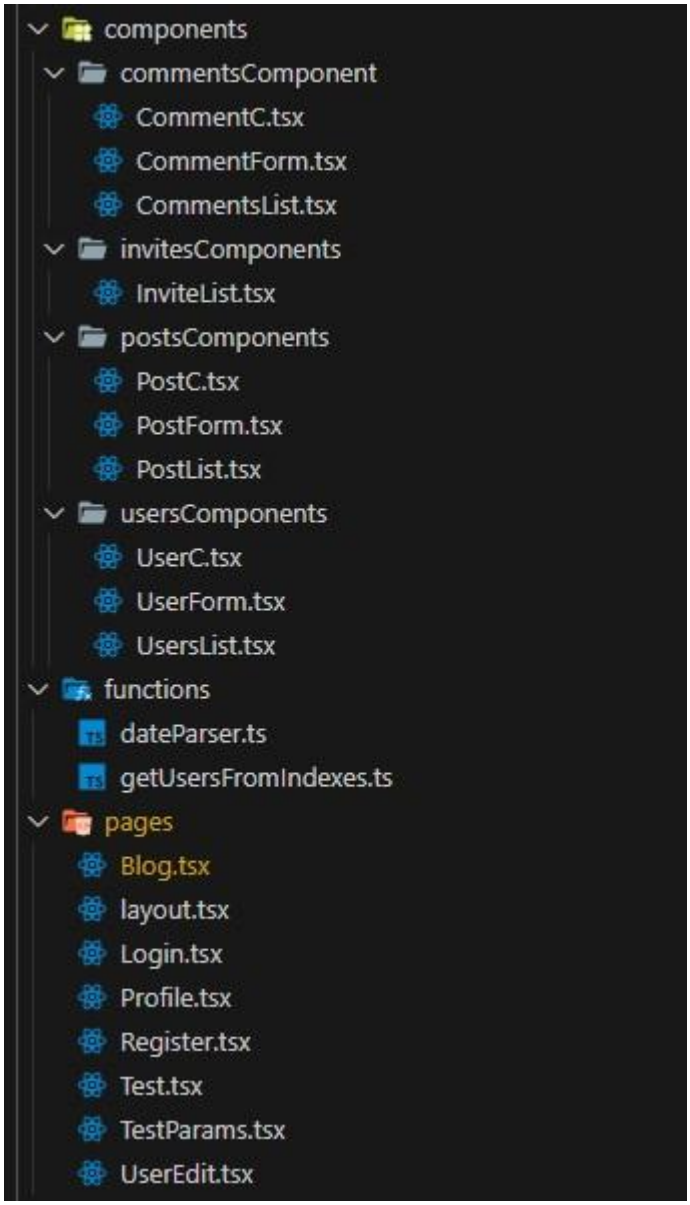
Wydział Informatyki Politechniki Białostockiej Aplikacje Internetowe oparte o komponenty	Data: 28.01.2024
Projekt 2: React Grupa: PS1, PS9 Piotr Muzyka 110882 Jakub Sak 111089 Łukasz Leończak 110841	Prowadzący: Dr. Inż. Urszula Kużelewska

1. Architektura Komponentów



Rys.1 – Struktura componentów

2. Routing aplikacji

Routing a aplikacji został zrealizowany przy pomocy bilblioteki „react-router-dom”. Przy jej pomocy element routingu został zrealizowany w pliku „index.tsx” w podstawowym folderze aplikacji poprzez stworzenie i zrenderowanie elementu html który służy za podstawę wszystkiego co wyświtlane jest w aplikacji. Poszczególne strony zostały zaimplementowane jako komponenty w folderze „Pages” w celu prostego rozróżnienia poszczególnym elementów

```
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <BrowserRouter>
    <Provider store={store}>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route path="Blog" element={<Blog />} />
          <Route path="Test" element={<Test />} />
          <Route path="Login" element={<Login />} />
          <Route path="Register" element={<Register />} />
          <Route path='UserEdit' element={<UserEdit/>}/>
          <Route path='Profile/:id' element={<Profile/>}></Route>

        </Route>
      </Routes>
    </Provider>
  </BrowserRouter>
);
```

