



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

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

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

Data ukończenia projektu: 23 listopada 2025	Recenzent: dr Elżbieta Puźniakowska-Gałuch
---	--

Spis treści

1	Wstęp	6
1.1	O projekcie	6
1.2	Cel i zakres prac	6
1.3	Geneza pomysłu	6
2	Opis problemu	7
2.1	Rich picture	7
2.2	Udziałowcy	7
2.3	Istniejące rozwiązania	9
2.4	Wizja rozwiązania	9
2.5	Aspekty społeczne i biznesowe	9
2.5.1	Aspekty społeczne	9
2.5.2	Aspekty biznesowe	9
3	Planowanie	10
3.1	Metodologia pracy	10
3.1.1	Przegląd rozważanych podejść	10
3.1.2	Odrzucone podejścia	10
3.1.3	Wybrane podejście: Disciplined Agile Delivery (Lean Life Cycle)	11
3.1.4	Narzędzia i komunikacja	11
3.1.5	Podział ról w zespole	12
3.2	Harmonogram projektu	12
3.3	Technologie i narzędzia	14
3.3.1	Technologie	14

3.3.2	Narzędzia	14
3.4	Zasoby i ograniczenia	17
3.4.1	Zasoby	17
3.4.2	Ograniczenia	17
3.4.3	Usługi zewnętrzne	17
3.5	Analiza ryzyka	20
4	Analiza wymagań	21
4.1	Przypadki użycia	21
4.1.1	Aktorzy	21
4.1.2	Diagramy przypadków użycia	23
4.1.3	Scenariusz przypadków użycia	32
4.2	Wymagania ogólne i dziedzinowe	32
4.3	Wymagania funkcjonalne	32
4.3.1	Funkcjonalności dla mapy	32
4.3.2	Funkcjonalności dla chatu	32
4.3.3	Funkcjonalności dla forum	32
4.3.4	Funkcjonalności dla konta użytkownika	32
4.3.5	Funkcjonalności dla logowania i rejestracji	42
4.3.6	Funkcjonalności dla wyszukiwarki spotów	43
4.3.7	Funkcjonalności dla motywu	45
4.4	Wymagania pozafunkcjonalne	47
4.5	Wymagania interfejs z otoczeniem	47
4.6	Wymagania na środowisko docelowe	47
5	Projekt	48
5.1	Wzorce projektowe	48
5.2	Architektura systemu	48
5.2.1	Diagram architektury	48
5.2.2	Komponenty systemu	48
5.3	Projekt bazy danych	48
5.3.1	Model danych	48

5.3.2	Diagram ERD	48
5.4	Architektura interfejsu użytkownika	48
5.4.1	Projekt strony głównej	48
5.4.2	Projekt panelu logowania	48
5.4.3	Projekt mapy	48
5.4.4	Projekt chatu	48
5.4.5	Projekt forum	48
5.4.6	Projekt konta użytkownika	48
6	Przebieg realizacji projektu	49
6.1	Sprint 1	49
6.2	Sprint 2	49
7	Realizacja Projektu	50
7.1	Implementacja backendu	50
7.1.1	Struktura projektu	50
7.1.2	Integracja z bazą danych	50
7.1.3	Obsługa uwierzytelnienia	50
7.1.4	Konteneryzacja	50
7.2	Implementacja frontendu	50
7.2.1	Struktura aplikacji	50
7.2.2	Zarządzanie stanem i przepływ danych	55
7.2.3	Integracja i komunikacja z backendem	57
7.2.4	Style	60
7.2.5	Wyszukiwarka spotów	64
7.2.6	Mapa	71
7.2.7	Chat	71
7.2.8	Forum	71
7.2.9	Konto użytkownika	71
7.2.10	Panel logowania	71
7.3	Implementacja CI/CD	71

8 Testy	72
8.1 Testy jednostkowe	72
8.2 Testy integracyjne	72
8.3 Testy E2E	72
8.4 Wyniki testów i wnioski	72
9 Prezentacja systemu	73
9.1 Strona główna	73
9.2 Strona mapy	73
9.3 Strona chatu	73
9.4 Strona forum	73
9.5 Panel logowania	73
9.6 Panel konta użytkownika	73
10 Nakład pracy	74
10.1 Ogólny nakład pracy	74
10.2 Indywidualne nakłady pracy	74
10.2.1 Adam Langmesser	74
10.2.2 Mateusz Redosz	74
10.2.3 Stanisław Oziemczuk	77
10.2.4 Kacper Badek	77
11 Podsumowanie	78
11.1 Osiągnięte rezultaty	78
11.2 Napotkane wyzwania	78
11.3 Plany na przyszłość	78
12 Słownik pojęć i skrótów	79
Spis tabel	80
Załączniki	82

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:	Droniarze
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 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 [DAD]** w wariantcie **Lean Life Cycle [DAD-LLF]**, 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 kodu](#). 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](#)([\[uml-def\]](#)) czy [BPMN](#)([\[bpmn-def\]](#)). 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

Usługa	GitHub Actions (CI) [github-actions-billing]
--------	--

Opis	Uruchomienia pipeline'ów CI/CD dla repozytorium GitHub.
Limit	3000 min/mies.

Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)

Usługa	Azure Blob Storage [azure-blob-scalability]
Opis	Magazyn plików (m.in. zdjęcia spotów, załączniki z czatu).
Limit	1 GB/mies.

Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage

Usługa	Mailtrap [mailtrap-limits]
Opis	Środowisko testowe SMTP oraz Email API do wysyłki maili.
Limit	150 maili/dzień

Tabela 3.3: Usługa zewnętrzna: Mailtrap

Usługa	LocationIQ [locationiq-pricing]
Opis	Geokodowanie adresu przy dodawaniu nowych spotów.
Limit	5 000 zapytań/dzień

Tabela 3.4: Usługa zewnętrzna: LocationIQ

Usługa	Google Maps (Maps URLs) [google-maps-urls]
Opis	Otwieranie nawigacji w aplikacji Map Google (deep link/URL).
Limit	Brak limitu w ramach dokumentowanego sposobu użycia.

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

Usługa	OpenFreeMap [openfreemap-docs , openfreemap-quickstart]
Opis	Publiczny serwer kafelków do renderu mapy na froncie.
Limit	30 000 zapytań/mies.

Tabela 3.6: Usługa zewnętrzna: OpenFreeMap

Usługa	Open-Meteo [open-meteo-usage]
Opis	Prognozy pogody wyświetlane dla spotów.
Limit	10 000 zapytań/dzień

Tabela 3.7: Usługa zewnętrzna: Open-Meteo

Usługa	Tenor GIF API [tenor-docs]
Opis	Wyszukiwanie GIF-ów w czacie.
Limit	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.8: Usługa zewnętrzna: Tenor GIF API

Usługa	Where the ISS at? [wheretheiss-docs]
Opis	HTTP API z bieżącą pozycją satelity, używane pomocniczo.
Limit	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

4.1 Przypadki użycia

4.1.1 Aktorzy

Niniejszy rozdział przedstawia aktorów wraz z opisami.

Użytkownik systemu - Reprezentuje każdą osobę korzystającą z aplikacji.

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

Użytkownik zalogowany - Ma dostęp do wszystkich darmowych funkcjonalności aplikacji. Zarządza kontem i ulubionymi spotami, dodaje posty i komentarze, korzysta z czatu.

Użytkownik premium - Użytkownik z wykupioną subskrypcją: ma dostęp do funkcji premium np. oznaczenie stref [PANSÁ](#) na mapie.

System Finansowo-księgowy - zewnętrzny system do prowadzenia księgowości, wystawiania faktur oraz rozliczania płatności.

Usługa SMTP - usługa Simple Mail Transfer Protocol wykorzystywana do wysyłania wiadomości e-mail.

Bramka Płatnicza - usługa obsługująca płatności elektroniczne (karta płatnicza, BLIK itp.).

Usługa OAuth - usługa uwierzytelniania i autoryzacji użytkowników z wykorzystaniem zewnętrznych dostawców tożsamości.

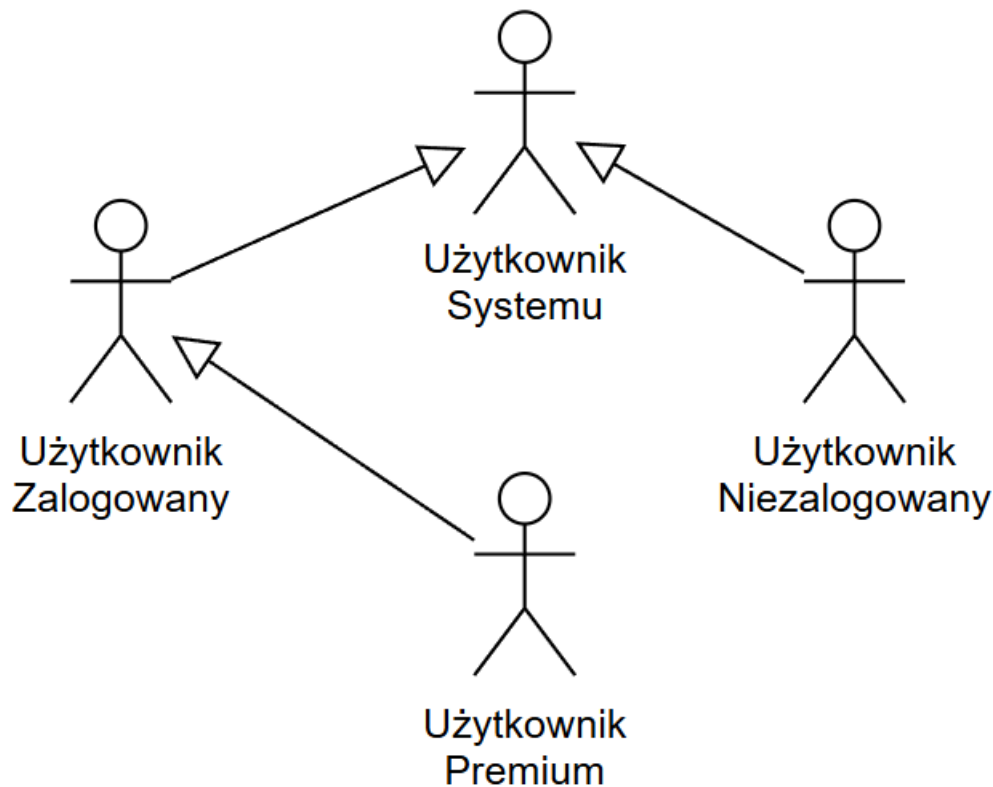
Usługa do przechowywania plików w chmurze - magazyn plików w chmurze służący do przechowywania załączników i multimediów użytkowników.

