



# POLSKO-JAPONSKA 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 doświadczeniem 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 stylów 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

<b>Temat projektu:</b> Aplikacja webowa: spotty-na-drony.pl <b>Temat projektu po angielsku:</b> Web application: spotty-na-drony.pl	<b>Akronim:</b> Merkury <b>Data ustalenia tematu</b> 2023-10-10
<b>Promotor:</b>  mgr Adam Urbanowicz	<b>Konsultanci:</b>  1. — brak —
<b>Cele projektu:</b> Stworzenie w pełni funkcjonalnej aplikacji internetowej do rozwijania hobby(latania dronem).	
<b>Rezultaty projektu:</b> 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.	
<b>Miary sukcesu:</b> Gotowa do wdrożenia aplikacja. Realizacja w terminie zgodnym z wymaganiami.	
<b>Ograniczenia:</b> 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

<b>Data ukończenia projektu:</b> 10 listopada 2025	<b>Recenzent:</b> dr Elżbieta Puźniakowska-Gałuch
---	--

# Spis treści

<b>1 Wstęp</b>	<b>6</b>
1.1 O projekcie . . . . .	6
1.2 Cel i zakres prac . . . . .	6
1.3 Geneza pomysłu . . . . .	6
<b>2 Opis problemu</b>	<b>7</b>
2.1 Rich picture . . . . .	7
2.2 Udziałowcy . . . . .	7
2.3 Istniejące rozwiązania . . . . .	7
2.4 Wizja rozwiązania . . . . .	7
2.5 Aspekty społeczne i biznesowe . . . . .	7
2.5.1 Aspekty społeczne . . . . .	7
2.5.2 Aspekty biznesowe . . . . .	7
<b>3 Planowanie</b>	<b>8</b>
3.1 Metodologia pracy . . . . .	8
3.1.1 Przegląd rozważanych podejść . . . . .	8
3.1.2 Odrzucone podejścia . . . . .	8
3.1.3 Wybrane podejście: Disciplined Agile Delivery (Lean Life Cycle) . . . . .	9
3.1.4 Narzędzia i komunikacja . . . . .	9
3.1.5 Podział ról w zespole . . . . .	10
3.2 Harmonogram projektu . . . . .	10
3.3 Technologie i narzędzia . . . . .	12
3.3.1 Technologie . . . . .	12

3.3.2	Narzędzia . . . . .	12
3.4	Zasoby i ograniczenia . . . . .	12
3.4.1	Zasoby . . . . .	12
3.4.2	Ograniczenia . . . . .	12
3.5	Analiza ryzyka . . . . .	12
<b>4</b>	<b>Analiza wymagań</b>	<b>13</b>
4.1	Przypadki użycia . . . . .	14
4.1.1	Aktorzy . . . . .	14
4.1.2	Diagram przypadków użycia . . . . .	14
4.1.3	Scenariusz przypadków użycia . . . . .	14
4.2	Wymagania ogólne i dziedzinowe . . . . .	14
4.3	Wymagania funkcjonalne . . . . .	14
4.3.1	Funkcjonalności dla mapy . . . . .	14
4.3.2	Funkcjonalności dla chatu . . . . .	14
4.3.3	Funkcjonalności dla forum . . . . .	14
4.3.4	Funkcjonalności dla konta użytkownika . . . . .	14
4.3.5	Funkcjonalności dla logowania i rejestracji . . . . .	14
4.3.6	Funkcjonalności dla wyszukiwarki spotów . . . . .	14
4.3.7	Funkcjonalności dla motywu . . . . .	14
4.4	Wymagania pozafunkcjonalne . . . . .	14
4.5	Wymagania interfejs z otoczeniem . . . . .	14
4.6	Wymagania na środowisko docelowe . . . . .	14
<b>5</b>	<b>Projekt</b>	<b>15</b>
5.1	Wzorce projektowe . . . . .	15
5.2	Architektura systemu . . . . .	15
5.2.1	Diagram architektury . . . . .	15
5.2.2	Komponenty systemu . . . . .	15
5.3	Projekt bazy danych . . . . .	15
5.3.1	Model danych . . . . .	15
5.3.2	Diagram ERD . . . . .	15