Rys. 2 – routing

Wszelkie przekierowywanie użytkownika odbywają według podanych w wyżej podanym rysunku nazw. Do przekierowywanie użyytkownika zostały użyte takie funckje jak tag „<Link/>” oraz klucz „navigate” pochodzący z react hook’a „useNavigate()”

```jsx
return (
  <>
    <AppBar position="fixed" sx={{ bgcolor: "#7D84B2" }}>

      <Toolbar>
        <Link to="/blog">
          <Button color="inherit" variant="contained" sx={{ mr: 2 }}>
            Home
          </Button>
        </Link>       muzykos, w zeszłym tygodniu • Quick navbar :>
        {loggedUser.someoneIsLogged ?
          (
            <>
              <Link to={`/Profile/${loggedUser.user?.id}`}>
                <Button color="inherit" variant="contained" sx={{ mr: 2 }}>
                  Profile
                </Button>
              </Link>
              <Button onClick={() => dispatch(logout())} color="inherit" variant="contained" sx={{ mr: 2 }}>
                Logout
              </Button>
            </>
          ) : (
            <>
              <Link to="/login">
                <Button color="inherit" variant="contained" sx={{ mr: 2 }}>
                  Login
                </Button>
              </Link>
              <Link to="/register">
                <Button color="inherit" variant="contained">
                  Register
                </Button>
              </Link>
            </>
          )
        }


      </Toolbar>
    </AppBar>
    <Outlet />
  </>
```

Rys.3 – layout panelu nawigacji

```jsx
const navigate = useNavigate();
navigate(`/blog`)
```

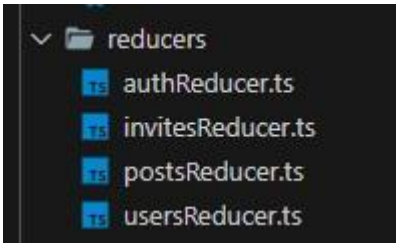
Rys.4 – sposób przekierowania uzytkownika poprzez „navigate”

3. Dane w aplikacji

Dane potrzebne dla aplikacji przy uruchomieniu aplikacji zostają zaimportowane (fetched) do kontekstu komponentu który uruchamia się podczas uruchomienia aplikacji i nastepnie przesłane do kontenera stanu „Redux”. Podstawowym elementem danej usługi jest zadeklarowany plik store.ts służacy jako podstawa działania kontenera W ten sposób dane są łatwo dostepne dla całej aplikacji a ich modyfikacja lub operacje na nich są przetwarzane przez klasy typu „reducer”

```ts
import { configureStore } from "@reduxjs/toolkit";
import usersReducer from "./reducers/usersReducer";
import postsReducer from "./reducers/postsReducer";
import invitesReducer from "./reducers/invitesReducer";
import authReducer from "./reducers/authReducer";

export const store = configureStore({
  reducer: {
    users: usersReducer,
    posts: postsReducer,
    invites: invitesReducer,
    auth: authReducer
  },
});

export type RootState = ReturnType<typeof store.getState>

export type AppDispatch = typeof store.dispatch
```

Rys.5 – konfiguracja REDUX store

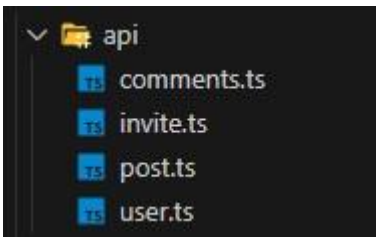


Rys.6 – lista REDUCER’ów

Rys.7 – Fragment kodu reducera dotyczącego postów

4. API

Jako api wybraliśmy JSON server który dzięki swojej prostocie pozwolił nam szybko i sprawnie stworzyć API do operacji typu CRUD dla poszczególnych typów danych. W folderze apliakcji znajduej się folder api który posiada funkcje z procesem FETCH dla każdego potrzebnego typu dancyh w aplikacji. Do pobierania danych z JSON serwera użyliśmy biblioteki „axios”