Usługa do wyświetlania mapy - zewnętrzne API dostarczające kafelki map, nawigację oraz dane geolokalizacyjne.

Usługa danych pogodowych - usługa udostępniająca bieżące warunki pogodowe oraz prognozy dla wybranych lokalizacji.

Usługa do GIF'ów - serwis umożliwiający wyszukiwanie i osadzanie animowanych obrazów GIF w aplikacji.

Usługa do określania strefy czasowej - usługa ustalająca strefę czasową spota na podstawie jego współrzędnych geograficznych.

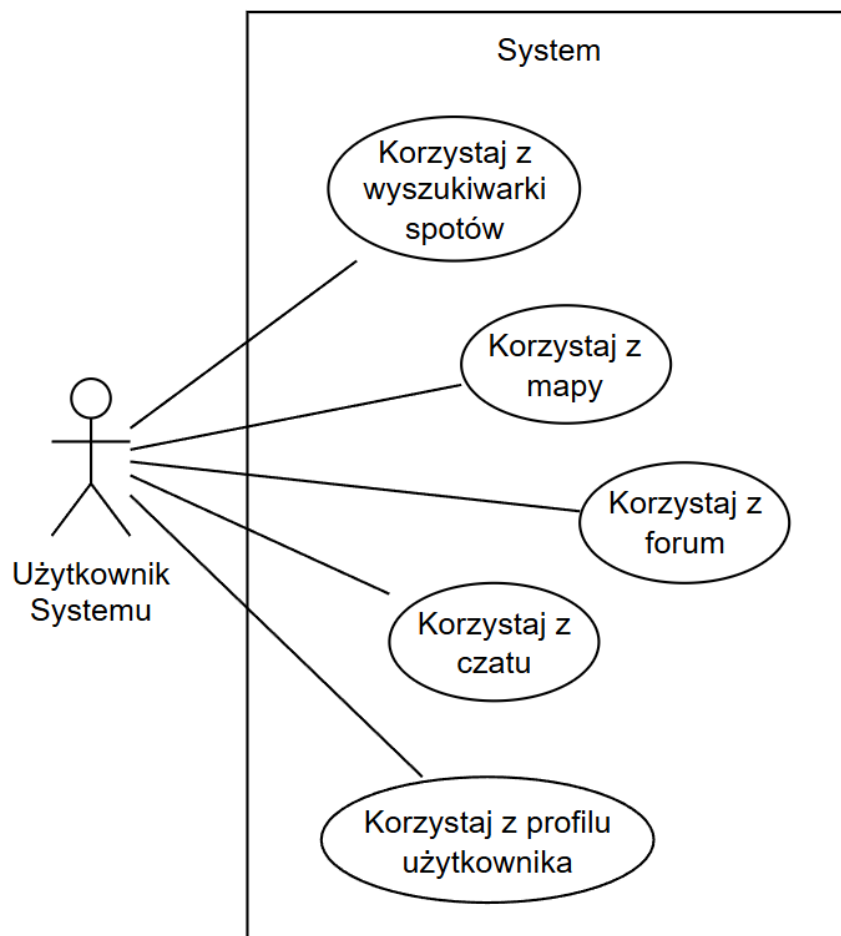


Rysunek 4.1: Diagram hierarchii użytkowników systemu

Na diagramie przedstawiono hierarchię aktorów systemu reprezentujących użytkownika. Podstawową rolą jest Użytkownik systemu, która reprezentuje każdą osobę korzystającą z aplikacji. Z niej dziedziczą dwie bardziej szczegółowe role: Użytkownik niezalogowany (ma dostęp tylko do funkcji publicznych) oraz Użytkownik zalogowany (posiada konto i dostęp do funkcji wymagających uwierzytelnienia). Dodatkowo Użytkownik premium jest wyspecjalizowaną wersją użytkownika zalogowanego i oprócz standardowych funkcji otrzymuje dostęp do funkcji premium.