5.4	Architektura interfejsu użytkownika . . . . .	15
5.4.1	Projekt strony głównej . . . . .	15
5.4.2	Projekt panelu logowania . . . . .	15
5.4.3	Projekt mapy . . . . .	15
5.4.4	Projekt chatu . . . . .	15
5.4.5	Projekt forum . . . . .	15
5.4.6	Projekt konta użytkownika . . . . .	15
<b>6</b>	<b>Przebieg realizacji projektu</b>	<b>16</b>
6.1	Sprint 1 . . . . .	16
6.2	Sprint 2 . . . . .	16
<b>7</b>	<b>Realizacja Projektu</b>	<b>17</b>
7.1	Implementacja backendu . . . . .	17
7.1.1	Struktura projektu . . . . .	17
7.1.2	Endpointy systemu . . . . .	17
7.1.3	Integracja z bazą danych . . . . .	20
7.1.4	Obsługa uwierzytelnienia . . . . .	20
7.1.5	Konteneryzacja . . . . .	20
7.2	Implementacja frontendu . . . . .	20
7.2.1	Struktura aplikacji . . . . .	20
7.2.2	Zarządzanie stanem i przepływ danych . . . . .	25
7.2.3	Integracja i komunikacja z backendem . . . . .	27
7.2.4	Style . . . . .	30
7.2.5	Strona główna . . . . .	34
7.2.6	Mapa . . . . .	34
7.2.7	Chat . . . . .	34
7.2.8	Forum . . . . .	34
7.2.9	Konto użytkownika . . . . .	34
7.2.10	Panel logowania . . . . .	34
7.3	Implementacja CI/CD . . . . .	34

<b>8 Testy</b>	<b>35</b>
8.1 Testy jednostkowe . . . . .	35
8.2 Testy integracyjne . . . . .	35
8.3 Testy E2E . . . . .	35
8.4 Wyniki testów i wnioski . . . . .	35
<b>9 Prezentacja systemu</b>	<b>36</b>
9.1 Strona główna . . . . .	36
9.2 Strona mapy . . . . .	36
9.3 Strona chatu . . . . .	36
9.4 Strona forum . . . . .	36
9.5 Panel logowania . . . . .	36
9.6 Panel konta użytkownika . . . . .	36
<b>10 Nakład pracy</b>	<b>37</b>
10.1 Ogólny nakład pracy . . . . .	37
10.2 Indywidualne nakłady pracy . . . . .	37
10.2.1 Adam Langmesser . . . . .	37
10.2.2 Mateusz Redosz . . . . .	37
10.2.3 Stanisław Oziemczuk . . . . .	40
10.2.4 Kacper Badek . . . . .	40
<b>11 Podsumowanie</b>	<b>41</b>
11.1 Osiągnięte rezultaty . . . . .	41
11.2 Napotkane wyzwania . . . . .	41
11.3 Plany na przyszłość . . . . .	41
<b>12 Słownik pojęć i skrótów</b>	<b>42</b>
<b>Bibliografia</b>	<b>46</b>
<b>Załączniki</b>	<b>47</b>

# Rozdział 1

## Wstęp

1.1 O projekcie

1.2 Cel i zakres prac

1.3 Geneza pomysłu

# 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

#### 3.1.1 Przegląd rozważanych podejść

Przy wyborze metodologii pracy rozważono trzy podejścia do prowadzenia projektu informatycznego:

- klasyczny Agile (w praktyce: Scrum),
- model kaskadowy (Waterfall),
- Disciplined Agile Delivery - Lean Life Cycle.

#### 3.1.2 Odrzucone podejścia

**„Klasyczny Agile” (Scrum).** Mimo elastyczności i popularności zakłada pracę w iteracjach 2–4 tygodni oraz stały zestaw ceremonii (planowanie, przegląd, retrospekcja). Ze względu na nierównomierną dostępność zasobów w kolejnych miesiącach studiów nie zapewniono możliwości utrzymania stałej kadencji sprintów, dlatego z podejścia zrezygnowano.

**Model kaskadowy (Waterfall).** Przewiduje sekwencyjne przechodzenie przez z góry określone etapy i ogranicza bieżącą weryfikację wymagań w trakcie prac deweloperskich. W projekcie wymagano możliwości częstych rewizji założeń oraz

wprowadzania istotnych zmian w docelowej wizji rozwiązania; dlatego z podejścia zrezygnowano.

### **3.1.3 Wybrane podejście: Disciplined Agile Delivery (Lean Life Cycle)**

Podjęto decyzję o zastosowaniu **Disciplined Agile Delivery** [1] w wariantie **Lean Life Cycle** [2], ponieważ podejście to łączy pożądane cechy Agile i Waterfall, a jednocześnie eliminuje stałe sprinty na rzecz pracy w ciągłym przepływie.

#### **Kluczowe argumenty wyboru:**

- **Brak sprintów.** Zastosowano przepływ ciągły, co pozwala dopasować tempo do zmiennej dostępności zespołu i unikać sztucznego „domykania” iteracji.
- **Rozbudowana faza startowa.** Na początku przewidziano większy wysiłek planistyczny: doprecyzowanie zakresu, wstępna wizja architektury, identyfikacja ryzyk, plan publikacji oraz kryteria jakości – bez zamrażania szczegółów.
- **Ciągła weryfikacja wymagań.** W trakcie realizacji przewidziano bieżące doprecyzowywanie backlogu, regularny feedback promotora oraz możliwość korygowania kierunku bez kosztów „przeskakiwania” między fazami.
- **Praktyki Lean i koncentracja na wartości.** Priorytetyzacja wartości biznesowej, wizualizacja pracy, małe partie dostaw.
- **Lekka governance i kamienie milowe.** Zastosowano lekkie mechanizmy nadzoru (peer review, prezentacje postępów) zapewniające przejrzystość bez nadmiernej biurokracji.

