POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki
Filia w Gdańsku

Langmesser Adam
Nr albumu s27119
Nazwa specjalizacji: Aplikacje Internetowe
Redosz Mateusz
Nr albumu s27094
Nazwa specjalizacji: Aplikacje Internetowe
Oziemczuk Stanisław
Nr albumu s26982
Nazwa specjalizacji: Aplikacje Internetowe
Badek Kacper
Nr albumu s29168
Nazwa specjalizacji: Aplikacje Internetowe

Aplikacja webowa: spoty-na-drony.pl

Rodzaj pracy
inżynierska
Imię i nazwisko promotora
mgr Adam Urbanowicz

Gdańsk, miesiąc, 2100 obrony

Streszczenie: Celem niniejszej pracy było stworzenie w pełni funkcjonalnej i działającej aplikacji internetowej pozwalającej na szybkie wyszukiwanie spotów w okolicy oraz dzielenie się zdjęciami, filmami oraz doświdczeniem z innymi użytkownikami. W ramach pracy stworzono system składający się z trzech komponentów: Frontendu, Backendu oraz bazy-danych. Aplikacja internetowa została wykonana przy pomocy Frameworka React w językach Javascript oraz Typescript, do styli został użyty Tailwind. Serwis backendowy został stworzony w języku Java oraz biblioteki Spring Boot. Baza danych to PostgreSql.
Komunikacja między komponentami odbywała się zgodnie ze standardem REST. Projekt został zrealizowany w podejściu ewolucyjno-przyrostowym z elementami Kanban.
Słowa kluczowe: — brak —

POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Karta projektu

Temat projektu: Aplikacja webowa: spoty-na-drony.pl **Temat projektu po angielsku:** Web application: spoty-na-drony.pl	**Akronim:** Merkury **Data ustalenia tematu** 2023-10-10
Promotor: mgr Adam Urbanowicz	**Konsultanci:** 1. — brak —
Cele projektu: Stworzenie w pełni funkcjonalnej aplikacji internetowej do rozwijania hobby(latania dronem).	
Rezultaty projektu: Aplikacja Internetowa, Dokumentacja Interaktywna mapa z wyświetlanymi spotami oraz pogodą. Zaawansowana wyszukiwarka spotów. Forum do dzielenia się informacjami na temat dronów. Chat jednoosobowy oraz grupowy. Konto użytkownika z możliwością zapisania ulubionych spotów.	
Miary sukcesu: Gotowa do wdrożenia aplikacja. Realizacja w terminie zgodnym z wymaganiami.	
Ograniczenia: Budżetowe: brak środków na wdrożenie. Zawodowe: brak doświadczenia. Czasowe: trzy semestry (09.2024 - 02.2026). Ludzkie: czteroosobowy zespół.	

Wykonawcy	**Numer albumu**	**Specjalizacja**	**Tryb studiów**
Langmesser Adam	s27119	Aplikacje Internetowe	Stacjonarny
Redosz Mateusz	s27094	Aplikacje Internetowe	Stacjonarny
Oziemczuk Stanisław	s26982	Aplikacje Internetowe	Stacjonarny
Badek Kacper	s29168	Aplikacje Internetowe	Stacjonarny

Data ukończenia projektu:	**Recenzent:**
1 listopada 2025	dr Elżbieta Puźniakowska-Gałuch

Spis treści

Rozdział 1

Wstęp

1.1 O projekcie

1.2 Cel i zakres prac

1.3 Geneza pomysłu

Słownik pojęć i skrótów

Backend Część aplikacji odpowiedzialna za logikę biznesową, przetwarzanie danych i komunikację z bazą danych. Działa po stronie serwera i obsługuje żądania wysyłane przez frontend.. 2, 30

Bibliote Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera.. 23

CI/CD Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania.. 29, 32

Design Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI).. 29

Folder by type Sposób organizowania struktury katalogów w projekcie, w którym pliki są grupowane według rodzaju (typu) zasobu, a nie według funkcjonalności. Na przykład wszystkie komponenty trafiają do jednego folderu, wszystkie style do innego itd.. 18