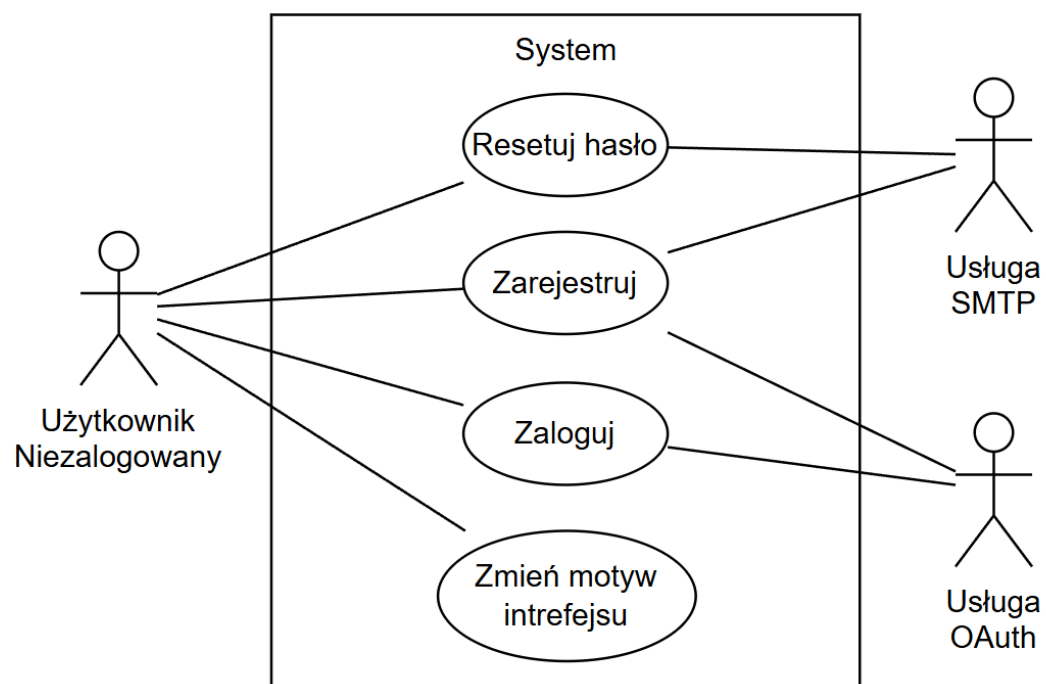
4.1.2 Diagramy przypadków użycia

Niniejszy rozdział przedstawia diagramy przypadków użycia.