Rys.8 – zawartośc folderu API

```ts
import axios, { AxiosResponse } from 'axios';
import type {User} from "../types/user";


const URL = 'http://51.83.130.126:3000/users_react'

You, 4 dni temu | 1 author (You) | Codeium: Explain
interface Users{
    users:User[]
}


Codeium: Refactor | Explain | Generate JSDoc | X
export const getUsers = async() => {
    try{
        const response: AxiosResponse<Users> = await axios.get<Users>(`${URL}`)
        return response.data
    }catch(error){
        return []
    }
}


Codeium: Refactor | Explain | Generate JSDoc | X
export const postUser = async(user:User) => {
    await axios.post(`${URL}`, {
        id: user.id,
        name: user.name,
        surname: user.surname,
        email: user.email,
        username: user.username,
        password: user.password,
        friends: []
    }).then(response => {
        console.log("post user successfully")
        console.log(response.data)
    }).catch(error => {
        console.log(error)
        return
    })
}
```

Rys.9 – Fragment pliku api/post.ts

5. Fragmenty kodu

```
function RemovePost(id: string){
    dispatch(removePost(id))
    deletePost(id)
}
Codeium: Refactor | Explain | Generate JSDoc | X
function EditPost(){
    setEditPost(!editPost)
}
Codeium: Refactor | Explain | Generate JSDoc | X
const likeClicked = () => {
    if(!loggedUser.someoneIsLogged) return
    const id = loggedUser.user?.id || 'defaultId'

    let likes:number = post.likes
    let likesId:string[] = post.userIdLikes
    // if user likes this post
    if(post.userIdLikes.includes(id)){
        dispatch(removeLike({postId: post.id, userId: id}))
        likes = likes - 1
        likesId = likesId.filter(userId => userId !== id)
    }
    // if he do not like this post and do not dislikes
    else if(!post.userIdDislikes.includes(id)){
        dispatch(addLike({postId: post.id, userId: id}))
        likes = likes + 1
        likesId = [...likesId.map(likeId => likeId), id]
    }
    if(likes === post.likes){
        const editedPost:Post = {
            id: post.id,
            title: post.title,
            body: post.body,
            authorId: post.authorId,
            authorUsername: post.authorUsername,
            likes: likes,
            userIdLikes: likesId,
            dislikes: post.dislikes,
            userIdDislikes: post.userIdDislikes,
            createdAt: post.createdAt,
            updatedAt: post.updatedAt
        }
        putPost(editedPost)
    }

}
```

Rys.10 – Fragment kodu dotyczący obsługi zdarzeń posta, na obrazie fidoczne są funkcjeobsługujące usuwanie Posta, wodiku edycji oraz obsługę klknięcia przycisku like przez uzytkownika