Framework Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu.. 2

Frontend Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript.. 2, 30

JWT Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników.. 29

Protected route Trasa w aplikacji, do której dostęp jest ograniczony, zwykle tylko dla zalogowanych użytkowników lub użytkowników z odpowiednimi uprawnieniami. Jeżeli użytkownik nie spełnia warunków, jest przekierowywany.. 22

Review kodu Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu.. 29

Sidebar Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji.. 29–31

Stan Aktualny zestaw danych przechowywanych przez aplikację lub komponent, na podstawie którego renderowany jest interfejs użytkownika. Stan może być lokalny (utrzymywany w pojedynczym komponencie) albo globalny (wspólny dla wielu komponentów).. 23

UI Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. 23

Rozdział 2

Opis problemu

2.1 Rich picture

2.2 Udziałowcy

2.3 Istniejące rozwiązania

2.4 Wizja rozwiązania

2.5 Aspekty społeczne i biznesowe

2.5.1 Aspekty społeczne

2.5.2 Aspekty biznesowe

Rozdział 3

Planowanie

3.1 Metodologia pracy

3.2 Harmonogram projektu

3.3 Technologie i narzędzia

3.3.1 Technologie

3.3.2 Narzędzia

3.4 Zasoby i ograniczenia

3.4.1 Zasoby

3.4.2 Ograniczenia

3.5 Analiza ryzyka

Rozdział 4

Analiza wymagań

4.1 Przypadki użycia

4.1.1 Aktorzy

4.1.2 Diagram przypadków użycia

4.1.3 Scenariusz przypadków użycia

4.2 Wymagania ogólne i dziedzinowe

4.3 Wymagania funkcjonalne

4.3.1 Funkcjonalności dla mapy

4.3.2 Funkcjonalności dla chatu

4.3.3 Funkcjonalności dla forum

4.3.4 Funkcjonalności dla konta użytkownika

4.3.5 Funkcjonalności dla logowania i rejestracji

4.3.6 Funkcjonalności dla wyszukiwarki spotów

4.3.7 Funkcjonalności dla motywu

4.4 Wymagania pozafunkcjonalne

4.5 Wymagania interfejs z otoczeniem

4.6 Wymagania na środowisko docelowe

Rozdział 5

Projekt

5.1 Wzorce projektowe

5.2 Architektura systemu

5.2.1 Diagram architektury

5.2.2 Komponenty systemu

5.3 Projekt bazy danych

5.3.1 Model danych

5.3.2 Diagram ERD

5.4 Architektura interfejsu użytkownika

5.4.1 Projekt strony głównej

5.4.2 Projekt panelu logowania

5.4.3 Projekt mapy

5.4.4 Projekt chatu

5.4.5 Projekt forum

5.4.6 Projekt konta użytkownika

Rozdział 6

Przebieg realizacji projektu

6.1 Sprint 1

6.2 Sprint 2

Rozdział 7

Realizacja Projektu

7.1 Implementacja backendu

7.1.1 Struktura projektu

7.1.2 Endpointy systemu

GET `/user-dashboard/profile`

Opis: Zwraca profil aktualnie zalogowanego użytkownika.

Metoda: `GET /user-dashboard/profile`

Zwraca (200 OK): `application/json` — obiekt `UserProfileDto`.

Błąd (404 Not Found): `text/plain` —
komunikat z wyjątku `UserNotFoundByUsernameException`.