### **3.1.4 Narzędzia i komunikacja**

Do zarządzania zadaniami zastosowana została **Jira** (monitorowanie postępu prac oraz ewidencja zadań członków zespołu). Komunikację w zespole zaplanowano w

formie regularnych spotkań oraz asynchronicznie z wykorzystaniem **Discorda** oraz **Messengera**.

### 3.1.5 Podział ról w zespole

- Adam
- Stanisław
- Kacper
- Mateusz

Każdy z członków zespołu uczestniczy również w przygotowaniu dokumentacji.

## 3.2 Harmonogram projektu

W poniższym harmonogramie przedstawiono plan prac nad poszczególnymi częściami projektu, rozłożony na miesiące.

### Rok 2024

**Czerwiec**     • Zebranie zespołu.

- Rozważenie potencjalnych pomysłów.

**Lipiec**     • Wybór technologii.

- Wstępne założenia architektoniczne.

**Sierpień**     • *(do uzupełnienia)*

- *(do uzupełnienia)*

**Wrzesień**     • *(do uzupełnienia)*

- *(do uzupełnienia)*

**Październik**     • *(do uzupełnienia)*

- *(do uzupełnienia)*

**Listopad** • (do uzupełnienia)

- (do uzupełnienia)

**Grudzień** • (do uzupełnienia)

- (do uzupełnienia)

## **Rok 2025**

**Styczeń** • (do uzupełnienia)

- (do uzupełnienia)

**Luty** • (do uzupełnienia)

- (do uzupełnienia)

**Marzec** • (do uzupełnienia)

- (do uzupełnienia)

**Kwiecień** • (do uzupełnienia)

- (do uzupełnienia)

**Maj** • (do uzupełnienia)

- (do uzupełnienia)

**Czerwiec** • (do uzupełnienia)

- (do uzupełnienia)

**Lipiec** • (do uzupełnienia)

- (do uzupełnienia)

**Sierpień** • (do uzupełnienia)

- (do uzupełnienia)

**Wrzesień** • (do uzupełnienia)

- (do uzupełnienia)

**Październik** • (*do uzupełnienia*)

- (*do uzupełnienia*)

**Listopad** • (*do uzupełnienia*)

- (*do uzupełnienia*)

**Grudzień** • (*do uzupełnienia*)

- (*do uzupełnienia*)

## Rok 2026

**Styczeń** • (*do uzupełnienia*)

- (*do uzupełnienia*)

### 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<sup>14</sup>

# 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 UserNotFoundException.

Przykładowa odpowiedź (200 OK):

```
1  {
2      "username": "john_doe",
3      "profilePhoto": "https://cdn.example.com/profiles/john_doe.
4          jpg",
5      "followersCount": 125,
6      "followedCount": 87,
7      "friendsCount": 32,
8      "photosCount": 58,
9      "mostPopularPhotos": [
10         {
11             "src": "https://cdn.example.com/photos/123.jpg",
12             "heartsCount": 240,
```

```
12      "viewsCount": 3400,
13      "title": "Sunset at the beach",
14      "id": 123
15    }
16  ]
17 }
```