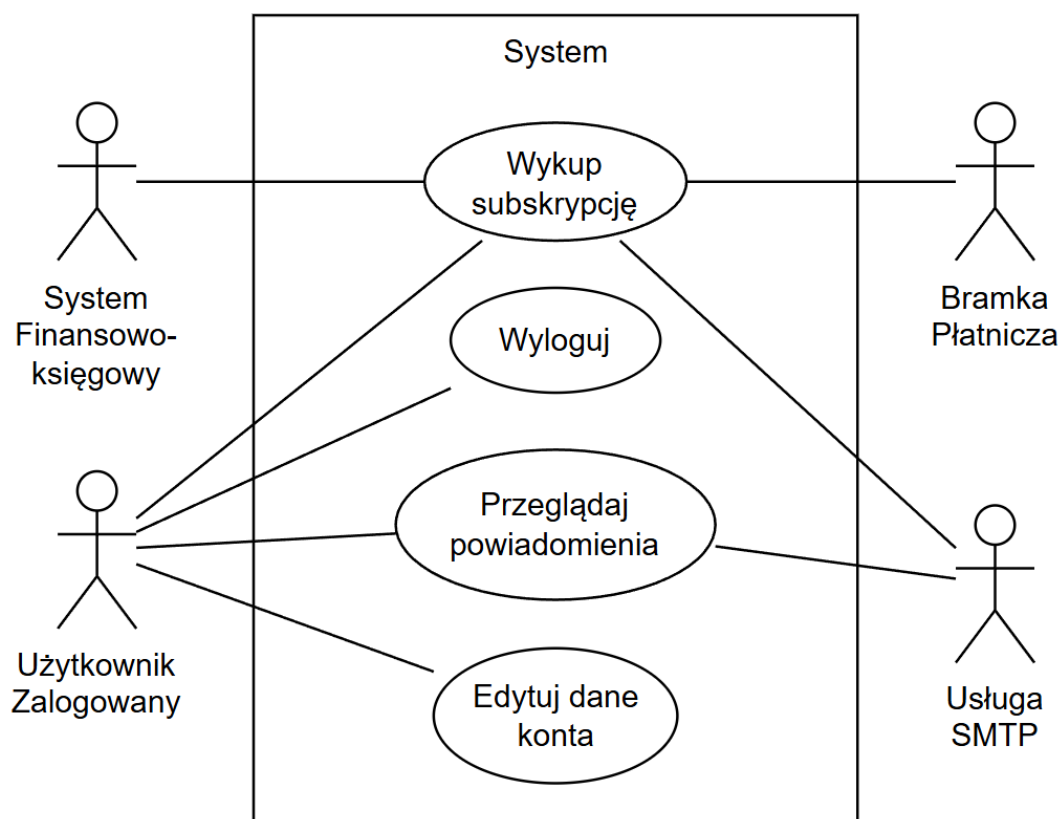


Rysunek 4.2: Wysokopoziomowy diagram przypadków użycia

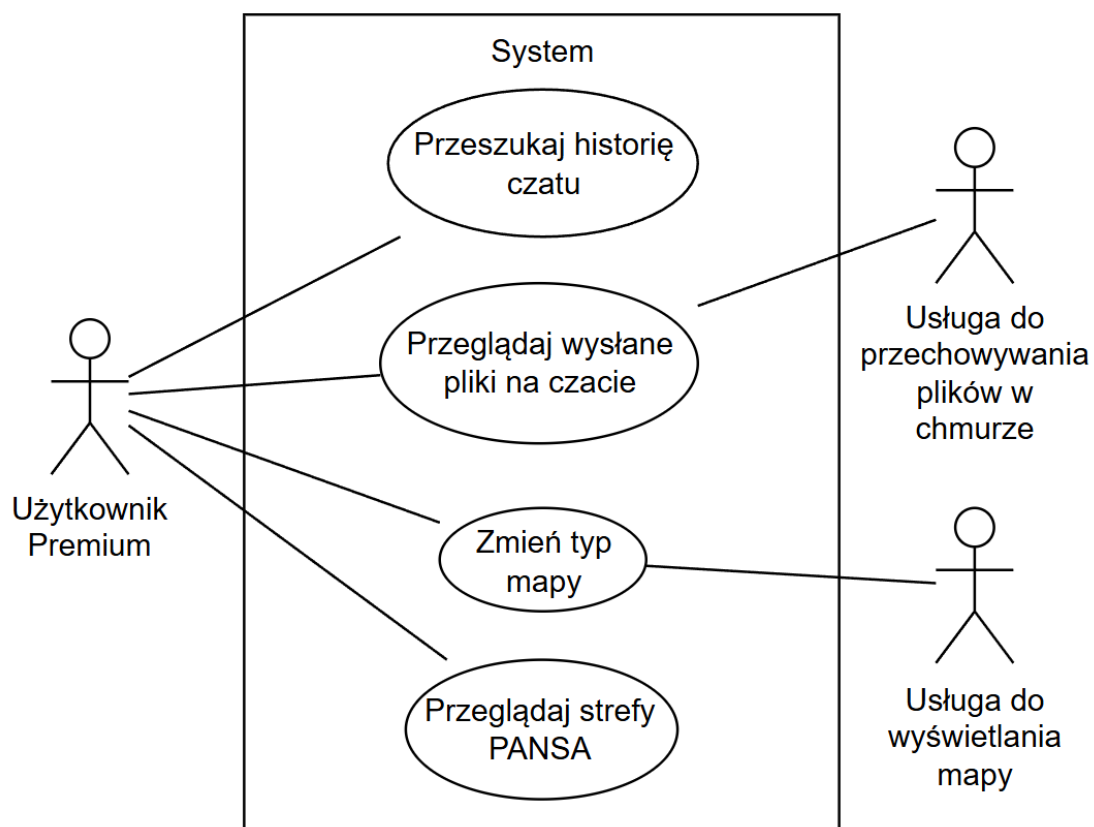
Diagram przedstawia podstawowe interakcje użytkownika z systemem. Na jego podstawie zespół projektowy podzielił architekturę aplikacji na 5 modułów: wyszukiwarkę spotów, mapę spotów, forum, czat oraz profil użytkownika. Pozostałe diagramy są bardziej szczegółowe.



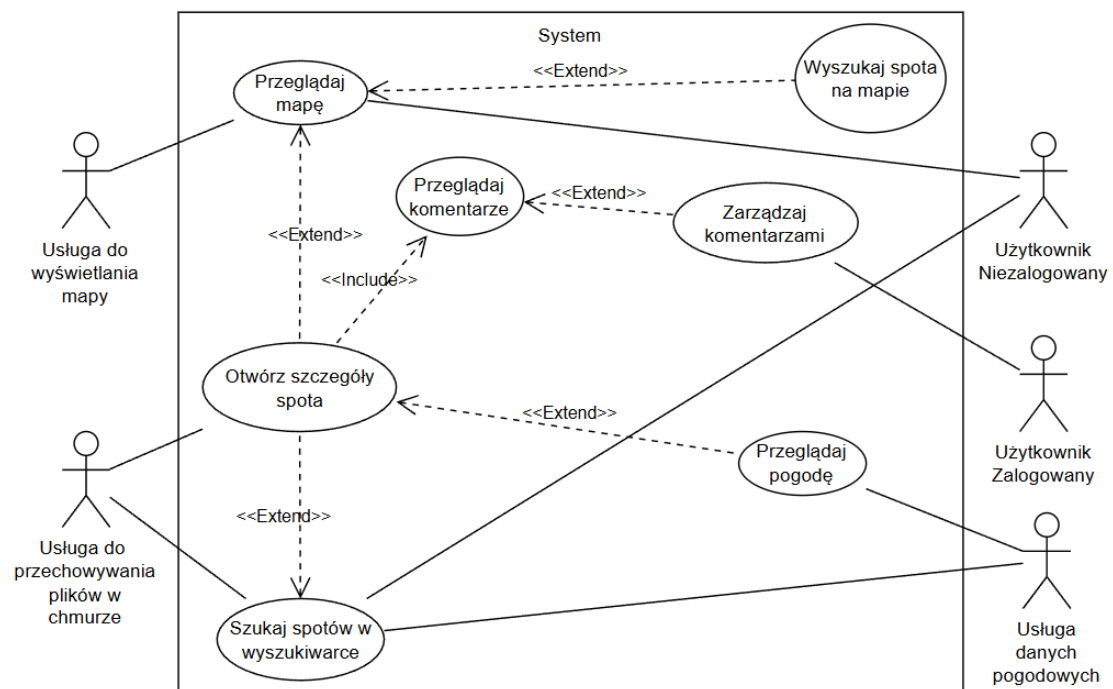
Rysunek 4.3: Diagram przypadków użycia dla użytkownika niezalogowanego



