



# 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 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 stylu 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: spoty-na-drony.pl <b>Temat projektu po angielsku:</b> Web application: spoty-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 al- bumbu	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> 30 listopada 2025	<b>Recenzent:</b> dr Elżbieta Puźniakowska-Gałuch
---	--

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>6</b>
1.1	O projekcie . . . . .	6
1.2	Cel i zakres prac . . . . .	6
1.3	Geneza pomysłu . . . . .	6
<b>2</b>	<b>Opis problemu</b>	<b>7</b>
2.1	Rich picture . . . . .	7
2.2	Udziałowcy . . . . .	7
2.3	Istniejące rozwiązania . . . . .	8
2.4	Wizja rozwiązania . . . . .	8
2.5	Aspekty społeczne i biznesowe . . . . .	8
2.5.1	Aspekty społeczne . . . . .	8
2.5.2	Aspekty biznesowe . . . . .	8
<b>3</b>	<b>Planowanie</b>	<b>9</b>
3.1	Metodologia pracy . . . . .	9
3.1.1	Przegląd rozważanych podejść . . . . .	9
3.1.2	Odrzucone podejścia . . . . .	9
3.1.3	Wybrane podejście: Disciplined Agile Delivery (Lean Life Cycle) . . . . .	10
3.1.4	Narzędzia i komunikacja . . . . .	10
3.1.5	Podział ról w zespole . . . . .	11
3.2	Harmonogram projektu . . . . .	11
3.3	Technologie i narzędzia . . . . .	13
3.3.1	Technologie . . . . .	13

3.3.2	Narzędzia . . . . .	13
3.4	Zasoby i ograniczenia . . . . .	16
3.4.1	Zasoby . . . . .	16
3.4.2	Ograniczenia . . . . .	16
3.4.3	Usługi zewnętrzne . . . . .	16
3.5	Analiza ryzyka . . . . .	19
<b>4</b>	<b>Analiza wymagań</b>	<b>20</b>
4.1	Przypadki użycia . . . . .	21
4.1.1	Aktorzy . . . . .	21
4.1.2	Diagram przypadków użycia . . . . .	22
4.1.3	Scenariusz przypadków użycia . . . . .	22
4.2	Wymagania . . . . .	22
4.2.1	Wymagania ogólne . . . . .	23
4.2.1.1	Wymagania ogólne dla czatu . . . . .	23
<b>5</b>	<b>Projekt</b>	<b>29</b>
5.1	Wzorce projektowe . . . . .	29
5.2	Architektura systemu . . . . .	29
5.2.1	Diagram architektury . . . . .	29
5.2.2	Komponenty systemu . . . . .	29
5.3	Projekt bazy danych . . . . .	29
5.3.1	Model danych . . . . .	29
5.3.2	Diagram ERD . . . . .	29
5.4	Architektura interfejsu użytkownika . . . . .	29
5.4.1	Projekt strony głównej . . . . .	29
5.4.2	Projekt panelu logowania . . . . .	29
5.4.3	Projekt mapy . . . . .	29
5.4.4	Projekt chatu . . . . .	29
5.4.5	Projekt forum . . . . .	29
5.4.6	Projekt konta użytkownika . . . . .	29

<b>6</b>	<b>Przebieg realizacji projektu</b>	<b>30</b>
6.1	Sprint 1 . . . . .	30
6.2	Sprint 2 . . . . .	30
<b>7</b>	<b>Realizacja Projektu</b>	<b>31</b>
7.1	Implementacja backendu . . . . .	31
7.1.1	Struktura projektu . . . . .	31
7.1.2	Integracja z bazą danych . . . . .	31
7.1.3	Obsługa uwierzytelnienia . . . . .	31
7.1.4	Konteneryzacja . . . . .	31
7.2	Implementacja frontendu . . . . .	31
7.2.1	Struktura aplikacji . . . . .	31
7.2.2	Zarządzanie stanem i przepływ danych . . . . .	36
7.2.3	Integracja i komunikacja z backendem . . . . .	38
7.2.4	Style . . . . .	41
7.2.5	Wyszukiwarka spotów . . . . .	45
7.2.6	Mapa . . . . .	52
7.2.7	Chat . . . . .	52
7.2.8	Forum . . . . .	52
7.2.9	Konto użytkownika . . . . .	52
7.2.10	Panel logowania . . . . .	52
7.3	Implementacja CI/CD . . . . .	52
<b>8</b>	<b>Testy</b>	<b>53</b>
8.1	Testy jednostkowe . . . . .	53
8.2	Testy integracyjne . . . . .	53
8.3	Testy E2E . . . . .	53
8.4	Wyniki testów i wnioski . . . . .	53
<b>9</b>	<b>Prezentacja systemu</b>	<b>54</b>
9.1	Strona główna . . . . .	54
9.2	Strona mapy . . . . .	54
9.3	Strona chatu . . . . .	54

9.4	Strona forum . . . . .	54
9.5	Panel logowania . . . . .	54
9.6	Panel konta użytkownika . . . . .	54
<b>10</b>	<b>Nakład pracy</b>	<b>55</b>
10.1	Ogólny nakład pracy . . . . .	55
10.2	Indywidualne nakłady pracy . . . . .	55
10.2.1	Adam Langmesser . . . . .	55
10.2.2	Mateusz Redosz . . . . .	55
10.2.3	Stanisław Oziemczuk . . . . .	58
10.2.4	Kacper Badek . . . . .	58
<b>11</b>	<b>Podsumowanie</b>	<b>59</b>
11.1	Osiągnięte rezultaty . . . . .	59
11.2	Napotkane wyzwania . . . . .	59
11.3	Plany na przyszłość . . . . .	59
<b>12</b>	<b>Słownik pojęć i skrótów</b>	<b>60</b>
	<b>Spis tabel</b>	<b>67</b>
	<b>Bibliografia</b>	<b>68</b>
	<b>Załączniki</b>	<b>70</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

KARTA UDZIAŁOWCA	
Identyfikator:	UO1
Nazwa:	Zespół projektowy
Opis:	Zespół czterech studentów odpowiedzialnych za analizę, projekt, implementację, testy oraz dokumentację systemu.
Typ udziałowca:	ożywiony, bezpośredni
Punkt widzenia:	techniczna, wykonawcza
Ograniczenia:	Ograniczone zasoby czasowe i doświadczenie komercyjne.
Wymagania:	Wymagania funkcjonalne i techniczne systemu, możliwość realizacji w ramach projektu dyplomowego.

Tabela 2.1: Zespół projektowy

KARTA UDZIAŁOWCA	
Identyfikator:	UO2
Nazwa:	Promotor
Opis:	Osoba nadzorująca przebieg projektu, weryfikująca poprawność merytoryczną i zgodność z wymaganiami uczelni.
Typ udziałowca:	ożywiony, pośredni
Punkt widzenia:	merytoryczna, formalna, jakościowa
Ograniczenia:	Nie odpowiada za implementację; rekomenduje, opiniuje i zatwierdza.
Wymagania:	Czytelna dokumentacja, zgodność z wytycznymi kierunku i dobry poziom techniczny rozwiązania.

Tabela 2.2: Promotor

KARTA UDZIAŁOWCA	
Identyfikator:	UO3
Nazwa:	<a href="#">Droniarze</a>
Opis:	Główna grupa docelowa systemu – osoby latające dronami rekreacyjnie lub półprofesjonalnie, szukające miejsc do lotów i wymiany doświadczeń.
Typ udziałowca:	ożywiony, bezpośredni
Punkt widzenia:	użytkownik końcowy: prostota obsługi, rzetelne informacje o spotach, wygodne dzielenie się treściami.
Ograniczenia:	Brak wpływu na architekturę techniczną systemu; oczekują intuicyjnego interfejsu.
Wymagania:	Lista spotów, informacje o ograniczeniach prawnych, oceny/komentarze, dodawanie treści, podstawowe funkcje społecznościowe.