Przykładowa odpowiedź (200 OK):

```
{
  "username": "john_doe",
  "profilePhoto": "https://cdn.example.com/profiles/john_doe.
     jpg",
  "followersCount": 125,
  "followedCount": 87,
  "friendsCount": 32,
  "photosCount": 58,
  "mostPopularPhotos": [
    {
      "src": "https://cdn.example.com/photos/123.jpg",
      "heartsCount": 240,
```

```
      "viewsCount": 3400,
      "title": "Sunset at the beach",
      "id": 123
    }
  ]
}
```

Example response (404 Not Found):

Panel użytkownika

- GET /user-dashboard/profile
- GET /public/user-dashboard/profile/"targetUsername"
- PATCH /user-dashboard/profile
- GET /user-dashboard/friends
- GET /public/user-dashboard/friends/"targetUsername"
- PATCH /user-dashboard/friends
- PATCH /user-dashboard/friends/change-status
- GET /user-dashboard/followers
- GET /public/user-dashboard/followers/"targetUsername"
- GET /user-dashboard/followed
- GET /public/user-dashboard/followed/"targetUsername"
- GET /user-dashboard/friends/find
- GET /user-dashboard/friends/invites
- PATCH /user-dashboard/followed
- GET /user-dashboard/favorite-spots
- PATCH /user-dashboard/favorite-spots

- GET /user-dashboard/photos
- GET /user-dashboard/comments
- PATCH /user-dashboard/settings
- GET /user-dashboard/settings
- GET /user-dashboard/movies
- GET /user-dashboard/photos/"targetUsername"
- GET /user-dashboard/add-spot
- POST /user-dashboard/add-spot
- GET /user-dashboard/add-spot/coordinates

Strona główna

- GET /public/spot/most-popular
- GET /public/spot/search/home-page
- GET /public/spot/search/home-page/locations
- GET /public/spot/search/home-page/advance

Konto użytkownika

- POST /public/account/register
- POST /public/account/login
- GET /account/login-success
- POST /public/account/forgot-password
- POST /public/account/set-new-password
- GET /account/check