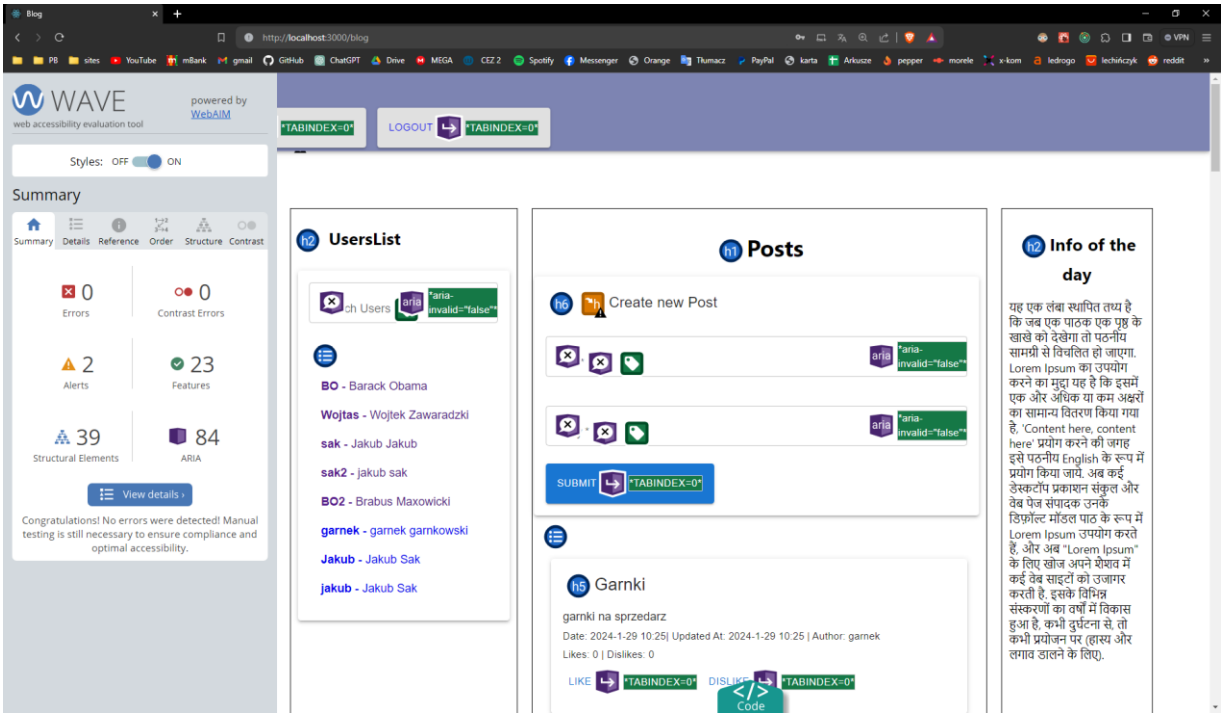
```
return (
    <Grid container spacing={2}>
      <Grid item xs={12}>
        {editPost ? (
          <Paper elevation={3} style={{ padding: '1rem' }}>
            <Typography variant="h6" gutterBottom>Edit Post</Typography>
            <form onSubmit={handleSubmit}>
              <TextField name="title" label="Title" fullWidth margin="normal"
              value={formData.title} onChange={handleChange} />
              <TextField name="body" label="Body" fullWidth multiline rows={4}
              required margin="normal" value={formData.body} onChange={handleChange} />
              <Button type="submit" variant="contained" color="primary">Submit</Button>
              <Button onClick={EditPost}>Cancel</Button>
            </form>
          </Paper>
        ) : (
          <Paper elevation={3} style={{ padding: '1rem' }}>
            <Typography variant="h5" gutterBottom>{post.title}</Typography>
            <Typography variant="body1" gutterBottom>{post.body}</Typography>
            {!editPost && loggedUser.user?.id === post.authorId && (
              <div>
                <IconButton onClick={EditPost}><EditIcon /></IconButton>
                <IconButton onClick={() => RemovePost(post.id)}><DeleteIcon /></IconButton>
              </div>
            )}
            <Typography variant="body2" gutterBottom>Date: {post.createdAt}
            | Updated At: {post.updatedAt} | Author: {post.authorUsername}</Typography>
            <Typography variant="body2">Likes: {post.likes} | Dislikes: {post.dislikes}</Typography>
            <Button onClick={likeClicked}>Like</Button>
            <Button onClick={dislikeClicked}>Dislike</Button>
          </Paper>
        )}
      </Grid>
      <Grid item xs={12}>
        <CommentsList comments={props.comments} post={post}/>
      </Grid>


    </Grid>

  )
```

Rys.11 – Widok komponentu postC który zwraca wartość html do wyświetlenia na stronie