Tabela 2.3: Droniarze

## 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, retrospektywa). 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 wyma-

gano 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 wariancie **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 - fullstack developer, lider zespołu
- Stanisław - fullstack developer
- Kacper - fullstack developer
- Mateusz - fullstack developer

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**

Do realizacji projektu wykorzystano wiele technologii oraz narzędzi informatycznych. Przy wyborze technologii kierowaliśmy się ich popularnością, dostępnością dokumentacji oraz artykułów, a także łatwością użycia. Narzędzia zostały dopasowane do wybranych technologii i specyfikacji zadań. Poniżej przedstawiono opis wybranych opcji.

### **3.3.1 Technologie**

### **3.3.2 Narzędzia**

Do niektórych płatnych narzędzi mieliśmy bezpłatny dostęp za pośrednictwem uczelni, w innych mogliśmy założyć konta edukacyjne, które oferowały dostęp do wszystkich funkcji narzędzia. Gdy żadna z wymienionych opcji nie była udostępniona, wybieraliśmy rozwiązania darmowe.

- **IntelliJ IDEA Ultimate**

Jest to [IDE](#) od firmy JetBrains. Dzięki licznie dostępnym pluginom oferuje obsługę wielu języków programowania oraz innych składni. Pozwala również na integrację z repozytorium. Używaliśmy go do programowania zarówno [frontendu](#), jak i [backendu](#) oraz tworzenia dokumentacji w LaTeX.

- **Docker Desktop**

To narzędzie do zarządzania obrazami, kontenerami oraz wolumenami Docker. Zawiera w sobie również silnik tej technologii. Wykorzystywaliśmy je do lokalnego uruchamiania bazy danych oraz serwisu do cachowania.

- **One Drive**

Usługa dysku chmurowego oferowana przez firmę Microsoft. Przechowywaliśmy tam dokumenty oraz obrazy diagramów.

- **Azure Blob Storage**

To rozwiązanie chmurowe Microsoft, służące do bezpiecznego przechowywania dużej ilości danych nieustrukturyzowanych, takich jak pliki multimedialne, dokumenty czy kopie zapasowe. Dane są dostępne poprzez interfejs [REST API](#) usługi Azure Storage. Wykorzystaliśmy go do przechowywania zdjęć profilowych użytkownika oraz multimedii (zdjęcia i filmy) ze spotów i forum.

- **Jira**

To narzędzie firmy Atlassian do zarządzania pracami nad projektem w metodykach zwinnych. Do [Backlogu](#) wpisywaliśmy zadania, a na [tablicy Kanbanowej](#) rejestrowaliśmy ich statusy oraz poświęcony czas.

- **GitHub**

Zdalne repozytorium służące do przechowywania i wersjonowania kodu aplikacji. Zamieściliśmy tam kod naszego projektu. Do każdego zadania tworzyliśmy osobną gałąź z właściwą nazwą, a po zakończeniu prac przeprowadzaliśmy [review kodu](#). Następnie łączyliśmy ją do głównej gałęzi deweloperskiej.

- **GitHub Actions**

To narzędzie do implementacji procesów [CI/CD](#) na platformie GitHub, które umożliwiają automatyczne testowanie lub wdrażanie kodu. Uruchamiają się w reakcji na różne operacje w repozytorium, na przykład przesłanie zmian na

wybraną gałąź. Stosowaliśmy je do automatycznego testowania i budowania projektu po każdorazowym wprowadzeniu zmian.

- **GitHub Copilot**

To narzędzie sztucznej inteligencji będące asystentem programisty. W projekcie analizuje plik oraz pliki powiązane. Wykorzystywaliśmy go podczas [review codu](#). Copilot skanował wszystkie pliki i w komentarzach opisywał sugerowane zmiany lub potencjalne błędy.

- **Discord**

Darmowa platforma komunikacyjna. Umożliwia udostępnienie obrazu z ekranu, komunikację głosową oraz tekstową, jak i również przesyłanie plików. Stosowaliśmy go do spotkań, na których omawialiśmy sprawy dotyczące projektu.

- **Messenger**

Komunikator będący usługą Facebooka. Daje możliwość tworzenia czatów grupowych lub prywatnych, a także udostępniania plików. Używaliśmy go do ustalania spotkań na Discordzie oraz szybkiej komunikacji.

- **Postman**

To narzędzie służące do testowania endpointów [API](#). Pozwala grupować zapytania w kolekcje, wysyłać ich różne typy oraz analizować odpowiedzi z serwera. Wykorzystywaliśmy go do testowania stworzonych endpointów oraz debugowania.

- **Figma**

Narzędzie chmurowe do projektowania interfejsów użytkownika ([UI](#)). Umożliwia zespołowe tworzenie w pełni interaktywnych prototypów. Wykonaliśmy w nim projekty ekranów naszej aplikacji.

- **Visual Paradigm**

To narzędzie do tworzenia różnych diagramów stosowanych w inżynierii oprogramowania, takich jak [UML](#)( [3]) czy [BPMN](#)( [4]). Zrobiliśmy w nim diagram przypadków użycia.

## 3.4 Zasoby i ograniczenia

### 3.4.1 Zasoby

- **Specjalizacja członków zespołu** — wszyscy członkowie zespołu projektowego specjalizują się w aplikacjach internetowych.
- **Dostęp do przedstawiciela grupy docelowej** — jeden z członków zespołu (Adam) jest [droniarzem foto/video](#).
- **Status studenta** — fakt bycia studentem zapewnia dostęp do wersji premium wielu usług (Figma Education, GitHub PRO).
- **Oprogramowanie zapewniane przez PJATK** - uczelnia zapewnia dostęp do pakietu JetBrains oraz usług firmy Microsoft (OneDrive).

### 3.4.2 Ograniczenia

- **Ograniczenia czasowe** — projekt jest ograniczony harmonogramem akademickim i terminem oddania pracy dyplomowej, co wymagało wysokiego tempa realizacji oraz sprawnej komunikacji w zespole.
- **Ograniczenia budżetowe** — projekt nie posiada finansowania i w związku z tym korzystano z rozwiązań darmowych oraz open source.

### 3.4.3 Usługi zewnętrzne

<b>Usługa</b>	GitHub Actions (CI) [5]
<b>Opis</b>	Uruchomienia pipeline'ów CI/CD dla repozytorium GitHub.

<b>Limit</b>	3000 min/mies.
--------------	----------------

**Tabela 3.1:** Usługa zewnętrzna: GitHub Actions (CI)

<b>Usługa</b>	Azure Blob Storage [6]
<b>Opis</b>	Magazyn plików (m.in. zdjęcia spotów, załączniki z czatu).
<b>Limit</b>	1 GB/mies.

**Tabela 3.2:** Usługa zewnętrzna: Azure Blob Storage

<b>Usługa</b>	Mailtrap [7]
<b>Opis</b>	Środowisko testowe SMTP oraz Email API do wysyłki maili.
<b>Limit</b>	150 maili/dzień

**Tabela 3.3:** Usługa zewnętrzna: Mailtrap

<b>Usługa</b>	LocationIQ [8]
<b>Opis</b>	Geokodowanie adresu przy dodawaniu nowych spotów.
<b>Limit</b>	5 000 zapytań/dzień

**Tabela 3.4:** Usługa zewnętrzna: LocationIQ

<b>Usługa</b>	Google Maps (Maps URLs) [9]
---------------	-----------------------------

<b>Opis</b>	Otwieranie nawigacji w aplikacji Map Google (deep link/URL).
<b>Limit</b>	Brak limitu w ramach dokumentowanego sposobu użycia.

