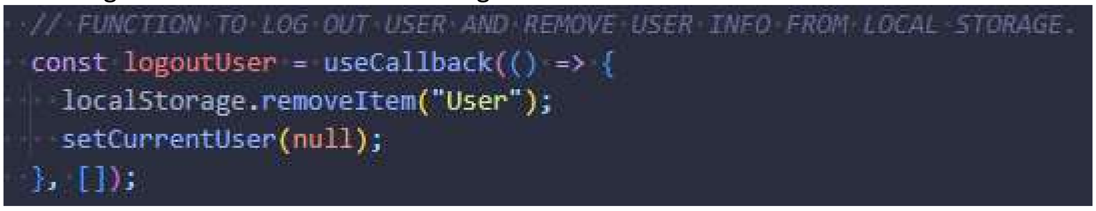
The last thing we have to do is pass the **"loginUserOnSubmit"** function to the **onSubmit** event of the form:



```
<form onSubmit={loginUserOnSubmit}>
```

3. LOGOUT

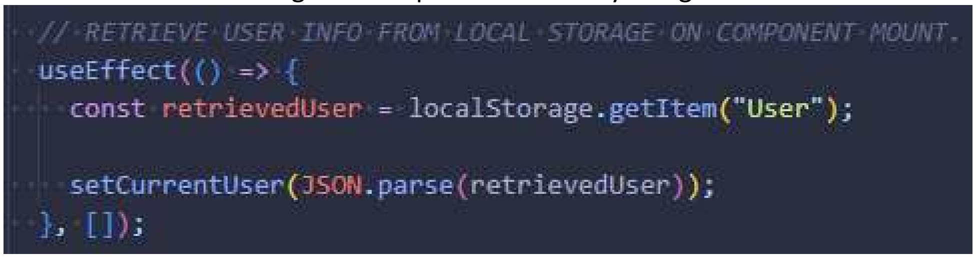
Once we trigger this function with **onClick** event, we'll be able to log out from the app by removing the user data from local storage.



```
// FUNCTION TO LOG OUT USER AND REMOVE USER INFO FROM LOCAL STORAGE.
const logoutUser = useCallback(() => {
  localStorage.removeItem("User");
  setCurrentUser(null);
}, []);
```

4. HOW TO KEEP THE USER LOGGED IN

Another method to keep the user logged in every time we open the app, is to retrieve the user info from the local storage on component mount by using useEffect.



```
// RETRIEVE USER INFO FROM LOCAL STORAGE ON COMPONENT MOUNT.
useEffect(() => {
  const retrievedUser = localStorage.getItem("User");

  setCurrentUser(JSON.parse(retrievedUser));
}, []);
```