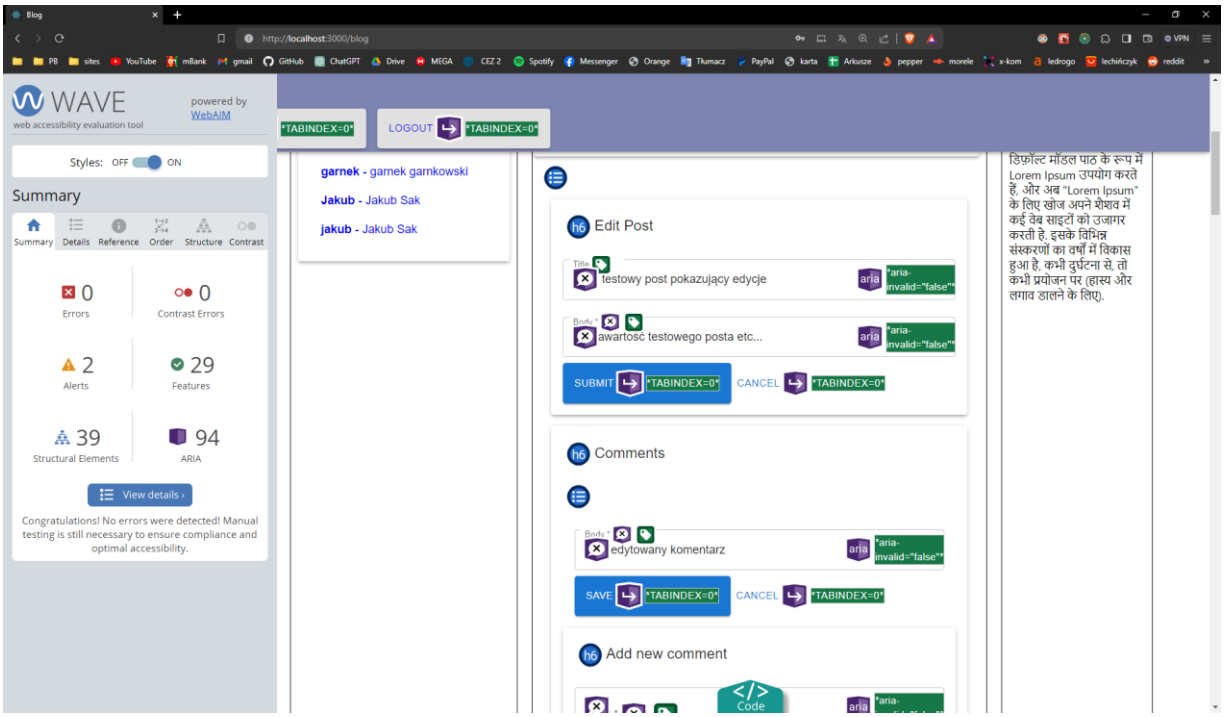
6. Zrealizowane elementy

- Mini Blog ○ Przeglądanie postów ○ Możliwość dodania nowego posta ○ Możliwosc komentowania postów
  - ○ Możliwosc dodawania do znajomych innych użytkowników
- Konta ○ Rejestracja uzytkownika ○ Logowanie uzytkownika ○ Widok Profilu uzytkownika
  - Widok postów użytkownika
  - Widok znajomych użytkownika