**Tabela 3.5:** Usługa zewnętrzna: Google Maps (Maps URLs)

<b>Usługa</b>	OpenFreeMap [10, 11]
<b>Opis</b>	Publiczny serwer kafelków do renderu mapy na froncie.
<b>Limit</b>	30 000 zapytań/mies.

**Tabela 3.6:** Usługa zewnętrzna: OpenFreeMap

<b>Usługa</b>	Open-Meteo [12]
<b>Opis</b>	Prognozy pogody wyświetlane dla spotów.
<b>Limit</b>	10 000 zapytań/dzień

**Tabela 3.7:** Usługa zewnętrzna: Open-Meteo

<b>Usługa</b>	Tenor GIF API [13]
<b>Opis</b>	Wyszukiwanie GIF-ów w czacie.
<b>Limit</b>	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

**Tabela 3.8:** Usługa zewnętrzna: Tenor GIF API

<b>Usługa</b>	Where the ISS at? [14]
---------------	------------------------

<b>Opis</b>	HTTP API z bieżącą pozycją satelity, używane pomocniczo.
<b>Limit</b>	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

**Tabela 3.9:** Usługa zewnętrzna: Where the ISS at?

## 3.5 Analiza ryzyka

# Rozdział 4

## Analiza wymagań

Niniejszy rozdział zawiera analizę wymagań postawionych systemowi.

Do określenia priorytetów realizacji wymagań skorzystano z metody MoSCoW. Metoda MoSCoW jest techniką priorytetyzacji wymagań. Polega ona na przypisaniu każdemu wymaganiu jednej z czterech kategorii priorytetu, określających jego znaczenie dla minimalnie użytecznej wersji systemu.

W niniejszej pracy przyjęto następującą interpretację priorytetów MoSCoW:

**M – *Must have*** wymagania krytyczne dla systemu. Muszą zostać zrealizowane w bieżącej wersji, aby system mógł zostać uznany za ukończony i spełniał podstawowe cele biznesowe.

**S – *Should have*** wymagania bardzo ważne. Powinny zostać zrealizowane, jeśli pozwolą na to dostępne zasoby (czas, zespół), jednak w sytuacji konieczności ograniczenia zakresu mogą zostać przesunięte do kolejnego wydania.

**C – *Could have*** wymagania opcjonalne, „mile widziane”. Zwiększają wygodę, kompletność lub atrakcyjność systemu, ale ich brak nie uniemożliwia osiągnięcia głównych celów projektu.

**W – *Won't have this time*** wymagania świadomie odłożone. Zostały zidentyfikowane, jednak nie będą realizowane w obecnym zakresie projektu (bieżącej wersji systemu); mogą stanowić bazę dla przyszłego rozwoju rozwiązania.

W dalszej części rozdziału każdy opis wymagania zawiera przypisany priorytet MoSCoW zgodnie z powyższą klasyfikacją.

Ponadto wymagania mogą mieć jeden z dwóch statusów realizacji:

**Zrealizowano** – zadanie zostało zrealizowane.

**Anulowano** – zadanie zostało anulowane.

## 4.1 Przypadki użycia

### 4.1.1 Aktorzy

**Użytkownik niezalogowany** Gość przeglądający publiczne treści (mapa, spoty, forum): może się zarejestrować lub zalogować.

**Użytkownik (nie premium)** Zarejestrowany użytkownik: zarządza kontem i ulubionymi spotami, dodaje treści/komentarze, korzysta z czatu.

**Użytkownik premium** Użytkownik z wykupioną subskrypcją: ma dostęp do funkcji premium (np. oznaczenie stref [PANSA](#) na mapie, rozbudowana prognoza pogody).

**Moderator** Moderacja treści: posty na forum, komentarze spotów.

**Deweloper** Rozwija i utrzymuje system.

#### Aktorzy będący zewnętrznymi usługami

Poniżej wymieniono aktorów, których opisy zamieszczono w rozdziale poświęconym integracji z zewnętrznymi usługami [3.4.3](#).

- Usługa mailowa (Mailtrap)
- Dostawca API do map (OpenFreeMap)
- Nawigacja (Google Maps)
- Dostawca API pogodowego (Open-Meteo)
- Dostawca API GIFów (Tenor)
- Dostawca API do określania strefy czasowej spota (“Where the ISS at?”)

- Dostawca API do geolokalizacji (LocationIQ)
- Azure Blob Storage
- Dostawca OAuth (Google)
- Dostawca OAuth (GitHub)

### 4.1.2 Diagram przypadków użycia

### 4.1.3 Scenariusz przypadków użycia

## 4.2 Wymagania

W niniejszym rozdziale przedstawiono wymagania stawiane projektowanemu systemowi. Celem rozdziału jest zebranie w jednym miejscu oczekiwań interesariuszy oraz usystematyzowanie ich w postaci jasno zdefiniowanych kategorii wymagań.

W pracy wyróżniono następujące typy wymagań:

- **Wymagania ogólne** – opisują system na wysokim poziomie, z perspektywy celu biznesowego i użytkownika, bez wchodzenia w szczegóły techniczne. Określają, jakie główne problemy ma rozwiązywać system i jakie korzyści ma dostarczać interesariuszom.
- **Wymagania dziedzinowe** – wynikają ze specyfiki obszaru, w którym wykorzystywany jest system (domena problemu). Odzwierciedlają reguły biznesowe, ograniczenia prawne oraz ustaloną praktykę w danej dziedzinie i muszą być spełnione niezależnie od przyjętych rozwiązań technicznych.
- **Wymagania funkcjonalne** – opisują konkretne funkcje systemu widziane z perspektywy użytkownika lub innego aktora. Definiują, jakie operacje system ma umożliwiać, jakie dane przetwarzać oraz jak reagować na określone zdarzenia i scenariusze użycia.
- **Wymagania pozafunkcjonalne** – określają, *jak* system ma realizować swoje funkcje, a nie *co* ma robić. Obejmują m.in. wymagania dotyczące

wydajności, bezpieczeństwa, niezawodności, użyteczności, skalowalności oraz utrzymywalności rozwiązania.

- **Wymagania dotyczące interfejsu z otoczeniem** – opisują sposób komunikacji systemu z innymi systemami, urządzeniami lub usługami zewnętrznymi. Określają format wymienianych danych, wykorzystywane protokoły, kierunek i częstotliwość wymiany informacji oraz wymagania dotyczące integracji.
- **Wymagania dotyczące środowiska docelowego** – definiują warunki techniczne, w jakich system ma być uruchamiany i eksploatowany. Dotyczą m.in. platformy sprzętowej i programowej, systemów operacyjnych, zasobów sieciowych oraz innych elementów infrastruktury niezbędnych do poprawnego działania aplikacji. (TODO do usunięcia?)

W dalszej części rozdziału wymagania zostały logicznie pogrupowane według obszarów funkcjonalnych systemu: przedstawiono wymagania ogólne, a następnie wymagania dotyczące czatu, forum, mapy oraz panelu użytkownika.