7.1.3 Integracja z bazą danych

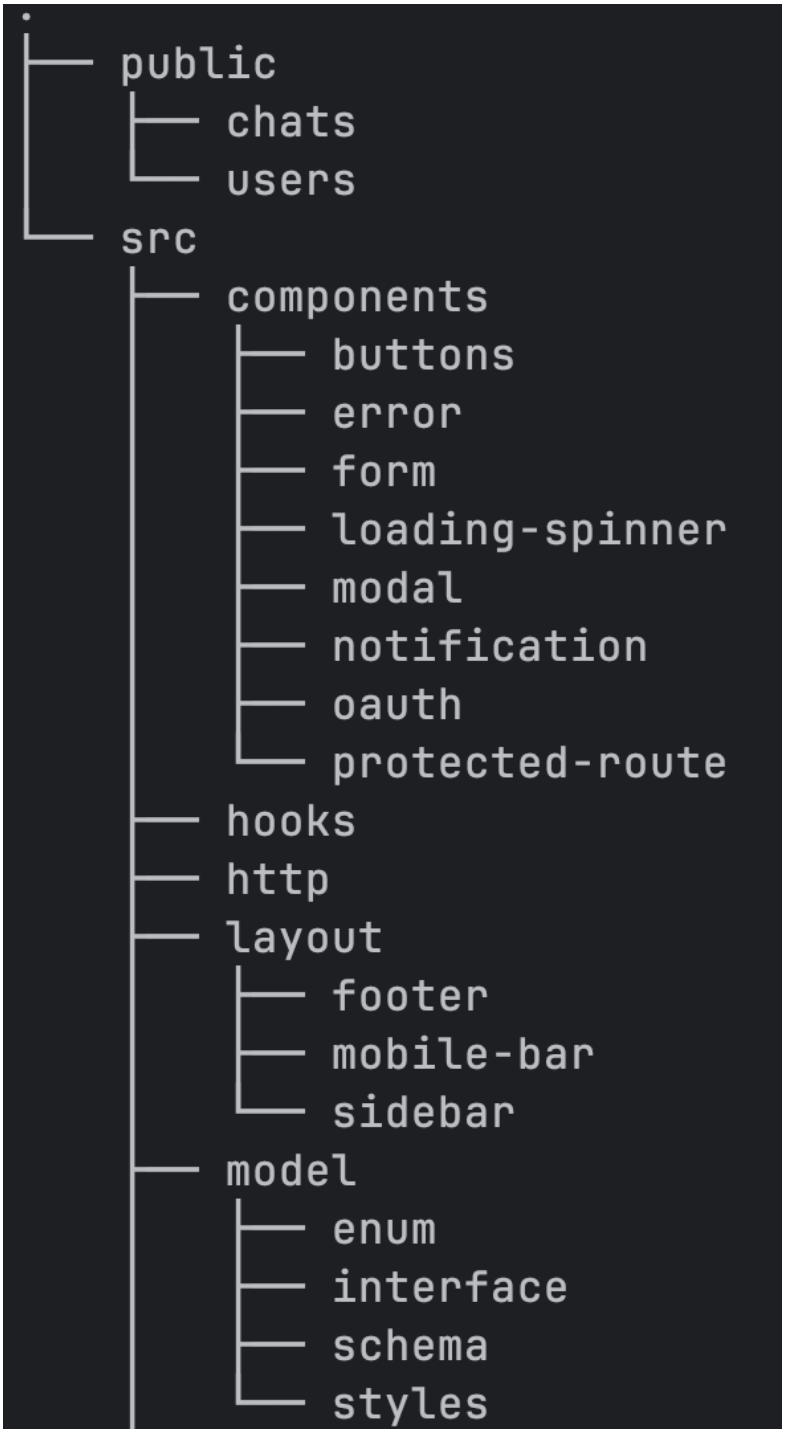
7.1.4 Obsługa uwierzytelnienia

7.1.5 Konteneryzacja

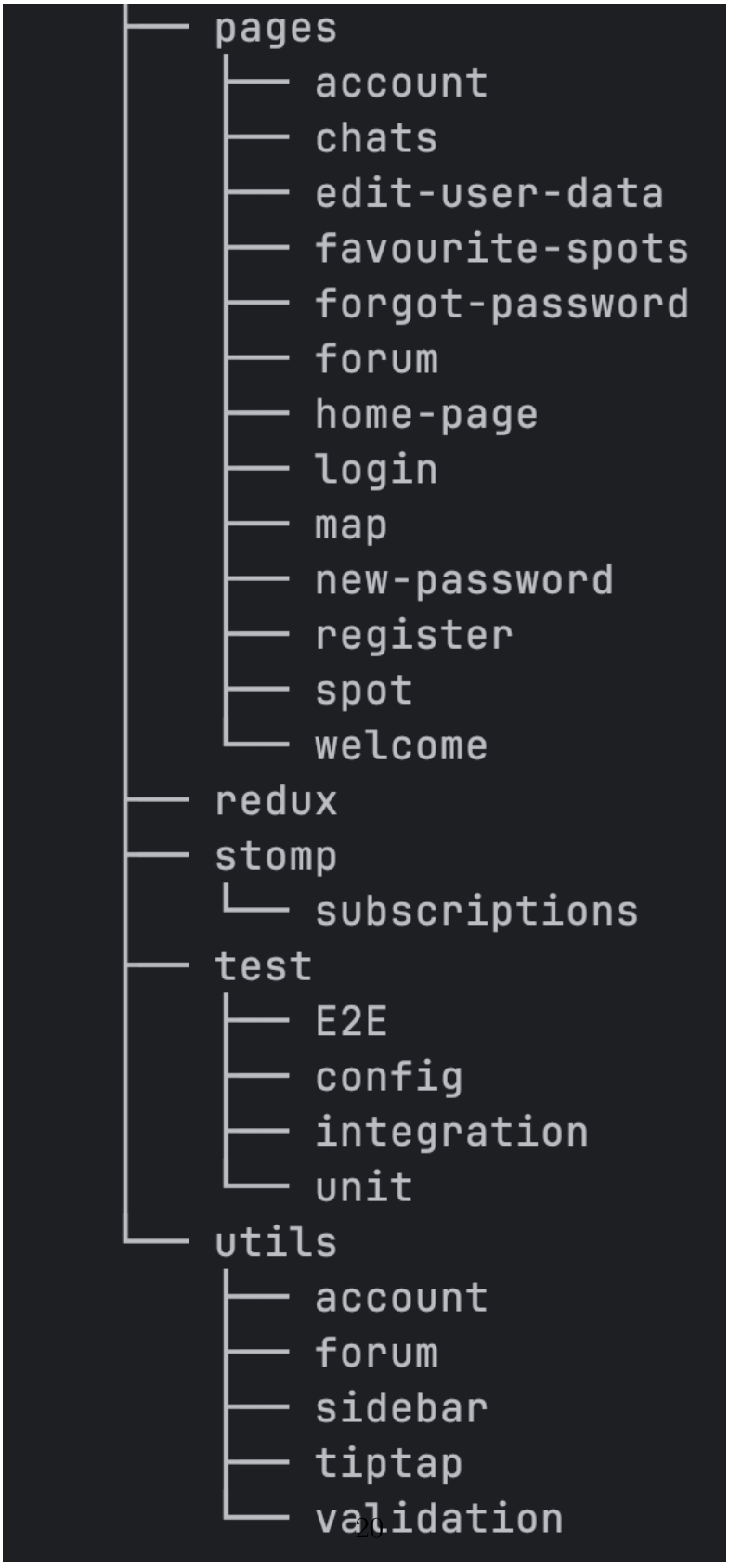
7.2 Implementacja frontendu

7.2.1 Struktura aplikacji

Architektura aplikacji frontendowej została zaprojektowana w strukturze Folder by type, która polega na podziale kodu według typu zasobu (komponenty, strony, modele itd.). Każdy plik znajduje się w katalogu odpowiadającym jego przeznaczeniu, co widać na rysunkach 7.1 oraz 7.2.



Rysunek 7.1: Struktura katalogów (1)



Rysunek 7.2: Struktura katalogów (2)

Głównym elementem aplikacji jest mechanizm routingu oparty na bibliotece React Router. Definiuje on ścieżki do poszczególnych funkcjonalności aplikacji. Dzięki temu możliwa jest płynna nawigacja między różnymi widokami bez konieczności przeładowywania strony.

```
const router : Router = createBrowserRouter([
    {
        path: "/",
        element: <Layout />,
        errorElement: <Error error={undefined} />,
        children: [
            {
                index: true,
                element: <HomePage />,
            },
            {
                path: "advanced",
                element: <AdvanceHomePage />,
            },
            {
                path: "account",
                children: [ 11 elements… ],
            },
            {
                path: "register",
                element: <Register />,
            },
            {
                path: "login",
                element: <Login />,
            },
            {
                path: "forgot-password",
                element: <ForgotPassword />,
            },
            {
```

Rysunek 7.3: Implementacja routera (1)

Rysunek 7.4: Implementacja routera (2)

W projekcie zastosowano również wzorzec Protected route, który służy do zabezpieczania wybranych tras przed dostępem użytkowników niezalogowanych. W pliku `router.tsx`, znajdującym się w głównym katalogu projektu, w konfiguracji przekazywanej do funkcji `createBrowserRouter` (rysunki 7.3 oraz 7.4), wybrane

ścieżki zostały opakowane w komponent `ProtectedRoute`. Komponent ten pełni rolę bramki (rysunek 7.5).