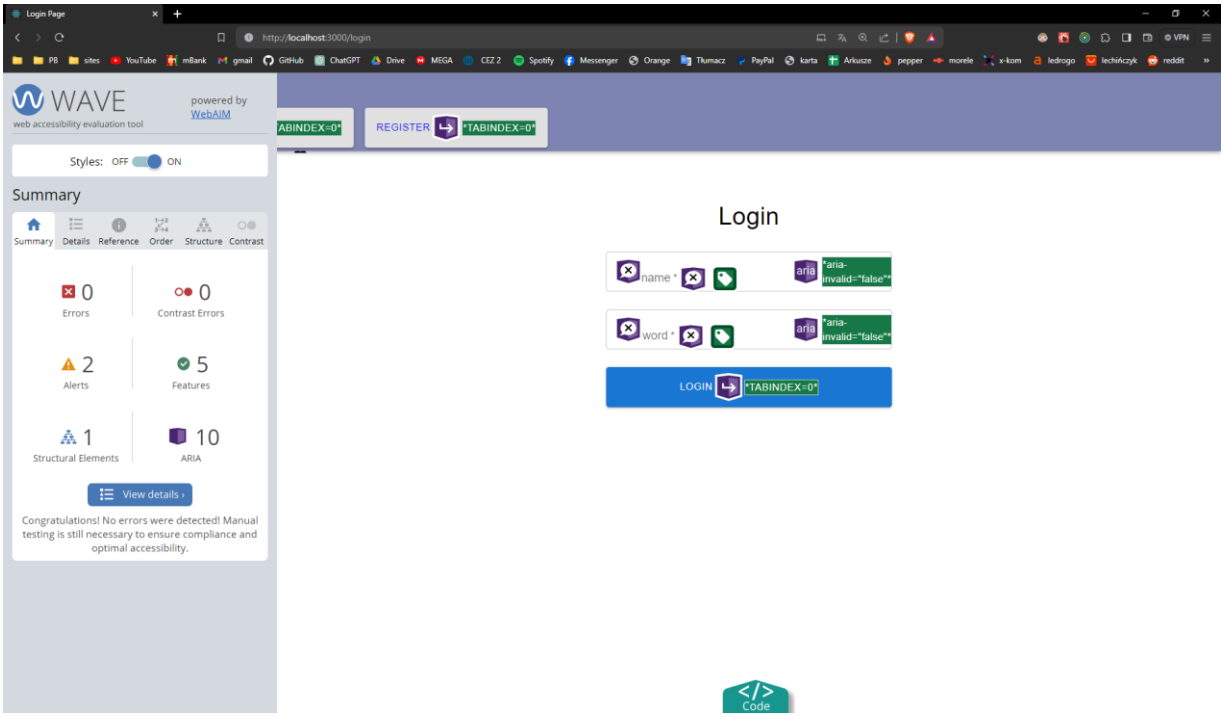
7. WAVE plugin



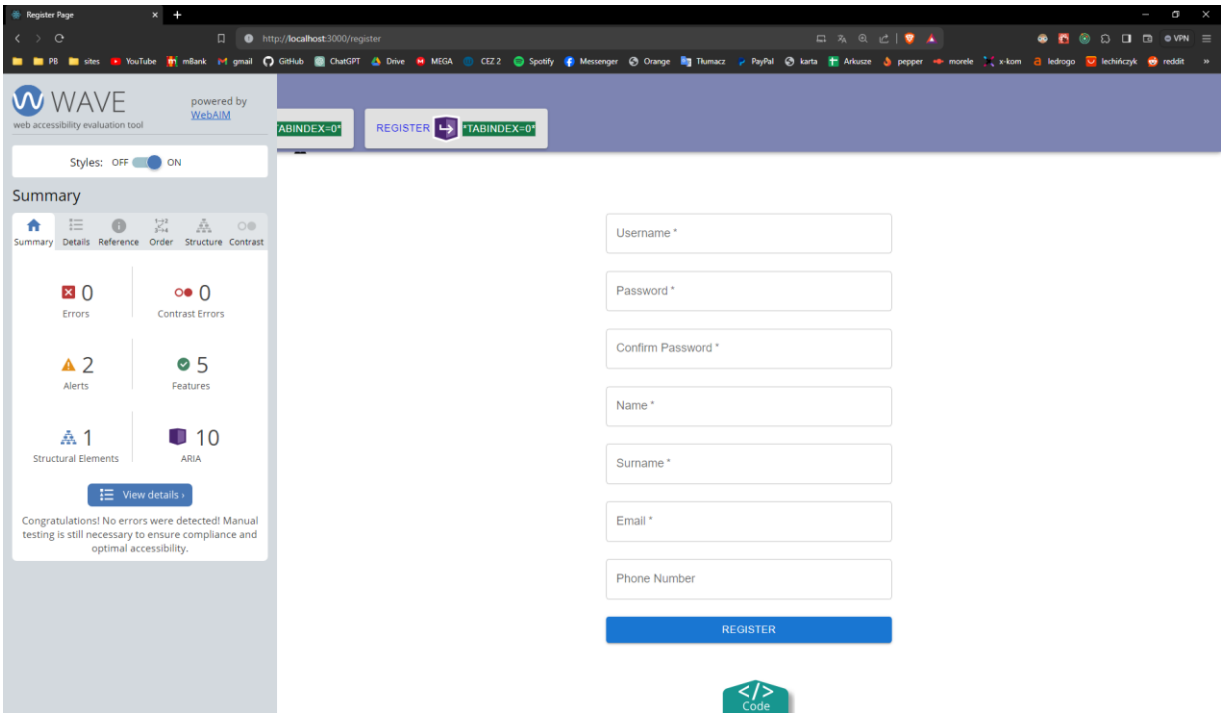
Rys.12 – Widok wtyczki wave na ekranie głównego widoku po zalogowaniu(widoczne są wtedy forularze do dodawania postów i komentarzy)



