**Entregable Final Full Stack**

En este Proyecto adaptamos algunos componentes del Frontend que ya teníamos para que reciban información de nuestra API (Backend), además fueron agregados 4 componentes, son ellos ReposListBD, UsersListBD, RepositoryDetails y UserDetails. También agregamos 2 archivos .CSS (ambos responsive) para dejar los nuevos componentes “bonitos”, los nuevos archivos .CSS son ReposListBD (sirve para ReposListBD y UsersListBD) y RepoDetails (sirve para UserDetails y RepositoryDetails).

1. AuthContext.tsx: Agregamos al useReducer el token para usarlo en el sitio accediendo a las rutas protegidas y que el token exista mientras estamos logueados.
2. GitLogin.tsx: Adaptamos el código para que el usuario “MichiCat” pueda loguearse usando su nombre de usuario y password real, caso ponga mal alguno de los datos va aparecer el error “Credenciales Incorrectas”.
3. if (username && password) {
4. try {
5. const response = await axios.post('http://localhost:3000/user/login', {
6. username: username,
7. password: password
8. })
9. *// console.log('Login response:', response)*
10. if (response.status === 200) {
11. const token = response.data.token
12. login(username, token)
13. navigate(`/user/${username}`)
14. } else {
15. setError('Credenciales incorrectas')
16. console.error('Error de inicio de sesión')
17. }
18. } catch (error) {
19. setError('Credenciales incorrectas')
20. console.error('Error de red:', error)
21. }
22. }

3. User.tsx: Adaptamos la página del usuario logueado para que reciba sus datos desde nuestra API (los datos de MichiCat).

4. UsersList.tsx / ReposList.tsx: Adaptamos ambos componentes para que hagan la búsqueda de usuarios o repositorios accediendo a las rutas de nuestra API y no directamente de la API de GitHub.

const handleSearch = async () => {

    if (searchQuery.trim() !== '') {

      setIsLoading(true) *// Iniciar la carga*

      try {

        const response = await axios.post(

          'http://localhost:3000/searchuser',

          { searchTerm: searchQuery },

          {

            headers: {

              Authorization: `${state.token}`

            }

          }

        )

        const searchData = response.data || {}

        const usersData = searchData.usersList || []

        setUsers(usersData)

        setCurrentPage(1)

        setHasSearched(true)

      } catch (error) {

        console.error('Error fetching GitHub users:', error)

      } finally {

        setIsLoading(false) *// Detener la carga después de la búsqueda*

      }

    }

  }

5. ReposListBD.tsx / UsersListBD.tsx: Creamos estos dos componentes para el proyecto final, mostramos el listado de lo que tenemos guardado en nuestra Base de Datos, o sea, las búsquedas que hicimos, además podemos hacer click en el nombre de la búsqueda para que nos lleve a la página de “detalles” sea de usuarios o repositorios, también podemos hacer click en la papelera para eliminar cada búsqueda, si elimino una búsqueda me actualiza el listado para que no se vea la búsqueda que acabamos de eliminar. Abajo hay una muestra de como traemos de nuestra API el listado de lo que fue buscado “fetchSearches” y el llamado a la API para eliminar una búsqueda “handleDeleteSearch”.

useEffect(() => {

    const fetchSearches = async () => {

      try {

        const response = await axios.get<SearchUser[]>(

          'http://localhost:3000/searchuser',

          {

            headers: {

              Authorization: `${state.token}`

            }

          }

        )

        setSearches(response.data)

      } catch (error) {

        console.error('Error al obtener las búsquedas de usuarios', error)

      }

    }

    fetchSearches()

  }, [state.token])

  const handleDeleteSearch = async (*id*: string) => {

    try {

      await axios.delete(`http://localhost:3000/searchuser/${*id*}`, {

        headers: {

          Authorization: `${state.token}`

        }

      })

*// Actualiza la lista de búsquedas después de eliminar*

      const updatedSearches = searches.filter(*search* => *search*.\_id !== *id*)

      setSearches(updatedSearches)

    } catch (error) {

      console.error('Error al eliminar la búsqueda de usuarios', error)

    }

  }

6. UserDetails.tsx / RepositoryDetails.tsx: Creamos estos dos componentes para mostrar los detalles(cantidad de resultados encontrados, los detalles de cada resultado), en este componente también podemos eliminar la búsqueda, al hacerlo elimina la búsqueda y nos envía a la página UserListBD o ReposListBD. Además acá es donde hacemos el Update, agreamos, modificamos o quitamos un comentario, al hacerlo también modifica la fecha y hora de la parte “Última actualización”, al hacer click en mostrar detalles nos muestra todos los detalles de los usuarios o repositorios encontrados, ambos componentes tienen paginacíon mostrando de a 10 usuarios o repos por página. Abajo una muestra de los llamados para mostrar la búsqueda seleccionada, update y delete.

useEffect(() => {

    const fetchRepositoryData = async () => {

      try {

        const response = await axios.get(

          `http://localhost:3000/searchrepos/${id}`,

          {

            headers: {

              Authorization: `${state.token}`

            }

          }

        )

        setRepositoryData(response.data)

      } catch (error) {

        console.error('Error al obtener los detalles del repositorio', error)

      }

    }

    fetchRepositoryData()

  }, [id])

  const handleAddOrUpdateComment = async () => {

    try {

      const currentDate = new Date() *// Obtiene la fecha actual*

      await axios.patch(

        `http://localhost:3000/searchrepos/${id}`,

        { comment: newComment, updatedAt: currentDate.toISOString() },

        {

          headers: {

            Authorization: `${state.token}`

          }

        }

      )

*// Actualiza el comentario y la fecha en el estado*

      if (repositoryData) {

        const updatedRepositoryData = {

          ...repositoryData,

          comment: newComment,

          updatedAt: currentDate.toISOString()

        }

        setRepositoryData(updatedRepositoryData)

      }

      setNewComment('') *// Limpia el campo de entrada*

*//console.log('Comentario actualizado:', response.data)*

    } catch (error) {

      console.error('Error al agregar o actualizar el comentario', error)

    }

  }

  const handleDeleteRepository = async () => {

    try {

      await axios.delete(`http://localhost:3000/searchrepos/${id}`, {

        headers: {

          Authorization: `${state.token}`

        }

      })

      navigate(`/user/${state.username}/reposlistbd`)

    } catch (error) {

      console.error('Error al eliminar el repositorio', error)

    }

  }

7. App.tsx: Por último agregamos 4 rutas para nuestros 4 nuevos componentes en este archivo para acceder a ellas en el frontend.

<Route

*path*="/user/:username/reposlistbd"

*element*={state.isLoggedIn ? <ReposListBD /> : <Navigate *to*="/" />}

      />

      <Route

*path*="/user/:username/reposlistbd/:id"

*element*={state.isLoggedIn ? <SelectedRepoBD /> : <Navigate *to*="/" />}

      />

      <Route

*path*="/user/:username/userslistbd"

*element*={state.isLoggedIn ? <UsersListBD /> : <Navigate *to*="/" />}

      />

      <Route

*path*="/user/:username/userslistbd/:id"

*element*={state.isLoggedIn ? <SelectedUserBD /> : <Navigate *to*="/" />}

      />