## 4.2.1 Wymagania ogólne

### 4.2.1.1 Wymagania ogólne dla czatu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
<b>Identyfikator:</b>	WOCZAT-01	<b>Priorytet:</b>	M
<b>Nazwa:</b>	Wysyłanie wiadomości na czacie		
<b>Opis:</b>	System umożliwia użytkownikowi wysyłanie wiadomości w ramach wybranego czatu, tak aby uczestnicy mogli przekazywać sobie informacje w kontekście platformy Merkury.		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)	
<b>Kryteria akceptacji:</b>	Wymaganie uznaje się za spełnione, jeżeli zalogowany użytkownik może wybrać czat, wprowadzić treść wiadomości i zobaczyć ją jako kolejną pozycję w rozmowie, a pozostali uczestnicy czatu widzą tę wiadomość po swojej stronie.
<b>Dane wejściowe:</b>	Zalogowany użytkownik, identyfikator wybranego czatu, treść wiadomości.
<b>Warunki początkowe:</b>	Użytkownik jest poprawnie zalogowany i posiada dostęp do danego czatu.
<b>Warunki końcowe:</b>	Nowo wysłana wiadomość staje się częścią historii danego czatu i jest dostępna dla wszystkich uprawnionych uczestników rozmowy.
<b>Sytuacje wyjątkowe:</b>	Brak dostępu do czatu, problemy z połączeniem sieciowym, niepoprawne lub puste dane wejściowe.
<b>Szczegóły implementacji:</b>	Ogólna obsługa wysyłania wiadomości w module czatu; szczegółowe rozwiązania (np. obsługa załączników, rodzaje wiadomości) zostaną doprecyzowane w wymaganiach szczegółowych.
<b>Udziałowiec:</b>	UO3
<b>Realizator:</b>	Adam Langmesser
<b>Status:</b>	Zrealizowano
<b>Notatka:</b>	— brak

**Tabela 4.1:** Karta wymagania funkcjonalnego dla czatu: Wysyłanie wiadomości na czacie

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
<b>Identyfikator:</b>	WOCZAT-02	<b>Priorytet:</b>	M
<b>Nazwa:</b>	Edycja czatu		

<b>KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)</b>	
<b>Opis:</b>	System umożliwia użytkownikowi z odpowiednimi uprawnieniami wprowadzanie podstawowych zmian w konfiguracji czatu, takich jak nazwa czatu czy skład uczestników.
<b>Kryteria akceptacji:</b>	Wymaganie uznaje się za spełnione, jeżeli uprawniony użytkownik może zmienić kluczowe parametry czatu, a zaktualizowane informacje są widoczne dla pozostałych uczestników podczas korzystania z modułu czatu.
<b>Dane wejściowe:</b>	Zalogowany użytkownik, identyfikator czatu, zestaw nowych wartości konfiguracyjnych (np. nowa nazwa, lista uczestników).
<b>Warunki początkowe:</b>	Użytkownik jest zalogowany i posiada uprawnienia do zarządzania danym czatem.
<b>Warunki końcowe:</b>	Parametry czatu są zapisane w systemie w nowej postaci, a użytkownicy korzystają z odświeżonych informacji podczas dalszej komunikacji.
<b>Sytuacje wyjątkowe:</b>	Próba edycji przez użytkownika bez wymaganych uprawnień, niepoprawne dane wejściowe, konflikt zmian, problemy z zapisem danych.
<b>Szczegóły implementacji:</b>	Ogólna obsługa edycji ustawień czatu w module czatu; konkretne operacje (np. edycja nazwy, avatara, listy uczestników) będą doprecyzowane w wymaganiach szczegółowych.
<b>Udziałowiec:</b>	UO3
<b>Realizator:</b>	Adam Langmesser
<b>Status:</b>	Zrealizowano
<b>Notatka:</b>	— brak

**Tabela 4.2:** Karta wymagania funkcjonalnego dla czatu: Edycja czatu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
<b>Identyfikator:</b>	WOCZAT-03	<b>Priorytet:</b>	M
<b>Nazwa:</b>	Przeglądanie historii czatu		
<b>Opis:</b>	System udostępnia użytkownikowi możliwość przeglądania wcześniejszych wiadomości na czacie, tak aby mógł wracać do poprzednich rozmów i ustaleń.		
<b>Kryteria akceptacji:</b>	Wymaganie uznaje się za spełnione, jeżeli użytkownik po otwarciu czatu widzi fragment historii rozmowy i może w prosty sposób docierać do starszych wiadomości zgodnie z przyjętym sposobem przeglądania (np. przewijanie).		
<b>Dane wejściowe:</b>	Zalogowany użytkownik, identyfikator czatu, ewentualne ogólne parametry zakresu historii (np. kierunek przeglądania).		
<b>Warunki początkowe:</b>	Użytkownik jest zalogowany i posiada dostęp do danego czatu, a w systemie istnieją zapisane wiadomości dla tego czatu lub ich brak.		
<b>Warunki końcowe:</b>	Użytkownik ma dostęp do historii rozmowy w takim zakresie, w jakim przewidziano to dla danego typu konta i konfiguracji systemu.		
<b>Sytuacje wyjątkowe:</b>	Brak uprawnień do danego czatu, tymczasowy problem z odczytem danych, brak jakiegokolwiek historii dla wskazanego czatu.		
<b>Szczegóły implementacji:</b>	Ogólny mechanizm udostępniania historii konwersacji w module czatu; szczegóły dotyczące paginacji, filtrów i limitów zostaną określone w wymaganiach szczegółowych.		
<b>Udziałowiec:</b>	UO3		
<b>Realizator:</b>	Adam Langmesser		
<b>Status:</b>	Zrealizowano		
<b>Notatka:</b>	— brak		

**Tabela 4.3:** Karta wymagania funkcjonalnego dla czatu: Przeglądanie historii

czatu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
<b>Identyfikator:</b>	WOCZAT-04	<b>Priorytet:</b>	M
<b>Nazwa:</b>	Tworzenie czatu		
<b>Opis:</b>	System umożliwia użytkownikowi inicjowanie nowych rozmów poprzez tworzenie czatów prywatnych (1:1) lub grupowych z wybranymi uczestnikami.		
<b>Kryteria akceptacji:</b>	Wymaganie uznaje się za spełnione, jeżeli zalogowany użytkownik może zainicjować nowy czat, wskazać podstawowe informacje oraz uczestników, a nowy czat pojawia się na liście czatów i jest gotowy do użycia.		
<b>Dane wejściowe:</b>	Zalogowany użytkownik, typ tworzonego czatu (np. 1:1, grupowy), podstawowe informacje opisujące czat oraz lista uczestników.		
<b>Warunki początkowe:</b>	Użytkownik jest zalogowany i posiada uprawnienia do inicjowania nowych czatów w systemie.		
<b>Warunki końcowe:</b>	Nowy czat jest dodany do systemu i widoczny na listach czatów odpowiednich użytkowników, a dalsza komunikacja może odbywać się w jego ramach.		
<b>Sytuacje wyjątkowe:</b>	Próba utworzenia czatu z nieprawidłowymi danymi (np. nieistniejący użytkownik), przekroczenie ogólnych limitów systemowych, problem z zapisem nowego czatu.		
<b>Szczegóły implementacji:</b>	Ogólna obsługa zakładania nowych czatów w module czatu; szczegóły dotyczące rodzajów czatów, limitów i dodatkowych ustawień zostaną doprecyzowane w wymaganiach szczegółowych.		
<b>Udziałowiec:</b>	UO3		
<b>Realizator:</b>	Adam Langmesser		
<b>Status:</b>	Zrealizowano		
<b>Notatka:</b>	— brak		