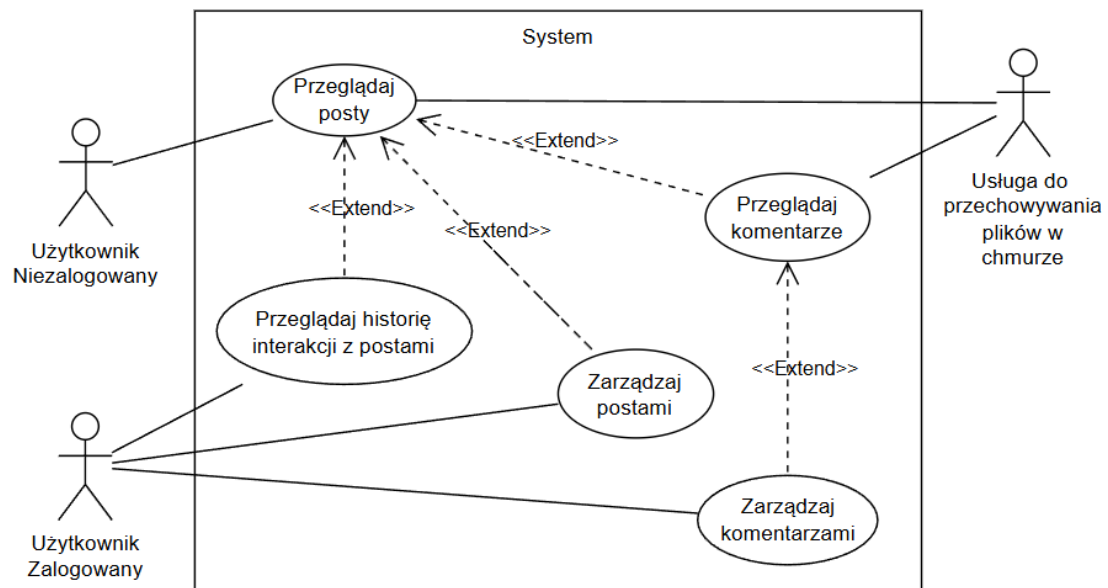
Rysunek 4.4: Diagram przypadków użycia dla użytkownika zalogowanego



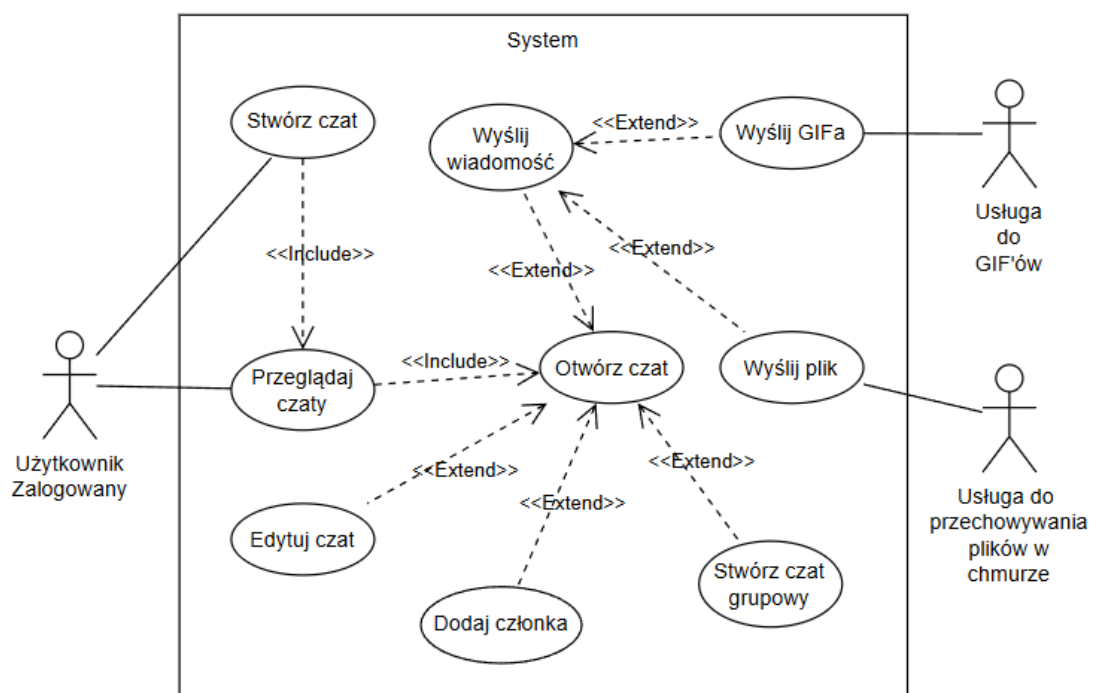
Rysunek 4.5: Diagram przypadków użycia dla użytkownika premium