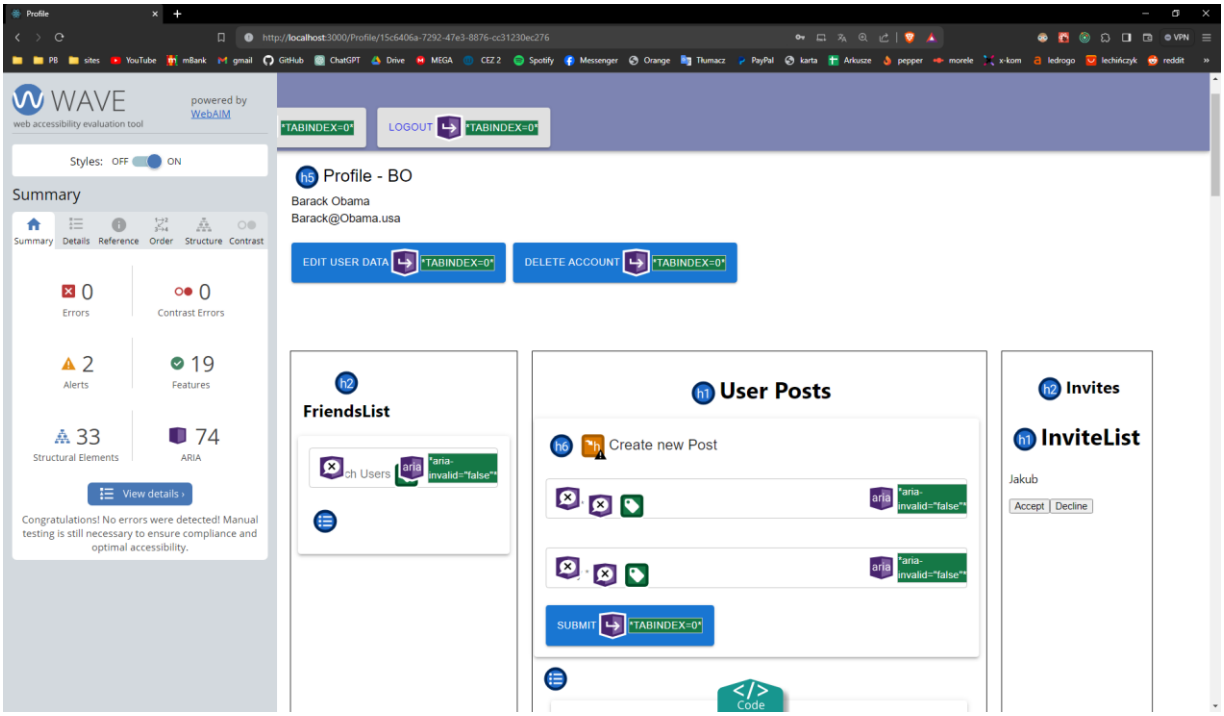
Rys.13 – Widok wtyczki wave na ekranie głównego widoku po zalogowaniu i wybraniu posta do edycji oraz komentarza do edycji(widoczne są wtedy forularze do eycji postów i komentarzy)