---

**Tabela 4.4:** Karta wymagania funkcjonalnego dla czatu: Tworzenie czatu

# 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 Integracja z bazą danych

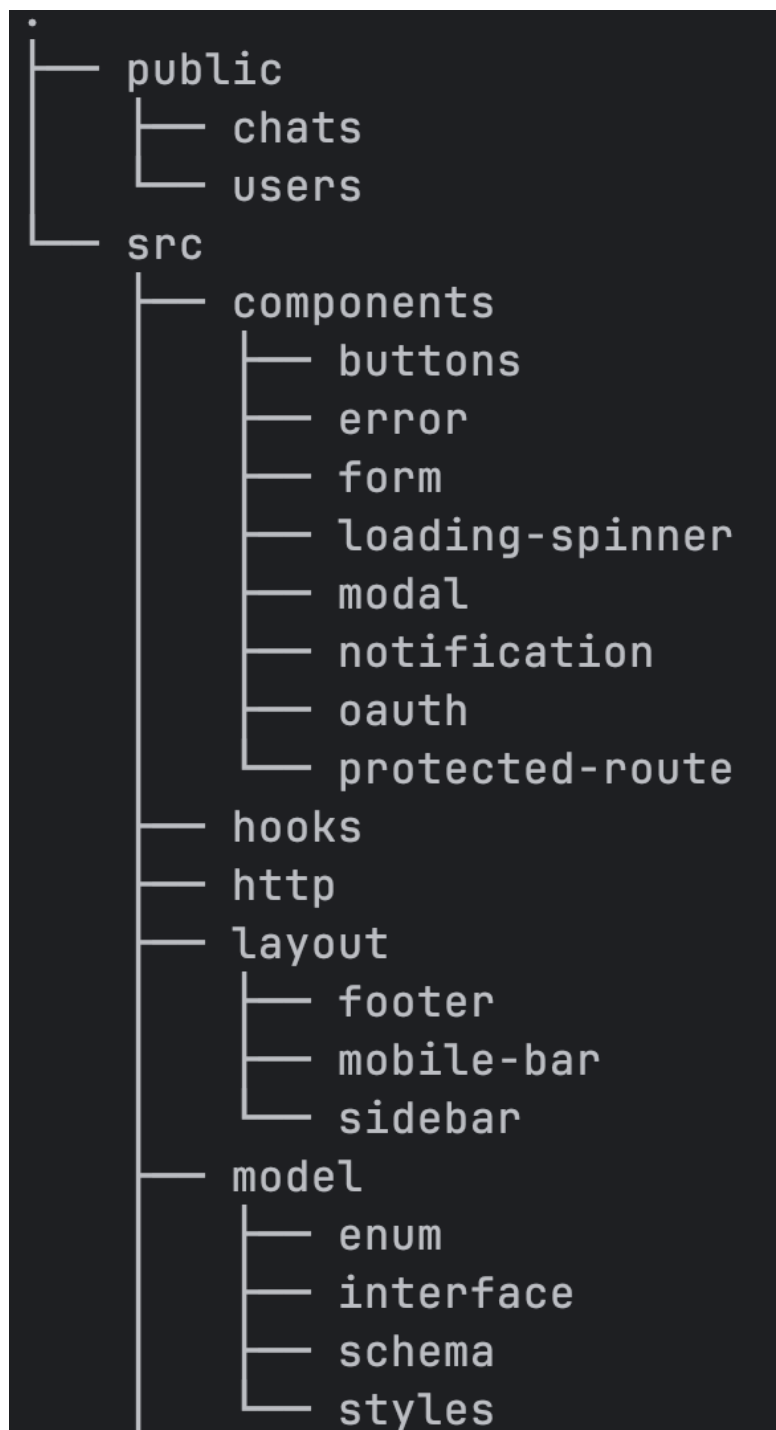
#### 7.1.3 Obsługa uwierzytelnienia

#### 7.1.4 Konteneryzacja

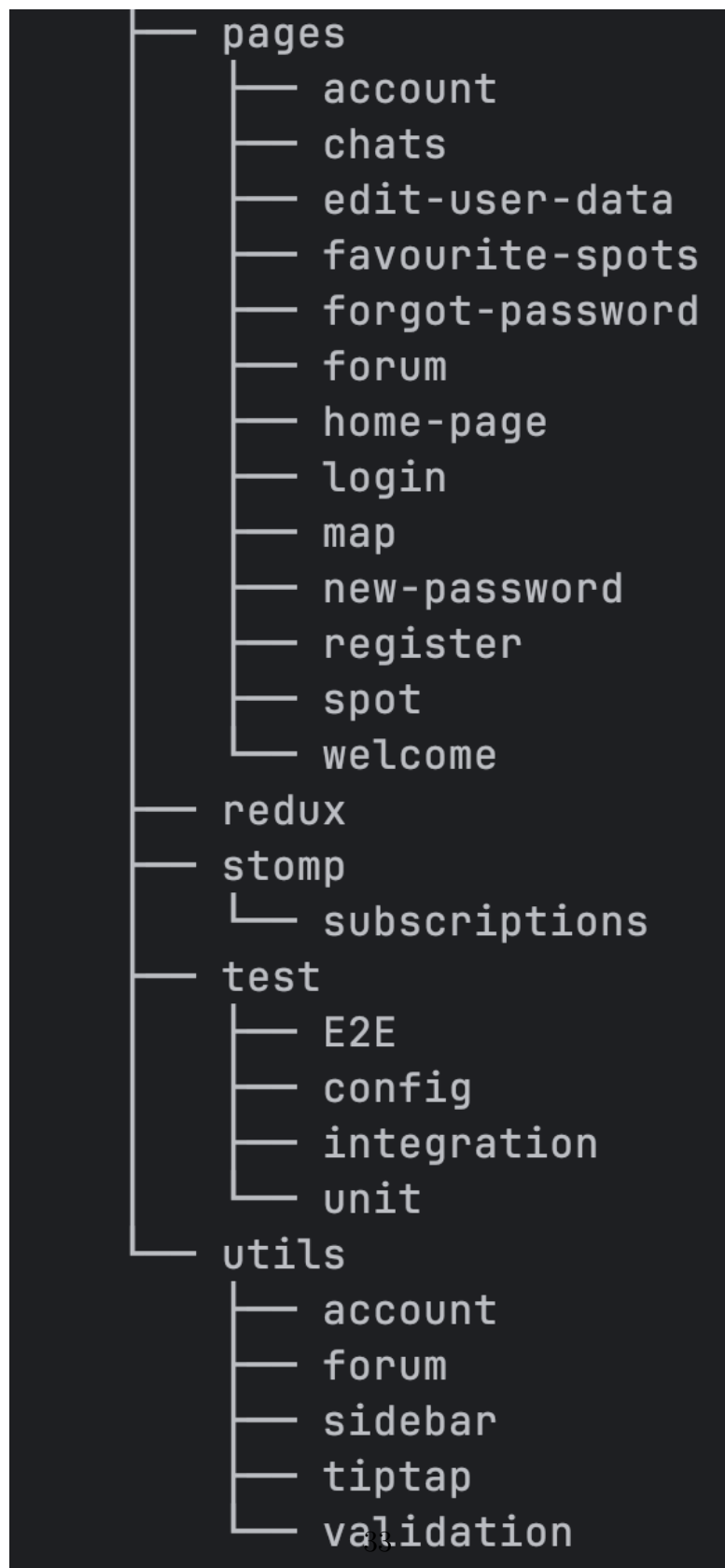
### 7.2 Implementacja frontendu

#### 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`)

### 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`) [15], jak i ze [Stanu](#) globalnego, utrzymywanego przez [Bibliotekę React Redux](#) [16]. 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 [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

wanie 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](#) [17] oraz [Biblioteki TanStack Query](#) [18]. 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 związane 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 setEmailWithNewPasswordLink(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 scentralizowane i mogą być w prosty sposób wykorzystywane w różnych częściach aplikacji. Zastosowaliśmy TanStack Query, ponieważ znacząco 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>) => Pr... = async (event: FormEvent<HTMLFormElement>) : Promise<void> => {
  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 [19]. 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.