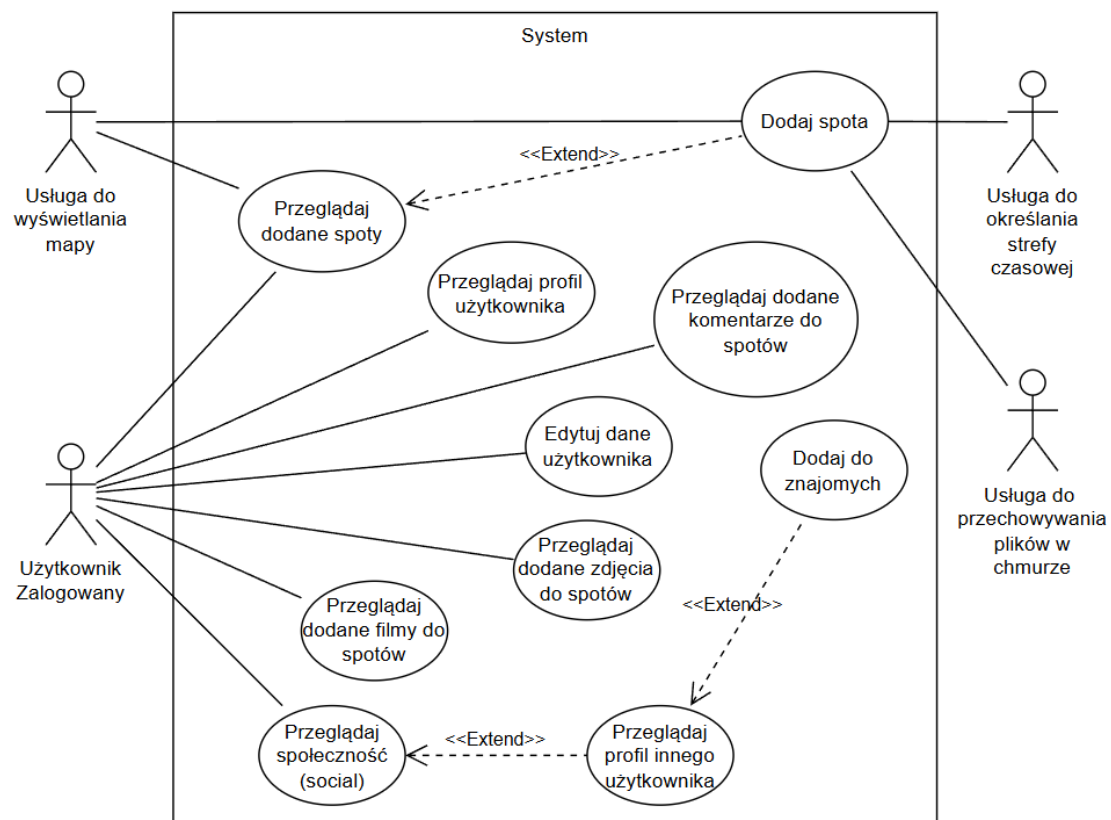
Rysunek 4.6: Diagram przypadków użycia wyszukiwarki spotów oraz mapy



Rysunek 4.7: Diagram przypadków użycia forum



Rysunek 4.8: Diagram przypadków użycia czatu



Rysunek 4.9: Diagram przypadków użycia profilu użytkownika

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

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Profil użytkownika		
Opis:	Jako użytkownik chcę mieć dostęp do strony profilu, aby sprawdzić informacje o swoim koncie.		
Kryteria akceptacji:	Użytkownik widzi liczby: znajomych, obserwowanych i obserwujących, a także najpopularniejsze zdjęcia.		
Dane wejściowe:	Lista zdjęć oraz liczby: znajomych, obserwujących i obserwowanych.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlone informacje o profilu.		
Sytuacje wyjątkowe:	Błąd połączenia z API; brak danych profilu; brak uprawnień (401/403).		
Szczegóły implementacji:	Frontend: React + Tailwind; pobieranie danych profilu przez <code>@tanstack/react-query</code> i <code>axios</code> z <code>withCredentials</code> . Prezentacja w widoku profilu.		
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.1: Profil użytkownika

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista dodanych spotów		
Opis:	Jako użytkownik chcę sprawdzić listę spotów, które dodałem.		
Kryteria akceptacji:	Użytkownik widzi listę własnych dodanych spotów.		
Dane wejściowe:	Lista dodanych spotów.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista dodanych spotów.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy z backendu (endpoint listy własnych spotów) przez <code>react-query</code> + <code>axios</code> ; prezentacja listy z podstawowymi danymi.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.2: Lista dodanych spotów