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**

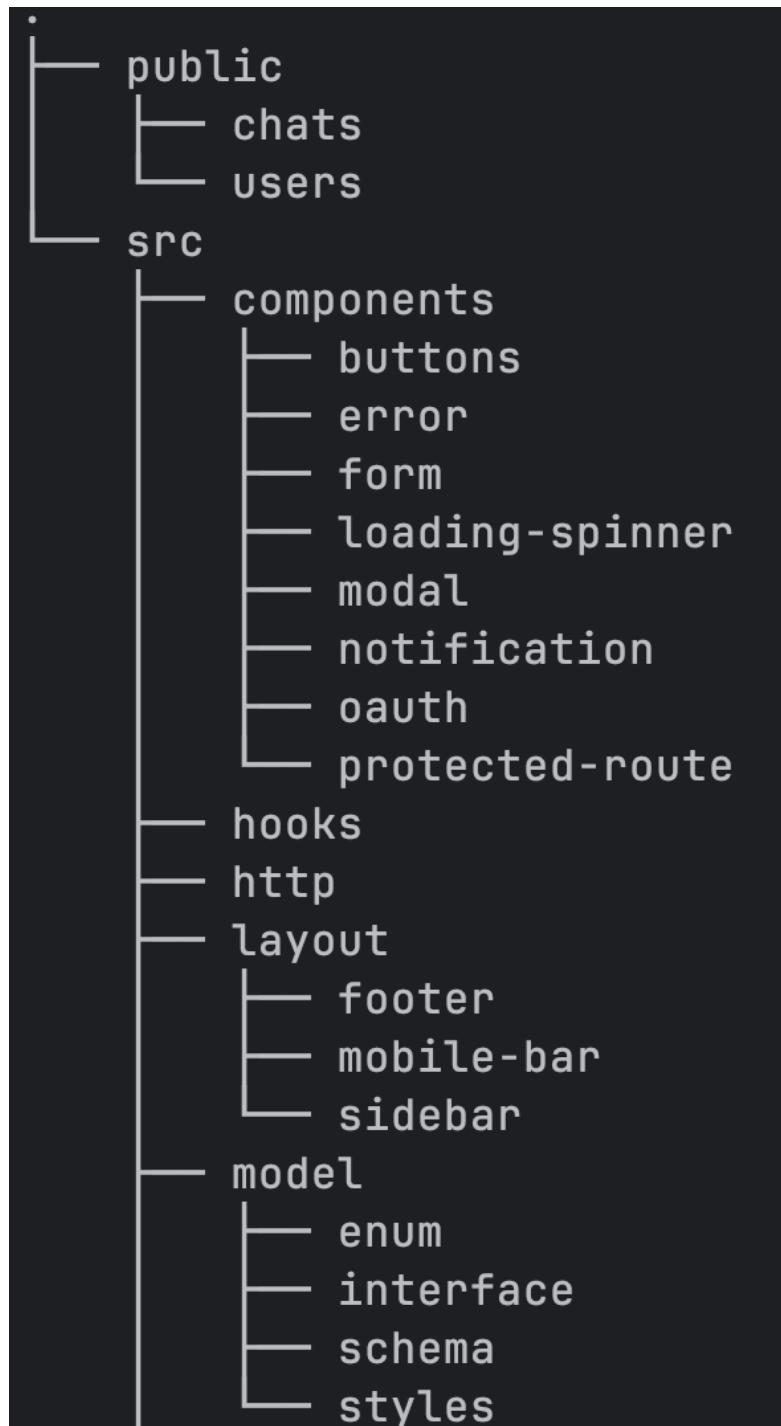
**7.1.4 Obsługa uwierzytelnienia**

**7.1.5 Konteneryzacja**

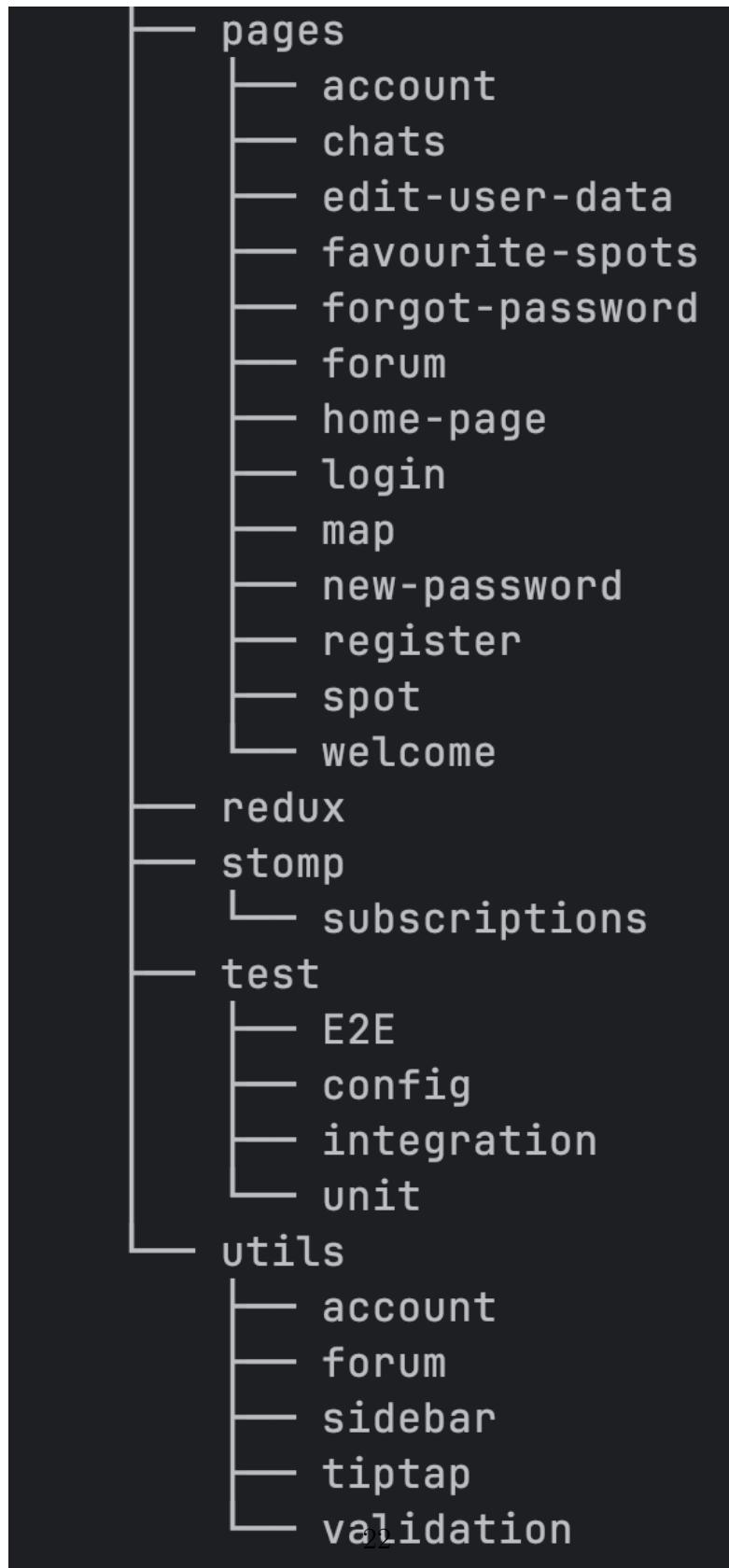
## **7.2 Implementacja frontendu**

### **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 jest przedstawione 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)

```
        {
          path: "new-password",
          element: <NewPassword />,
        },
        {
          path: "forum",
          element: <Forum />,
        },
        {
          path: "forum/:postId/:slugTitle?",
          element: <ForumThread />,
        },
        {
          path: "map",
          element: <MapPage />,
        },
        {
          path: "chat",
          element: (
            <ProtectedRoute>
              <ChatsPage />
            </ProtectedRoute>
          ),
        },
      ],
    );
  }

export default router;
```

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 isLoggedIn = useSelector((state) => state.account.isLoggedIn);

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

Rysunek 7.5: Implementacja komponentu bramki (`ProtectedRoute`)

glossaries

### 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 Hook (React) `useState`) [3], jak i ze Stanu globalnego, utrzymywanego przez Bibliotekę React Redux [4]. 