	<code>--color-violetDark: #363041;</code>
	<code>--color-violetLight: #6d6183;</code>
	<code>--color-violetLightDarker: #4f4660;</code>
	<code>--color-violetLightDark: #554a69;</code>
	<code>--color-violetLighter: #9b8cbd;</code>
	<code>--color-violetDarker: #2c2734;</code>
	<code>--color-violetHeavyDark: #1e1b23;</code>
	<code>--color-violetBtnBorderDark: #625b6e;</code>
	<code>--color-violetBright: #835ace;</code>
	<code>--color-darbVioletBtnOutline: #816ba6;</code>
	<code>--color-mediumDarkBlue: #424b77;</code>
	<code>--color-first: #2c3e50;</code>
	<code>--color-second: #34495e;</code>
	<code>--color-third: #1abc9c;</code>
	<code>--color-fourth: #16a085;</code>
	<code>--color-fifth: #ecf0f1;</code>
	<code>--color-sixth: #e94560;</code>
	<code>--color-magenta: #a01bc1;</code>
	<code>--color-darkYellow: #c5a03c;</code>
	<code>--color-ratingStarColor: #fadb14;</code>
	<code>--color-locationMarkerDarkerBlue: #a3dcff;</code>
	<code>--color-locationMarkerLightBlue: #52bafb;</code>
	<code>--color-userLocationDot: #4285f4;</code>
	<code>--color-spotLocationMarker: #a8071a;</code>

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 znacząco 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](#) [20]. 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 Wyszukiwarka spotów

Niniejszy rozdział opisuje sposób implementacji wyszukiwarki spotów.

Jednym z głównych modułów aplikacji jest wyszukiwarka spotów, która umożliwia użytkownikowi szybkie odnalezienie interesujących lokalizacji. Funkcjonuje ona w dwóch wariantach: prostym i zaawansowanym (rys. 7.16 oraz 7.17).

```

<div className={`${dark:bg-darkBg} ${dark:text-darkText} ${bg-lightBg} ${text-lightText}
flex min-h-screen w-full flex-col items-center space-y-4 overflow-hidden p-8 pt-18">
  <Switch />
  <SearchBar
    onSetSpots={handleSetSearchedSpots}
    loadMoreRef={loadMoreRef}
    onSetFetchingNextPage={setIsFetchingNextPage}
  />
  <div className="flex w-full flex-col items-center space-y-4">
    <h1 className="text-center text-3xl">The Most Popular Spots</h1>
    <div className="flex w-full flex-col items-center space-y-5">
      <Carousel spots={data!} spotsPerPage={spotsPerPage} />
      <SearchSpotList
        spots={searchedSpots}
        isFetchingNextPage={isFetchingNextPage}
        loadMoreRef={loadMoreRef}
      />
    </div>
  </div>
</div>

```

Rysunek 7.16: Implementacja prostej wersji wyszukiwarki

```

<div className={`${dark:bg-darkBg} ${dark:text-darkText} ${bg-lightBg} ${text-lightText}
flex min-h-screen w-full flex-col items-center space-y-4 overflow-hidden p-8 pt-18">
  <Switch />
  <AdvanceSearchBar
    onSetSpots={handleSetSearchedSpots}
    loadMoreRef={loadMoreRef}
    onSetFetchingNextPage={setIsFetchingNextPage}
  />
  <div className="flex w-full flex-col items-center space-y-10">
    <SearchSpotList
      spots={searchedSpots}
      loadMoreRef={loadMoreRef}
      isFetchingNextPage={isFetchingNextPage}
    />
  </div>
</div>

```

Rysunek 7.17: Implementacja zaawansowanej wersji wyszukiwarki

Przełączanie pomiędzy tymi widokami odbywa się za pomocą przycisku umieszczonego w górnej części strony (rys. 7.18).

```
<div className="dark:shadow-darkBgSoft flex rounded-full shadow-lg shadow-black/20">
  <NavLink
    to="/"
    className={({ isActive } : NavLinkRenderProps ) : string =>
      `dark:shadow-darkBgSoft flex rounded-full shadow-lg shadow-black/20
        hover:dark:bg-violetDark hover:bg-violetLight rounded-l-full px-2.5 py-1.5
        transition-all duration-300 ${isActive ? "dark:bg-violetDark bg-violetLight" : ""}`
  >
    Simple filters
  </NavLink>
  <NavLink
    to="/advanced"
    className={({ isActive } : NavLinkRenderProps ) : string =>
      `dark:shadow-darkBgSoft flex rounded-full shadow-lg shadow-black/20
        hover:dark:bg-violetDark hover:bg-violetLight rounded-r-full px-2.5 py-1.5
        transition-all duration-300 ${isActive ? "dark:bg-violetDark bg-violetLight" : ""}`
  >
    Advanced filters
  </NavLink>
</div>
```

Rysunek 7.18: Implementacja komponentu do przełączania trybów

W trybie prostym prezentowana jest karuzela (rys. 7.19) z dwunastoma najpopularniejszymi **spotami** w całej aplikacji. Użytkownik może w tym miejscu wyszukiwać **spoty** po lokalizacji (kraj, region, miasto).

```

<div className="relative flex w-full items-center justify-center">
  <button
    onClick={() : void => paginate(-1)}
    className="hover:text-darkBorder z-10 cursor-pointer transition-all duration-300"
  >
    <RiArrowLeftWideFill className="text-5xl sm:text-6xl" />
  </button>

  <div className="relative h-[440px] w-full max-w-[1200px] overflow-hidden">
    <AnimatePresence custom={direction} initial={false} mode="sync">
      <motion.div
        key={page}
        custom={direction}
        variants={sliderVariants}
        initial="incoming"
        animate="active"
        exit="exit"
        transition={[ 3 elements... ]}
        className="grid w-full grid-cols-1 grid-rows-1 justify-items-center gap-4
          lg:grid-cols-2 lg:grid-rows-2 2xl:grid-cols-3 2xl:grid-rows-2"
      >
        {currentSpots.map((spot : TopRatedSpot ) : Element => (
          <MostPopularSpot
            spot={spot}
            key={` ${spot.id}-${page}`}
          />
        ))}
      </motion.div>
    </AnimatePresence>
  </div>

  <button
    onClick={() : void => paginate(1)}
    className="hover:text-darkBorder z-10 cursor-pointer transition-all duration-300"
  >
    <RiArrowRightWideFill className="text-5xl sm:text-6xl" />
  </button>
</div>

```

Rysunek 7.19: Implementacja karuzeli z najpopularniejszymi [spotami](#)

Widok zaawansowany udostępnia rozszerzoną wyszukiwarę, która umożliwia filtrowanie wyników po mieście, tagach oraz ocenie, a także ich sortowanie według popularności i średniej oceny (rys. 7.17).

Wyszukiwarka spotów została zbudowana z dwóch głównych komponentów: `HomePage` oraz `AdvanceHomePage`. W skład prostej wersji wchodzi następujące komponenty:

- `Switch` – służy do przełączania widoku między trybem podstawowym a zaawansowanym,
- `SearchBar` – podstawowa wyszukiwarka [spotów](#),
- `Carousel` – wyświetla najpopularniejsze [spoty](#),
- `SearchSpotList` – wyświetla znalezione [spoty](#).

W skład zaawansowanej wersji wchodzi następujące komponenty:

- `Switch` – służy do przełączania widoku między trybem podstawowym a zaawansowanym,
- `AdvanceSearchBar` – zaawansowana wyszukiwarka [spotów](#),
- `SearchSpotList` – wyświetla znalezione [spoty](#).

Komponent `Switch` (rys. 7.18) zawiera dwa elementy `NavLink` z biblioteki `React Router`, co pozwala na przełączanie widoków bez konieczności przeładowywania całej strony.