Przykładem takiej chronionej ścieżki jest trasa `/chat`, prowadząca do modułu czatu dostępnego wyłącznie dla zalogowanych użytkowników. Jeśli niezalogowany użytkownik spróbuje uzyskać dostęp do tej ścieżki, zostanie automatycznie przekierowany na stronę główną.

```
export default function ProtectedRoute({ children }) {
    const isLogged = useSelector((state) => state.account.isLogged);

    return isLogged ? children : <Navigate to={"/"} />;
}
```

Rysunek 7.5: Implementacja komponentu bramki (`ProtectedRoute`)

7.2.2 Zarządzanie stanem i przepływ danych

W projekcie postawiliśmy na zrównoważone podejście do zarządzania Stanem. Korzystamy zarówno z lokalnego Stanu komponentów (za pomocą hooka `useState`), jak i ze Stanu globalnego, utrzymywanego przez Bibliotekę React Redux. Globalny Stan został wprowadzony po to, aby możliwie najbardziej ograniczyć przekazywanie propsów w głąb drzewa komponentów oraz uniknąć niepotrzebnych ponownych renderów.

Do przechowywania Stanu lokalnego, ograniczonego tylko do danego komponentu (lub jego najbliższych elementów podrzędnych), wykorzystujemy hooki `useState` oraz `useEffect`. Natomiast w przypadku bardziej złożonej logiki lub potrzeby ponownego wykorzystania kodu powstały hooki niestandardowe, takie jak `useScreenSize`, `useDarkMode` czy `useClickOutside`. Dzięki temu większość logiki biznesowej została wydzielona z warstwy UI, co poprawia czytelność i ułatwia utrzymanie kodu.

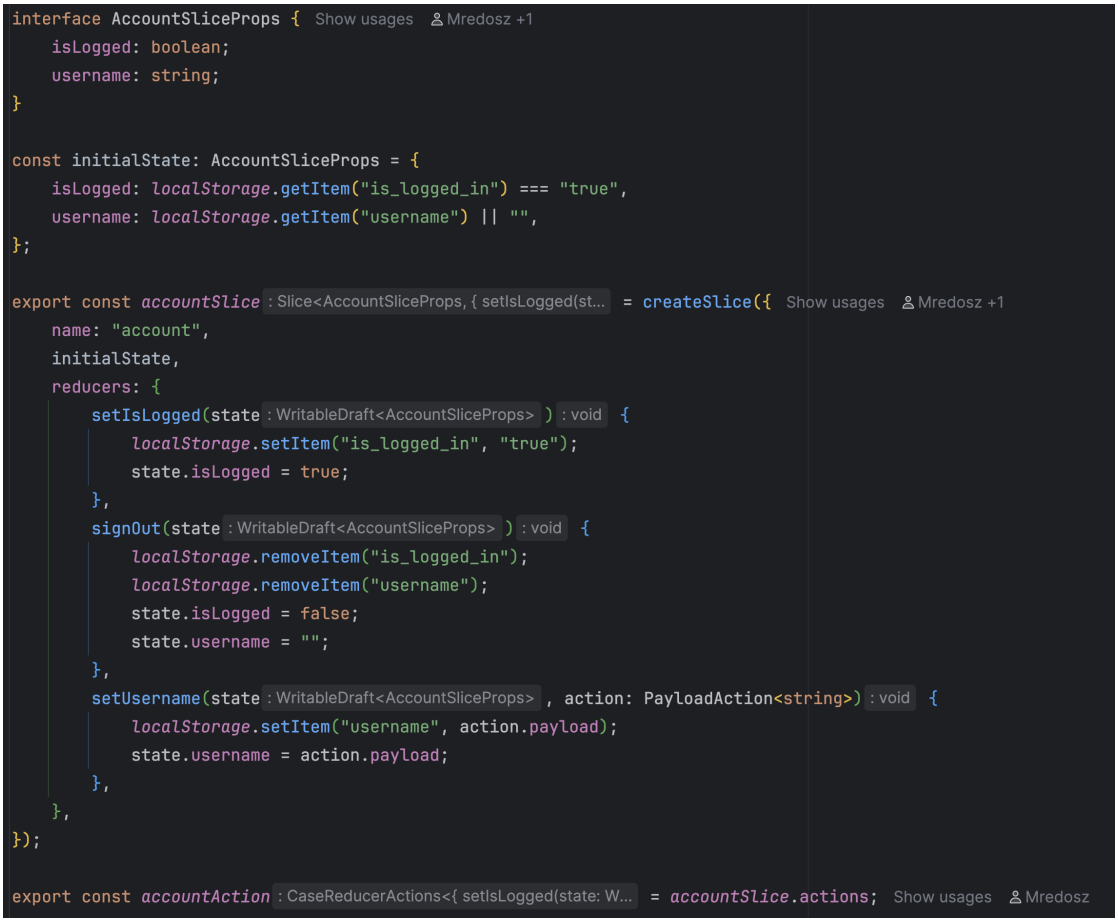
Z racji tego, że korzystamy z Reacta w połączeniu z TypeScriptem, przygotowaliśmy również własne hooki wspomagające typowanie, takie jak `useDispatchTyped` oraz `useSelectorTyped`. Pozwalają one na bezpieczne typowanie akcji oraz selek-