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 podległych), wykorzystujemy Hook (React) `useState`. Natomiast efekty uboczne i synchronizację realizujemy za pomocą `useEffect`. W przypadku bardziej złożonej logiki lub potrzeby ponownego wykorzystania kodu powstały Hook (React)i niestandardowe, takie jak `useScreenSize`, `useDarkMode` czy `useClickOutside`. Dzięki temu większość logiki prezentacji 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 Hook (React)i wspomagające typowanie, takie jak `useDispatchTyped` oraz `useSelectorTyped`. Pozwalają one na bezpieczne typowanie akcji oraz selektorów 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)

```

interface AccountSliceProps { Show usages & Mredosz +1
  isLoggedIn: boolean;
  username: string;
}

const initialState: AccountSliceProps = {
  isLoggedIn: localStorage.getItem("is_logged_in") === "true",
  username: localStorage.getItem("username") || "",
};

export const accountSlice : Slice<AccountSliceProps, { setisLoggedIn(st... = createSlice({ Show usages & Mredosz +1
  name: "account",
  initialState,
  reducers: {
    setIsLoggedIn(state : WritableDraft<AccountSliceProps>) : void {
      localStorage.setItem("is_logged_in", "true");
      state.isLoggedIn = true;
    },
    signOut(state : WritableDraft<AccountSliceProps>) : void {
      localStorage.removeItem("is_logged_in");
      localStorage.removeItem("username");
      state.isLoggedIn = false;
      state.username = "";
    },
    setUsername(state : WritableDraft<AccountSliceProps>, action: PayloadAction<string>) : void {
      localStorage.setItem("username", action.payload);
      state.username = action.payload;
    },
  },
},
);
};

export const accountAction : CaseReducerActions<{ setisLoggedIn(state: W... = accountSlice.actions; Show usages & Mredosz