W komponencie `SearchBar` (rys. 7.20) po wpisaniu co najmniej dwóch znaków wyświetlana jest lista podpowiedzi dla kraju, regionu oraz miasta, w zależności od aktualnie uzupełnianego pola. Po pojawieniu się listy użytkownik może wybrać interesującą go lokalizację, co ułatwia określenie, w jakich miejscach znajdują się dostępne [spoty](#).

```

<div className="dark:bg-darkBgSoft light:bg-lightBgSoft flex w-full flex-col items-center justify-between
space-y-3 rounded-md px-3 py-2 shadow-md md:flex-row md:space-y-0 lg:w-3/4 lg:space-x-3 xl:w-1/2
dark:shadow-black">
  <div className="flex w-full flex-col space-y-2">
    <h1>Location</h1>
    <div className="flex w-full flex-col space-y-3 md:flex-row md:space-y-0 md:space-x-2">
      {inputList.map(({ id, label } : { readonly label: "Your Country"; readonl... } : Element ) => (
        <div key={id} className="relative w-full">
          <SearchInput
            label={label}
            id={id}
            value={searchLocation[id] ?? ""}
            onChange={(e : ChangeEvent<HTMLInputElement> ) : void =>
              handleSetLocation(id, e.target.value)
            }
            onFocus={() : void => setActiveInput(id)}
          />
          {activeInput === id && suggestions.length > 0 && (
            <SearchSuggestions
              suggestions={suggestions}
              onClick={handleSuggestionClick}
              id={id}
              onClose={() : void => setActiveInput(null)}
            />
          )}
        </div>
      )}
    </div>
  </div>
  <button
    className="dark:bg-darkBgMuted dark:hover:bg-darkBgMuted/80 light:bg-lightBgMuted
    light:hover:bg-lightBgMuted/80 flex w-full cursor-pointer justify-center rounded-md p-2 md:w-fit"
    onClick={handleSearchSpots}
  >
    <FaSearch />
  </button>
</div>

```

Rysunek 7.20: Implementacja prostej wyszukiwarki

Komponent `SearchSpotList` (rys. 7.21) odpowiada za prezentację wyników wyszukiwania. Został w nim zaimplementowany mechanizm przewijania nieskończonego (*infinite scroll*), który automatycznie pobiera kolejne strony wyników w momencie, gdy użytkownik zbliża się do końca listy. Wykorzystuje on listę komponentów `SpotTile`, a także komponent `LoadingSpinner` oraz komunikat informujący o braku wyników, jeżeli nie zostanie odnaleziony żaden *spot*.

```

<>
<ul className="grid w-full grid-cols-1 place-items-center gap-8 xl:grid-cols-2 2xl:grid-cols-3">
  {spots.map((spot : HomePageSpotDto ) : Element => (
    <SpotTile key={spot.id} spot={spot} />
  ))}
</ul>
<div ref={loadMoreRef} className="h-10" />
{isFetchingNextPage && <LoadingSpinner />}
{spots.length === 0 && (
  <p className="text-center text-2xl">
    Search for spots to see results.
  </p>
)}
</>

```

Rysunek 7.21: Implementacja listy do wyświetlania [spotów](#)

Komponent `SpotTile` zawiera następujące informacje:

- zdjęcie [spota](#),
- miasto, w którym się znajduje,
- nazwę [spota](#),
- ocenę oraz liczbę ocen,
- tagi,
- podstawowe informacje pogodowe (temperatura i typ pogody),
- dwa przyciski: jeden prowadzący do widoku szczegółów [spota](#) oraz drugi informujący, jak daleko znajduje się dany [spot](#); po kliknięciu przycisku lokalizacja [spota](#) jest prezentowana na mapie.

Komponent `AdvanceSearchBar` jest zbliżony wyglądem i strukturą do wersji podstawowej, jednak w polu lokalizacji można podać wyłącznie miasto. Dodatkowo dostępna jest możliwość dodawania tagów z przygotowanej listy. Wyszukiwarka umożliwia także filtrowanie po ocenie oraz sortowanie wyników według oceny i popularności z wykorzystaniem komponentów typu `Dropdown`.

Oba widoki (HomePage i AdvanceHomePage) współdzielą część komponentów, między innymi Switch oraz SearchSpotList. Dzięki temu kod odpowiedzialny za wyświetlanie listy wyników jest zdefiniowany w jednym miejscu, a zmiany w sposobie prezentacji [spotów](#) wymagają modyfikacji tylko w komponentach współdzielonych.

#### **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, 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 [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

# Rozdział 12

## Słownik pojęć i skrótów

### API

(ang. *application programming interface*); zbiór reguł i operacji do komunikacji z oprogramowaniem.. [15](#)

### 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](#), [13](#), [56](#)

### Backlog

Lista zadań, które należy wykonać w ramach projektu, używane w metodykach zwinnych.. [14](#)

### Biblioteka

Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. [34](#), [36](#), [38](#), [40](#), [43](#), [44](#), [49](#)

### BPMN

(ang. *Business Process Model and Notation*); standardowa notacja graficzna, która umożliwia szczegółowe przedstawienie i dokumentowanie procesów biznesowych.. [16](#)

## CI/CD

Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. [14](#), [55](#), [58](#)

## 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.. [43](#), [44](#)

## Design

Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). [55](#)

## Disciplined Agile Delivery - Lean Life Cycle

Disciplined Agile Delivery w wariancie Lean Life Cycle to sposób prowadzenia projektu, który łączy elastyczność Agile z przewidywalnością Waterfalla, ale bez stałych sprintów — praca toczy się w ciągłym przepływie. Na starcie zakłada mocniejszą fazę przygotowawczą: doprecyzowanie zakresu, szkic architektury, identyfikację ryzyk i kryteria jakości. W realizacji następuje ciągle doprecyzowywanie wymagań i backlogu, oparte na regularnym feedbacku udziałowców. Całość opiera się na praktykach Lean oraz lekkim governance: code review i regularnych przeglądach postępów. . [9](#)

## Droniarz

Potoczne określenie osoby, która jest jednocześnie pilotem oraz operatorem drona. Zwykle entuzjasta dronów.. [8](#)

## Droniarz foto/video

Pilot wykorzystujący drony fotograficzne/filmowe do rejestracji materiałów wizualnych (zdjęcia, wideo), zwykle z naciskiem na stabilizację i jakość obrazu.. [16](#)

## 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. [31](#)

## Framework

Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. [2](#), [41](#)

## Frontend

Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. [2](#), [13](#), [56](#)

## 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`).. [36](#), [40](#)

## IDE

(ang. *integrated development environment*); to zintegrowane środowisko programistyczne, służące do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. [13](#)

## Infinite scroll

Wzorzec interfejsu użytkownika, w którym kolejne porcje treści są automatycznie doładowywane podczas przewijania strony w dół, zamiast być podzielone na odrębne, ręcznie przełączane strony. [50](#)

## JWT

Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. [38](#), [55](#)

## 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*).. [64](#)

## PANSA

Polish Air Navigation Services Agency, pol. Polska Agencja Żeglugi Powietrznej. Instytucja ta zapewnia m.in. mapę z zaznaczonymi strefami lotów. Każda strefa ma swoje właściwości prawne. . [21](#)

## Props

Właściwości przekazywane do komponentu React przez komponent nadrzędny; służą do konfiguracji i przekazywania danych. Powinny być traktowane jako tylko do odczytu (read-only) wewnątrz komponentu potomnego.. [36](#)

## 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 (np. na stronę główną). [35](#)

## React

Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. [36](#)

## 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.. [36](#), [37](#)

## 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.. [44](#)

## REST API

Architektura budowania usług sieciowych komunikujących się poprzez metody protokołu HTTP (GET, PUT, POST, DELETE, PATCH). Wymiana danych występuje często w formacie JSON lub XML.