torów Reduxa bez konieczności powtarzania adnotacji typów w każdym komponencie. Fragmenty tej implementacji przedstawiono na rysunkach 7.6 oraz 7.7.

```
const store : EnhancedStore<{ account: AccountSliceProp... } = configureStore({
    reducer: {
        account: accountSlice.reducer,
        notification: notificationSlice.reducer,
        spotDetails: spotDetailsModalSlice.reducer,
        searchedSpotsListModal: searchedSpotListModalSlice.reducer,
        expandedSpotMediaGallery: expandedSpotMediaGallerySlice.reducer,
        spotFilters: spotFiltersSlice.reducer,
        chats: chatsSlice.reducer,
        map: mapSlice.reducer,
        sidebar: sidebarSlice.reducer,
        searchedSpots: searchedSpotsSlice.reducer,
        social: socialSlice.reducer,
        spotComments: spotCommentSlice.reducer,
        currentViewSpots: currentViewSpotsSlice.reducer,
        currentViewSpotsListModal: currentViewSpotsListModalSlice.reducer,
        currentViewSpotsParams: currentViewSpotParamsSlice.reducer,
        spotWeather: spotWeatherSlice.reducer,
        expandedSpotGalleryMediaList: expandedSpotGalleryMediaListSlice.reducer,
        expandedSpotMediaGalleryModals:
            expandedSpotMediaGalleryModalsSlice.reducer,
        expandedSpotMediaGalleryFullscreenSizeModal:
            expandedSpotMediaGalleryFullscreenSizeSlice.reducer,
        expandedSpotGalleryCurrentMedia:
            expandedSpotGalleryCurrentMediaSlice.reducer,
    },
});

export default store; Show usages  Mredosz
export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Rysunek 7.6: Konfiguracja sklepu (`Redux store`)



Rysunek 7.7: Przykładowy slice odpowiedzialny za sprawdzenie czy użytkownik jest zalogowany

7.2.3 Integracja i komunikacja z backendem

7.2.4 Style

7.2.5 Strona główna

7.2.6 Mapa

7.2.7 Chat

7.2.8 Forum

7.2.9 Konto użytkownika

7.2.10 Panel logowania

7.3 Implementacja CI/CD

Rozdział 8

Testy

8.1 Testy jednostkowe

8.2 Testy integracyjne

8.3 Testy E2E

8.4 Wyniki testów i wnioski

Rozdział 9

Prezentacja systemu

9.1 Strona główna

9.2 Strona mapy

9.3 Strona chatu

9.4 Strona forum

9.5 Panel logowania

9.6 Panel konta użytkownika

Rozdział 10

Nakład pracy

10.1 Ogólny nakład pracy

10.2 Indywidualne nakłady pracy

10.2.1 Adam Langmesser

10.2.2 Mateusz Redosz

Na projekt poświęciłem łącznie 324 godziny, z czego 237 przeznaczyłem na prace deweloperskie, 111 na pisanie dokumentacji, 19 godzin na Review kodu, 19 na spotkania dotyczące omówienia dalszych prac projektowych oraz przy pomocy innym członkom zespołu oraz 49 godzin poświęciłem nad stworzeniem widoków na figmie. Prace nad częścią deweloperską rozpocząłem 04.08.2024 a zakończyłem 08.09.2025. W projekcie pracowałem nad Rejestracją użytkownika, tokenenm JWT, częściową implementacją CI/CD, stroną główną, zaimplementowaniem Sidebara oraz podstroną dla użytkownika. Moje wylistowane zadania z Jira:

