Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)

ЛАБОРАТОРНАЯ РАБОТА 3

Управление ресурсами и использование хуков

По курсу

Разработка мобильных приложений

Выполнил **Килеев С.И.**

Студент Гр 211-321

Проверил ФИО

Захаров И.А,

Речинский В.А.

Натур В.В.

Москва, 2024

Цель работы: Научиться эффективно управлять ресурсами мобильного приложения и использовать хуки для управления состоянием и жизненным циклом компонентов.

Задачи:

1. Управление ресурсами приложения
   Реализовать управление ресурсами приложения, такими как данные, API запросы, локальное хранилище и другие ресурсы, используемые в приложении.
2. Использование хуков для управления состоянием
   Применить различные хуки (например, useState, useEffect) для управления состоянием компонентов, обработки событий и выполнения определенных действий в зависимости от жизненного цикла компонентов.

Отчет по выполнению:

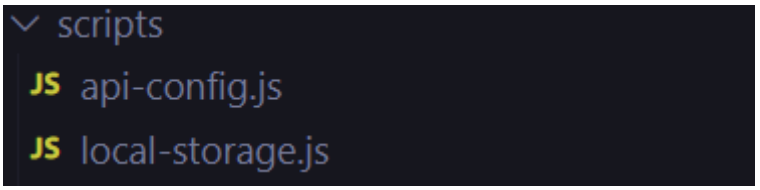

Рисунок 1 - Создадим 2 скрипта

```
export const createUser = async (email, password, username) => {
    try{
        let account = {
            username: username,
            email: email,
            password: password
        }

        fetch(signUpURL, {
            method: "post",
            headers: {
              'Accept': 'application/json',
              'Content-Type': 'application/json'
            },
            body: JSON.stringify(account)
          })
          .then((response) => response.json())
          .then((user) => {
            alert(user.id)
            signIn(username, password)
          })
    }
    catch(error){
        console.log(error)
        throw new Error(error)
    }
}
```

Рисунок 2 - В файле api-config создадим запрос на создание пользователя

```
export const signIn = async (username, password) => {
    let sign_in_account = {
        username: username,
        password: password
    }
    try{
      const response = await fetch(signInURL, {
          method: "post",
          headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
          },
          body: JSON.stringify(sign_in_account)
        })
      const json = await response.json()
      console.log(json)
      if(response.status == 200){
        await storeData(key_local_storage_token, json.token)
      }
      return [json.token, response.status]
    }
    catch(error){
      console.log(error)
      throw new Error(error)
    }
}
```

Рисунок 3 - В файле api-config создадим запрос на вход пользователя

```
export const getCurrentUser = async () => {
  const current_account_token = await getData(key_local_storage_token)

  if(current_account_token == null) throw Error;

  const response = await fetch(signInURL, {
    method: "post",
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json',
      'Authorization' : `Bearer ${current_account_token}`
    },
    body: ""
  })
  const json = await response.json()
  console.log(json)
  if(response.status != 200) throw Error;

  return json.id
```

Рисунок 4 - В файле api-config создадим запрос на получение ответа при отправке JWT-токена

```jsx
import { useContext, createContext, useEffect, useState } from "react";
import { getCurrentUser } from "../scripts/api-config";

const GlobalContext = createContext();
export const useGlobalContext = () => useContext(GlobalContext);

const GlobalProvider = ({ children }) => {
    const [isLoggedIn, setIsLoggedIn] = useState(false);
    const [user, setUser] = useState(null);
    const [isLoading, setIsLoading] = useState(true);

    useEffect(() => {
        getCurrentUser()
            .then((res) => {
                if(res) {
                    setIsLoggedIn(true)
                    setUser(res)
                }else{
                    setIsLoggedIn(false)
                    setUser(null)
                }
            })
            .catch((error) => {
                console.log(error)
            })
            .finally(() => {
                setIsLoading(false)
            })
    }, [])

    return(
        <GlobalContext.Provider
            value={{
                isLoggedIn,
                setIsLoggedIn,
                user,
                setUser,
                isLoading
            }}
        >
            {children}
        </GlobalContext.Provider>
    )
}

export default GlobalProvider;
```

Рисунок 5 - В файле GlobalProvider создадим контекст, который позволит проходить через авторизацию.

Рисунок 6 - В файле index.jsx добавим условие для переадресации пользователя

Ссылка на репозиторий: https://github.com/Ctfy21/labs-RMP