REST API musi spełniać następujące reguły:

1. **Rozdzielenie klient-serwer** — klient i serwer są od siebie niezależne, komunikują się poprzez interfejs.
2. **Bezstanowość** — każde żądanie przez klienta zawiera wszystkie informacje niezbędne do jego obsłużenia. Po otrzymaniu żądania serwer nie przechowuje o nim żadnych informacji.
3. **Buforowalność (cache)** — odpowiedzi z API powinny informować, czy dane można cache'ować. Jeśli tak, to przy kolejnym żądaniu mogą być zwrócone z cache'a.
4. **Jednolity interfejs:**
  - **Identyfikacja zasobów** — każdy zasób musi być jednoznacznie zidentyfikowany w interakcji klient-serwer.
  - **Manipulacja zasobów poprzez reprezentację** — po otrzymaniu reprezentacji klient może zmienić stan zasobu przesyłając zmodyfikowaną reprezentację.

- **Samoopisujące się wiadomości** — każde żądanie i odpowiedź powinny zawierać informacje do jego poprawnego przetworzenia.
  - **Hypermedia jako silnik stanu aplikacji (HATEOAS)** — po otrzymaniu odpowiedzi klient powinien móc dynamicznie poznać inne interakcje przez linki.
5. **Warstwowość** — klient nie wie czy komunikuje się bezpośrednio z serwerem, czy poprzez pośrednika (np. proxy) oraz nie wie z czym komunikuje się obsługująca go warstwa.
  6. **Kod na żądanie (opcjonalnie)** — serwer może przesłać fragment kodu, który zostanie wykonany przez klienta.

[14](#)

## Review kodu

Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. [14](#), [15](#), [55](#)

## Sidebar

Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. [55–57](#)

## Spot

Spotkanie zespołu projektowego, zazwyczaj krótkie i regularne, służące omówieniu postępów prac, problemów oraz planów na najbliższy okres. [47–52](#)

## 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).. [36](#)

## Tablica Kanban

Narzędzie do zarządzania przepływem pracy, które pomaga zespołom śledzić zadania oraz ich postępy. Składa się z kolumn reprezentujących stan etapu prac, na przykład „Do zrobienia” lub „W trakcie”.. [14](#)

## TypeScript

Rozszerzenie do języka JavaScript dodający statyczne typowanie, interfejsy i narzędzia do większych projektów. Kompiluje się do czystego JavaScript, ułatwiając wykrywanie błędów w czasie kompilacji i refaktoryzację.. [36](#)

## UI

Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. [15](#), [36](#)

## UML

(ang. *Unified Modeling Language*); graficzny język wizualizacji, specyfikowania oraz dokumentowania składników systemów informatycznych. . [16](#)

# Spis tabel

2.1	Zespół projektowy . . . . .	7
2.2	Promotor . . . . .	8
2.3	Droniarze . . . . .	8
Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI) . . . . .		17
Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage . . . . .		17
Tabela 3.3: Usługa zewnętrzna: Mailtrap . . . . .		17
Tabela 3.4: Usługa zewnętrzna: LocationIQ . . . . .		17
Tabela 3.5: Usługa zewnętrzna: Google Maps (Maps URLs) . . . . .		18
Tabela 3.6: Usługa zewnętrzna: OpenFreeMap . . . . .		18
Tabela 3.7: Usługa zewnętrzna: Open-Meteo . . . . .		18
Tabela 3.8: Usługa zewnętrzna: Tenor GIF API . . . . .		18
Tabela 3.9: Usługa zewnętrzna: Where the ISS at? . . . . .		19
Tabela 4.1: Karta wymagania funkcjonalnego dla czatu: Wysyłanie wiadomości na czacie . . . . .		24
Tabela 4.2: Karta wymagania funkcjonalnego dla czatu: Edycja czatu . . . . .		25
Tabela 4.3: Karta wymagania funkcjonalnego dla czatu: Przeglądanie historii czatu . . . . .		27
Tabela 4.4: Karta wymagania funkcjonalnego dla czatu: Tworzenie czatu . . . . .		28

# Bibliografia

- [1] *Disciplined Agile Delivery*. PMI. 1 stycznia 2025. URL: <https://www.pmi.org/disciplined-agile/process/introduction-to-dad/why> (dostęp 30.10.2025).
- [2] *Disciplined Agile Delivery — Lean Life Cycle*. PMI. 1 stycznia 2025. URL: <https://www.pmi.org/disciplined-agile/lifecycle/lean-lifecycle> (dostęp 30.10.2025).
- [3] Stanisław Wrycza, Bartosz Marcinkowski i Krzysztof Wyrzykowski. „Język UML 2.0 w modelowaniu systemów informatycznych”. Warszawa: Helion, 2006. ISBN: 83-736-1892-9, 8373618929.
- [4] Michał Wolski. *10 wskazówek poprawiających modelowanie procesów biznesowych w notacji BPMN*. 14 maja 2024. URL: <https://wolski.pro/2024/05/10-wskazowek-poprawiajacych-modelowanie-procesow-biznesowych-w-notacji-bpmn/> (dostęp 19.11.2025).
- [5] *About billing for GitHub Actions*. GitHub Docs. 1 stycznia 2024. URL: <https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions> (dostęp 2.11.2025).
- [6] *Scalability and performance targets for Blob storage*. Microsoft Learn. 1 stycznia 2024. URL: <https://learn.microsoft.com/azure/storage/blobs/scalability-targets> (dostęp 2.11.2025).
- [7] *What are the limitations in Mailtrap?* Mailtrap Docs. 1 stycznia 2024. URL: <https://help.mailtrap.io/article/111-what-are-the-limitations-in-mailtrap/> (dostęp 2.11.2025).
- [8] *LocationIQ Pricing*. LocationIQ. 1 stycznia 2024. URL: <https://locationiq.com/pricing> (dostęp 2.11.2025).
- [9] *Google Maps (Maps URLs)*. Google Maps. 1 stycznia 2024. URL: <https://developers.google.com/maps/documentation/urls/get-started?hl=pl> (dostęp 2.11.2025).
- [10] *OpenFreeMap Documentation*. OpenFreeMap. 1 stycznia 2024. URL: <https://openfreemap.org/docs> (dostęp 2.11.2025).

- [11] *OpenFreeMap Quick Start*. OpenFreeMap. 1 stycznia 2024. URL: <https://openfreemap.org/docs/quick-start> (dostęp 2.11.2025).
- [12] *Open-Meteo API Usage & Pricing*. Open-Meteo. 1 stycznia 2024. URL: <https://open-meteo.com/en/docs/usage-and-pricing> (dostęp 2.11.2025).
- [13] *Tenor API — Documentation*. Tenor. 1 stycznia 2024. URL: <https://tenor.com/gifapi/documentation> (dostęp 2.11.2025).
- [14] *Where the ISS at? API*. wheretheiss.at. 1 stycznia 2024. URL: <https://wheretheiss.at/> (dostęp 2.11.2025).
- [15] *React useState*. 1 stycznia 2025. URL: <https://react.dev/reference/react/useState> (dostęp 3.11.2025).
- [16] *Redux*. 1 stycznia 2025. URL: <https://redux.js.org/> (dostęp 3.11.2025).
- [17] *Axios*. 1 stycznia 2025. URL: <https://axios-http.com/> (dostęp 3.11.2025).
- [18] *Tanstack Query*. 1 stycznia 2025. URL: <https://tanstack.com/query/latest> (dostęp 3.11.2025).
- [19] *Tailwind*. 1 stycznia 2025. URL: <https://tailwindcss.com/> (dostęp 3.11.2025).
- [20] *Motion*. 1 stycznia 2025. URL: <https://motion.dev/> (dostęp 3.11.2025).

# 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.