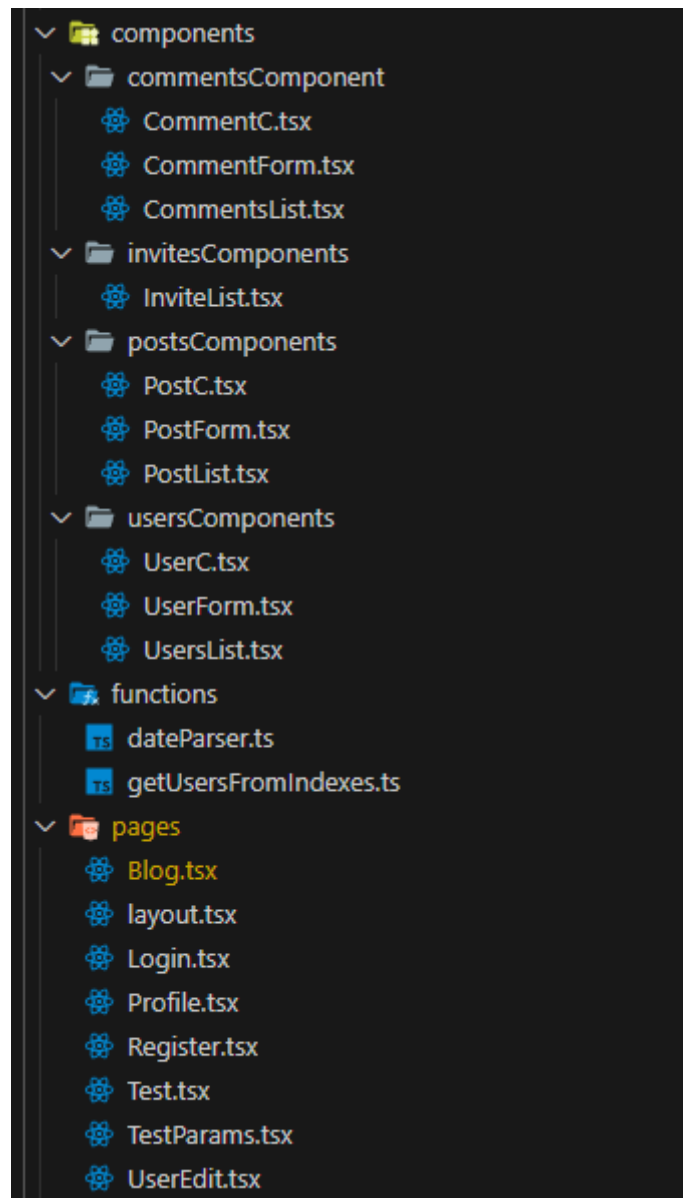


Wydział Informatyki Politechniki Białostockiej Aplikacje Internetowe oparte o komponenty	Data: 28.01.2024
Projekt 2: React Grupa: PS1, PS9 Piotr Muzyka 110882 Jakub Sak 111089 Łukasz Leończak 110841	Prowadzący: Dr. Inż. Urszula Kuźelewska

1. Architektura Komponentów



Rys.1 – Struktura componentów

2. Routing aplikacji

Routing aplikacji został zrealizowany przy pomocy biblioteki „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 wyświetlane jest w aplikacji. Poszczególne strony zostały zaimplementowane jako komponenty w folderze „Pages” w celu prostego rozróżnienia poszczególnych 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>  
      </Routes>  
    </Provider>  
  </BrowserRouter>  
)
```

Rys. 2 – routing

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

```

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

```

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

```

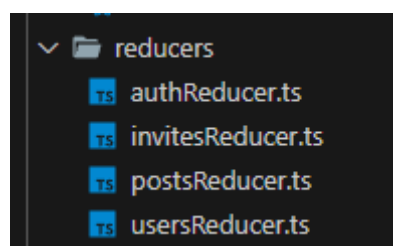
Rys.4 – sposób przekierowania użytkownika poprzez „navigate”

3. Dane w aplikacji

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

```
1 import { configureStore } from "@reduxjs/toolkit";
2 import usersReducer from "../reducers/usersReducer";
3 import postsReducer from "../reducers/postsReducer";
4 import invitesReducer from "../reducers/invitesReducer";
5 import authReducer from "../reducers/authReducer";
6
7 export const store = configureStore({
8   reducer: {
9     users: usersReducer,
10    posts: postsReducer,
11    invites: invitesReducer,
12    auth: authReducer
13  },
14 });
15
16 export type RootState = ReturnType<typeof store.getState>
17
18 export type AppDispatch = typeof store.dispatch
```

Rys.5 – konfiguracja REDUX store



Rys.6 – lista REDUCER'ów

```

... | Codeium: Explain
interface PostsState {
  posts: Post[]
  comments: Comment[][]
  init_status: true|false,
}

const initialState: PostsState = {
  init_status: true,
  posts: [],
  comments: []
};

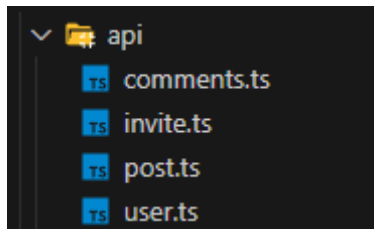
export const postSlice = createSlice({
  name: 'posts',
  initialState,
  reducers: [
    Codeium: Refactor | Explain | Generate JSDoc | X
    setPosts(state, action) {
      if(state.init_status) {
        console.log("set posts")
        console.log(action.payload)
        console.log(action.payload.comments)
        state.posts = action.payload.posts
        state.comments = action.payload.comments
        state.init_status = false
      }
    },
    Codeium: Refactor | Explain | Generate JSDoc | X
    addPost(state, action) {
      console.log("adding new post to redux")
      console.log(action.payload)
      if(state.posts === undefined) {
        state.posts = [action.payload]
      }else{
        state.posts.push(action.payload)
      }
    },
  ],

```

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 aplikacji znajduje się folder api który posiada funkcje z procesem FETCH dla każdego potrzebnego typu danych w aplikacji. Do pobierania danych z JSON serwera użyliśmy biblioteki „axios”



Rys.8 – zawartość folderu API

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

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

interface Users{
  users:User[]
}

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

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.someoneIsLoggedIn) 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 widoczne są funkcje obsługujące usuwanie Posta, edycję oraz obsługę kliknięcia przycisku like przez użytkownika

```
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
 - Możliwość komentowania postów
 - Możliwość dodawania do znajomych innych użytkowników
- Konta
 - Rejestracja użytkownika
 - Logowanie użytkownika
 - Widok Profilu użytkownika
 - Widok postów użytkownika
 - Widok znajomych użytkownika

7. Bibliografia

- <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>

8. Podział Prac

Sak Jakub – budowa projektu, api, JSON server, Autoryzacja użytkownika logowanie, rejestracja, komponenty dotyczące użytkownika, implementacja REDUX store, stylowanie css przy pomocy Material-UI

Piotr Muzyka – deklaracje typów, komponenty Postów/Komentarzy, routing, Komponenty-Strony przydzielone do routing

Łukasz Leończak – prace nad stylowaniem CSS, Komponenty dotyczące zaproszeń do znajomych