```

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

### 7.2.3 Integracja i komunikacja z backendem

Jest to kluczowy element aplikacji, ponieważ wymaga bezpiecznego przesyłania danych użytkownika. W celu uproszczenia komunikacji z serwerem skorzystaliśmy z biblioteki `axios` [5] oraz Biblioteki `TanStack Query` [6]. We wszystkich ścieżkach, które wymagają aby użytkownik był zalogowany, do zapytania dołączany jest token JWT. Token jest przekazywany w ciasteczku dzięki ustawieniu parametru `withCredentials` na wartość `true`. Przykładem pliku odpowiedzialnego za taką komunikację jest `account.js` (rys. 7.8 i 7.9), który obsługuje operacje

związanego z logowaniem, rejestracją, zmianą hasła oraz wylogowaniem.

```
export async function loginUser(user) { Show usages new *
  return await axios.post(` ${BASE_URL} /public/account/login` , user , {
    withCredentials: true ,
  });
}

export async function registerUser(user) { Show usages Adam Langmesser +2
  return await axios.post(` ${BASE_URL} /public/account/register` , user , {
    withCredentials: true ,
  });
}

export async function sentEmailWithNewPasswordLink(email) { Show usages Adam Langmesser +1
  console.log("sending email...");
  return await axios.post(
    `${BASE_URL}/public/account/forgot-password` ,
    email ,
    {
      headers: {
        "Content-Type": "text/plain" ,
      },
    },
  );
}
```

Rysunek 7.8: Implementacja modułu account (1)

```

export async function changePassword(userData) { Show usages  ↳ stanoz +1
    return await axios.post(
        `${BASE_URL}/public/account/set-new-password`,
        userData,
    );
}

export async function logout() { Show usages  ↳ stanoz +1
    await axios.post(
        `${BASE_URL}/account/oauth2/logout`,
        {},
        {
            withCredentials: true,
        },
    );
}

export const googleLoginUrl = `${BASE_URL}/oauth2/authorization/google`; Show usages  ↳ stanoz
export const githubLoginUrl = `${BASE_URL}/oauth2/authorization/github`; Show usages  ↳ stanoz

```

Rysunek 7.9: Implementacja modułu `account` (2)

Funkcje odpowiedzialne za komunikację z backendem zostały umieszczone w katalogu `/http`. Dzięki temu są one skoncentrowane i mogą być w prosty sposób wykorzystywane w różnych częściach aplikacji. Zastosowaliśmy TanStack Query, ponieważ znaczco ogranicza on powtarzalny kod oraz upraszcza obsługę błędów i stanów zapytania (takich jak ładowanie danych, błąd, sukces). udostępnia m.in. wartość `isLoading`, dzięki czemu komponent może łatwo wyświetlić ekran ładowania bez ręcznego zarządzania własnym stanem. Dodatkowo Hook (React) `useQuery` z tej Biblioteki umożliwia automatyczne pobieranie danych po wejściu na daną podstronę. Oznacza to, że komponent deklaruje jedynie „jakie dane są mu potrzebne”, a TanStack Query zajmuje się ich pobraniem, cache’owaniem oraz odświeżaniem. Do operacji, które wymagają wywołania akcji po stronie użytkownika (np. wysłania formularza logowania), wykorzystujemy Hook (React) `useMutation` z TanStack Query. Przykład użycia tego rozwiązania w procesie logowania został przedstawiony na rys. 7.10.

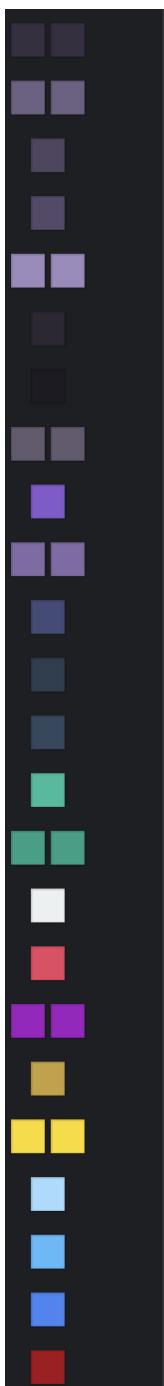
```
const { mutateAsync, isSuccess, error } = useMutation({
  mutationFn: loginUser,
});

const handleSubmit: (event: FormEvent<HTMLFormElement>) => Promise<void> = async (event) => {
  event.preventDefault();
  await mutateAsync({
    username: enteredValue.username,
    password: enteredValue.password,
  });
  navigate(-1);
};
```

Rysunek 7.10: Wykorzystanie TanStack Query przy logowaniu użytkownika

#### 7.2.4 Style

Do stylowania interfejsu wykorzystaliśmy Framework Tailwind CSS [7]. Dzięki gotowym klasom udostępnianym przez Tailwind mogliśmy definiować wygląd elementów bezpośrednio w kodzie komponentu, bez konieczności przechodzenia do osobnych plików ze stylami. Ułatwia to zarówno tworzenie widoków, jak i późniejsze modyfikacje — w przypadku zmiany stylu dokładnie wiadomo, gdzie należy jej dokonać. Korzystanie ze zdefiniowanych klas pozwoliło nam również zachować spójność wizualną w całej aplikacji. W pliku `index.css` zdefiniowaliśmy zmienne kolorystyczne (rys. 7.11 i 7.12). Dzięki temu zmiana motywu kolorystycznego w przyszłości sprowadza się do edycji wartości w jednym miejscu.