1. Dokumentacja
    - TODO
2. Design
    - Ustalić paletę kolorystyczną

- Propozycja wyglądu

3. Backend i Frontend

    - Formularz rejestracji
    - Routing
    - Formatowanie w React (prettier)
    - Obsługa JWT na frontend
    - oAuth Frontend
    - Update JWT
    - Refactor JWT
    - Stworzenie komponentu Notification i poprawa błędów
    - Implementacja pierwszych testów
    - Zaimplementowanie kolejki w komponencie notification
    - Dodanie reduxa do rejestracji
    - Zmiana sposobu pobierania danych o spotach
    - Obsługa customowych błędów z jakarta.validation
    - Obsługa auto wylogowania przy starcie
    - Domyślna wiadomość w notification
    - Poprawa headera
    - Ciemny motyw
    - Refactor pogody
    - Propozycja wyglądu
    - Przeniesienie zdjęć z google drive
    - Dodać Type script do Reacta
    - Aktualizacja tailwinda i dodanie kolorów
    - Podstawowy Sidebar

- Strona główna z prostymi filtrami
- Strona główna z zaawansowanymi filtrami
- Sidebar
- Strona profilu
- Ustawienia
- Listy spotów
- Lista zdjęć
- Lista filmów
- Lista znajomych
- Dodanie spotów
- Lista komentarzy
- Strona główna profilu
- Listy
- Poprawa Sidebara
- Zmiana kropki na przyciemnienie tła na Sidebar
- Poprawa strony do logowania i rejestracji
- Usunięcie username z account Redux
- Dodanie zamknięcia Sidebara na małych ekranach po kliknięciu nav linka
- Poprawić tooltipa na sidebar
- Zmiana sposobu pobierania username na backendzie z tokena jwt
- Paginacja z infinity scrollem
- Lista zdjęć innego usera
- Walidacja i responsywność w dodaniu spotów
- Dodanie sortowania i filtrów na zaawansowanej stronie
- Zmiana na infinity scrola

- Zmiana zdjęcia profilowego użytkownika
- Czyszczenie formularza w dodawaniu spota
- Dodanie wyszukiwarki znajomych w Social
- Zatwierdzenie przez drugiego użytkownika dodania do znajomych
- Sprawdzenie czy wszystko działa i poprawki Mateusz

4. CI/CD

    - Dodanie testów z frontendu do github actions
    - Poprawa github actions
    - Poprawa pipeline od Javy i Reacta

5. Praca dyplomowa

    - Uzupełnienie informacji o zespole i podział na rozdziały

10.2.3 Stanisław Oziemczuk

10.2.4 Kacper Badek

Rozdział 11

Podsumowanie

11.1 Osiągnięte rezultaty

11.2 Napotkane wyzwania

11.3 Plany na przyszłość

Załączniki

Płyta CD z następującą zawartością:

- *pliki projektowe* – pliki składające się na całość projektu
  - repozytorium kodu źródłowego wraz z instrukcją zbudowania i uruchomienia projektu
  - źródło pracy inżynierskiej.
- *Langmesser Adam_Redosz Mateusz_Oziemczuk Stanisław_Badek Kacper_praca pisemna* – katalog zawierający plik PDF z pracą inżynierską.