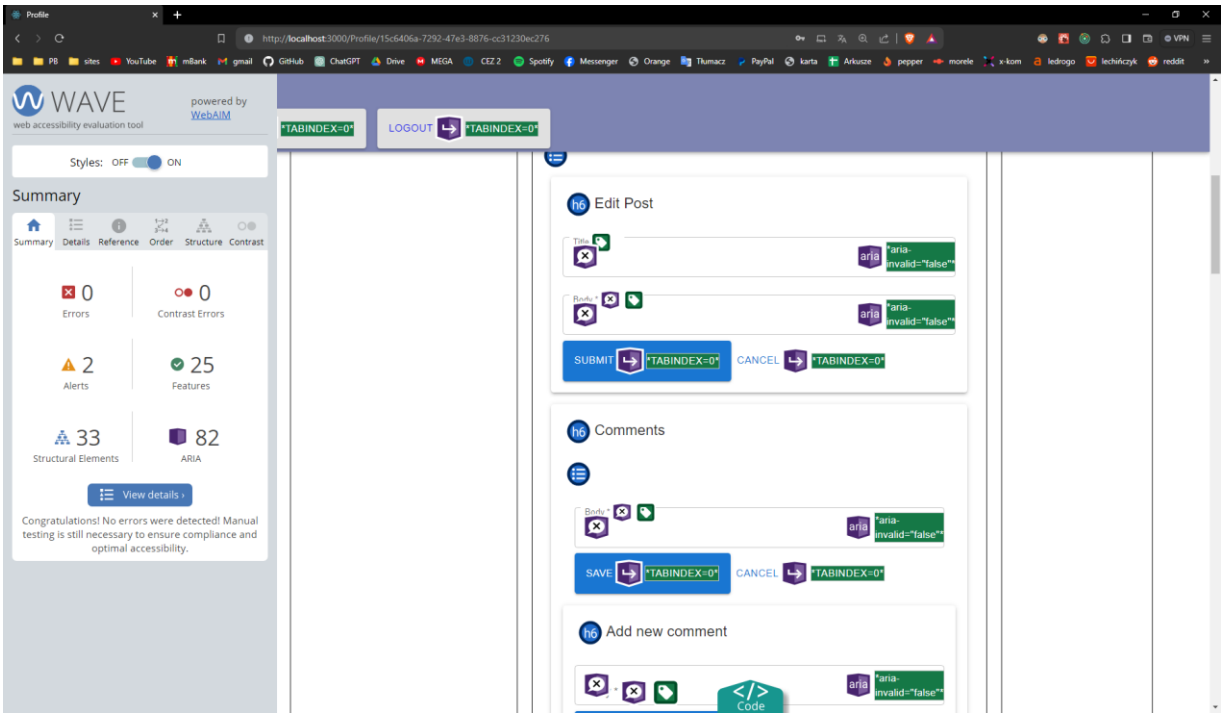
Rys.14 – Widok wtyczki WAVE w wiodku logowania



Rys.15 – Widok wtyczki WAVE w wiodku rejestracji



Rys.16 – Widok wtyczki WAVE w wiodku profilu użytkownika



Rys.17 – Widok wtyczki WAVE w wiodku profilu użytkownika z otwartymi formularzami edycji postów i komentarzy

8. Biliografia

- https://www.npmjs.com/package/axios
- https://www.npmjs.com/package/@reduxjs/toolkit
- https://www.npmjs.com/package/react-redux
- https://www.npmjs.com/package/@mui/material
- https://www.npmjs.com/package/@emotion/react
- https://www.npmjs.com/package/@emotion/styled
- https://www.npmjs.com/package/react-router-dom
- https://www.npmjs.com/package/uuid

9. Podział Prac

Sak Jakub – budowa projetku, api, JSON server, Autoryzacja uzytkownika logowanie, rejestracja, komponenty dotyczące użytkownika, implemetacja REDUX store, stylowanie css przy pomocu Material-UI

Piotr Muzyka – deklracje typów, komponenty Postów/Komentarzy, routing, KomponenrtyStrony przydzielone do routingu