	--color-violetDark: #363041;
	--color-violetLight: #6d6183;
	--color-violetLightDarker: #4f4660;
	--color-violetLightDark: #554a69;
	--color-violetLighter: #9b8cbd;
	--color-violetDarker: #2c2734;
	--color-violetHeavyDark: #1e1b23;
	--color-violetBtnBorderDark: #625b6e;
	--color-violetBright: #835ace;
	--color-darkVioletBtnOutline: #816ba6;
	--color-mediumDarkBlue: #424b77;
	--color-first: #2c3e50;
	--color-second: #34495e;
	--color-third: #1abc9c;
	--color-fourth: #16a085;
	--color-fifth: #ecf0f1;
	--color-sixth: #e94560;
	--color-magenta: #a01bc1;
	--color-darkYellow: #c5a03c;
	--color-ratingStarColor: #fadbd1;
	--color-locationMarkerDarkerBlue: #a3dcff;
	--color-locationMarkerLightBlue: #52bafb;
	--color-userLocationDot: #4285f4;
	--color-spotLocationMarker: #a8071a;

Rysunek 7.11: Implementacja zmiennych kolorystycznych (1)



Rysunek 7.12: Implementacja zmiennych kolorystycznych (2)

W niektórych miejscach konieczne było zapisanie stylów w czystym CSS, ponieważ część użytych Bibliotek tego wymagała. W innych przypadkach wystarczyło skorzystać z klas zdefiniowanych w `index.css` oraz klas Tailwinda. Cała aplikacja

jest Responsywna. Tailwind udostępnia predefiniowane prefiksy Responsywne (np. `md:`, `lg:`) (rys. 7.13), stworzyliśmy również własny (`3xl:`) na ekrany o rozdzielczości 2560px. Pozwalają one przypisywać style zależnie od szerokości ekranu bez pisania własnych reguł @media. Dzięki temu implementacja widoków mobilnych i desktopowych była znaczco szybsza.

```
<div className="mt-17 flex flex-col items-center gap-7 lg:mt-0 lg:-ml-40 lg:flex-row xl:-ml-42 xl:gap-10 2xl:-ml-80">
  <div className="relative">
    <img alt="profileImage"
      src={userData?.profilePhoto}
      className="dark:drop-shadow-darkBgMuted aspect-square h-64 rounded-full
      shadow-md sm:h-80 lg:h-85 xl:h-96 dark:drop-shadow-md"
    />
```

Rysunek 7.13: Przykładowe użycie klas Tailwind (w tym prefiksów responsywności)

Tailwind został też wykorzystany do obsługi trybu jasnego i ciemnego. Wystarczy dodać klasę z prefiksem `dark:` (np. `dark:bg-black`), aby zmienić kolorystykę elementu, gdy aplikacja jest w trybie ciemnym (rys. 7.14).

```
<input
  id={id}
  value={value}
  type={type}
  onChange={onChange}
  onFocus={setFocusedToTrue}
  onBlur={handleOnBlur}
  className="dark:bg-darkBgMuted bg-lightBgMuted dark:text-darkText text-lightText w-full
  rounded-md p-2 shadow-md focus:outline-none dark:shadow-black/50"
/>
```

Rysunek 7.14: Przykładowe użycie klas Tailwind (w tym wariantu `dark:`)

Aby uzyskać płynniejsze i przyjemniejsze animacje, wykorzystaliśmy Bibliotekę Motion [8]. Pozwala ona w prosty sposób tworzyć animacje elementów interfejsu, bez potrzeby ręcznego pisania złożonych reguł CSS. W naszej aplikacji użyliśmy jej m.in. w polach formularza logowania i rejestracji (rys. 7.15). Na początku etykieta pola (np. „username”) jest wyświetlana wewnątrz pola tekstowego, natomiast po

kliknięciu w pole jest płynnie przesuwana nad to pole, co poprawia czytelność i ergonomię formularza.

```
<div className="relative">
  <motion.label
    htmlFor={id}
    initial={false}
    animate={{
      top: shouldFloat ? "-0.7rem" : "0.5rem",
      left: "0.75rem",
      fontSize: shouldFloat ? "0.75rem" : "1rem",
      opacity: shouldFloat ? 1 : 0.6,
    }}
    transition={{ type: "spring", stiffness: 300, damping: 25 }}
    className="■dark:text-darkText □text-lightText pointer-events-none absolute z-10 px-1 capitalize"
  >
    {label}
  </motion.label>
  <input
    id={id}
    value={value}
    type={type}
    onChange={onChange}
    onFocus={setFocusedToTrue}
    onBlur={handleOnBlur}
    className="■ dark:bg-darkBgMuted □bg-lightBgMuted ■dark:text-darkText □text-lightText w-full
    rounded-md p-2 shadow-md focus:outline-none dark:shadow-black/50"
  />
```

Rysunek 7.15: Implementacja animacji z wykorzystaniem Motion

### 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 rozpoczęłem 04.08.2024 a zakończyłem 08.09.2025. W projekcie pracowałem nad Rejestracją użytkownika, tokenem 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 Sidebaru
- Zmiana kropki na przyciemnienie tła na Sidebar
- Poprawa strony do logowania i rejestracji
- Usunięcie username z account Redux
- Dodanie zamknięcia Sidebaru na małych ekranach po kliknięciu nav linka
- Poprawić tooltipa na sidebar
- Zmiana sposobu pobierania username na backendzie z tokena jwt
- Paginacja z infinity scrolllem
- 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

# Rozdział 12

## 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, 38

**Biblioteka** Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. 23, 25, 27, 29, 32, 33

**CI/CD** Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. 37, 40

**CSS** Kaskadowe arkusze stylów (Cascading Style Sheets) — język opisu prezentacji dokumentów (np. HTML). Definiuje wygląd interfejsu: układ, kolory, typografię, odstępy, animacje i zachowania responsywne, oddzielając warstwę treści od warstwy prezentacji.. 32, 33

**Design** Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). 37

**Disciplined Agile Delivery - Lean Life Cycle** Kanbanowy tryb pracy bez sprintów: zespół pobiera małe zadania z tablicy, pilnuje limitów pracy w toku i często wdraża małe zmiany. Planowanie jest lekkie i na bieżąco; priorytet mają rzeczy o najwyższej wartości. Skupiamy się na płynności—skracamy się

czas przejścia zadań, usuwa blokady, usprawnia proces małymi krokami. Jakość jest wbudowana: code review, testy automatyczne i prosta definicja „gotowe”. 8

**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. 20

**Framework** Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. 2, 30

**Frontend** Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. 2, 38

**Hook (React)** Prosta funkcja w React, która „dodaje” możliwości do elementu interfejsu — np. pozwala mu coś zapamiętać (stan) albo zrobić coś po zmianie/załadowaniu. Wszystkie hooki zaczynają się od `use...` (np. `useState`, `useEffect`).. 29

**JWT** Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. 27, 37

**Media queries** Konstrukcja CSS pozwalająca stosować reguły stylów w zależności od cech urządzenia/okna (np. szerokości ekranu, orientacji, preferencji użytkownika). Podstawa responsywnego projektowania (*responsive design*).. 44

**Props** Właściwości przekazywane do komponentu React przez komponent nadzędny; służą do konfiguracji i przekazywania danych. Powinny być traktowane

wane jako tylko do odczytu (read-only) wewnątrz komponentu potomnego..  
25

**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 przekierowany (np. na stronę główną). 24

**React** Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. 25

**Redux** Biblioteka do przewidywalnego zarządzania stanem aplikacji. Opiera się na jednym *store*, akcjach i czystych *reducerach*, promuje niemutowalność i jednokierunkowy przepływ danych. Często używana z Reactem, ale niezależna od niego.. 25, 26

**Responsywność** Określenie związane z projektowaniem responsywnym (Responsive Web Design, RWD), czyli dostosowywaniem interfejsu do różnych rozmiarów i parametrów ekranów. Obejmuje m.in. elastyczne siatki, grafiki i Media queries, tak aby układ i czytelność były zachowane na telefonach, tabletach i desktopach.. 33

**Review kodu** Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. 37

**Sidebar** Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. 37–39

**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).. 25

**UI** Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. 25

# Bibliografia

- [1] *Disciplined Agile Delivery*. PMI. 1 sty. 2025. URL: <https://www.pmi.org/disciplined-agile/process/introduction-to-dad/why> (term. wiz. 30.10.2025).
- [2] *Disciplined Agile Delivery — Lean Life Cycle*. PMI. 1 sty. 2025. URL: <https://www.pmi.org/disciplined-agile/lifecycle/lean-lifecycle> (term. wiz. 30.10.2025).
- [3] *React useState*. 1 sty. 2025. URL: <https://react.dev/reference/react/useState> (term. wiz. 03.11.2025).
- [4] *Redux*. 1 sty. 2025. URL: <https://redux.js.org/> (term. wiz. 03.11.2025).
- [5] *Axios*. 1 sty. 2025. URL: <https://axios-http.com/> (term. wiz. 03.11.2025).
- [6] *Tanstack Query*. 1 sty. 2025. URL: <https://tanstack.com/query/latest> (term. wiz. 03.11.2025).
- [7] *Tailwind*. 1 sty. 2025. URL: <https://tailwindcss.com/> (term. wiz. 03.11.2025).
- [8] *Motion*. 1 sty. 2025. URL: <https://motion.dev/> (term. wiz. 03.11.2025).

# 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.