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Dodanie spota		
Opis:	Jako użytkownik chcę mieć dostęp do formularza dodania spota.		
Kryteria akceptacji:	Użytkownik ma dostęp do formularza dodania spota i może go wysłać.		
Dane wejściowe:	Formularz dodania spota.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlony formularz dodania spota (po wysłaniu: zapis na backendzie).		
Sytuacje wyjątkowe:	Nieprawidłowe dane formularza; błąd połączenia z API.		
Szczegóły implementacji:	Formularz w React; walidacja przeglądarkowa; wysyłka przez <code>axios</code> (POST) z <code>withCredentials</code> .		
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.3: Dodanie spota

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista zdjęć		
Opis:	Jako użytkownik chcę mieć dostęp do listy zdjęć, które dodałem na forum, do komentarzy pod spotem oraz do spota.		
Kryteria akceptacji:	Użytkownik widzi listę swoich zdjęć.		
Dane wejściowe:	Lista zdjęć.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista zdjęć.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy zdjęć użytkownika przez <code>react-query</code> + <code>axios</code> ; prezentacja z miniaturowymi.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.4: Lista zdjęć

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista filmów		
Opis:	Jako użytkownik chcę mieć dostęp do listy filmów, które dodałem na forum, do komentarzy pod spotem oraz do spota.		
Kryteria akceptacji:	Użytkownik widzi listę swoich filmów.		
Dane wejściowe:	Lista filmów.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista filmów.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy filmów użytkownika przez <code>react-query</code> + <code>axios</code> ; prezentacja z miniaturowymi.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.5: Lista filmów

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista znajomych		
Opis:	Jako użytkownik chcę mieć dostęp do listy znajomych.		
Kryteria akceptacji:	Użytkownik ma dostęp do listy znajomych.		
Dane wejściowe:	Lista znajomych.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista znajomych.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy znajomych przez <code>react-query</code> + <code>axios</code> ; standardowa prezentacja listy.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.6: Lista znajomych

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista obserwujących		
Opis:	Jako użytkownik chcę mieć dostęp do listy obserwujących.		
Kryteria akceptacji:	Użytkownik ma dostęp do listy obserwujących.		
Dane wejściowe:	Lista obserwujących.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista obserwujących.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy obserwujących przez <code>react-query</code> + <code>axios</code> ; standardowa prezentacja listy.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.7: Lista obserwujących

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista obserwowanych		
Opis:	Jako użytkownik chcę mieć dostęp do listy obserwowanych.		
Kryteria akceptacji:	Użytkownik ma dostęp do listy obserwowanych.		
Dane wejściowe:	Lista obserwowanych.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista obserwowanych.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy obserwowanych przez <code>react-query</code> + <code>axios</code> ; standardowa prezentacja listy.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.8: Lista obserwowanych

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista spotów		
Opis:	Jako użytkownik chcę mieć dostęp do listy spotów, które polubiłem, odwiedziłem i planuję odwiedzić.		
Kryteria akceptacji:	Użytkownik ma dostęp do listy spotów w wymienionych kategoriach.		
Dane wejściowe:	Listy spotów: polubione, odwiedzone, planowane.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlone listy spotów.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie list przez <code>react-query</code> + <code>axios</code> ; prezentacja w zakładkach/kategoriach.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.9: Lista polubionych/odwiedzonych/planowanych spotów

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Lista komentarzy		
Opis:	Jako użytkownik chcę mieć dostęp do listy komentarzy.		
Kryteria akceptacji:	Użytkownik ma dostęp do listy swoich komentarzy.		
Dane wejściowe:	Lista komentarzy.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlona lista komentarzy.		
Sytuacje wyjątkowe:	Brak wyników; błąd połączenia z API.		
Szczegóły implementacji:	Pobranie listy komentarzy użytkownika przez <code>react-query</code> + <code>axios</code> ; standardowa prezentacja listy.		
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.10: Lista komentarzy

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Ustawienia		
Opis:	Jako użytkownik chcę mieć możliwość zmiany danych.		
Kryteria akceptacji:	Użytkownik może edytować wybrane dane profilu i zapisać zmiany.		
Dane wejściowe:	Formularz edycji danych.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Wyświetlony formularz edycji; po zapisie — zaktualizowane dane.		
Sytuacje wyjątkowe:	Nieprawidłowe dane formularza; błąd połączenia z API.		
Szczegóły implementacji:	Formularz w React; walidacja pól; wysyłka przez <code>axios</code> (PUT/PATCH) z <code>withCredentials</code> . Po sukcesie — komunikat i odświeżenie danych przez <code>react-query</code> .		
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.11: Ustawienia profilu

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Resetowanie hasła		
Opis:	Jako użytkownik chcę mieć możliwość zresetowania hasła do swojego konta.		
Kryteria akceptacji:	Po kliknięciu w odpowiedni link użytkownik może zresetować hasło do konta.		
Dane wejściowe:	Adres e-mail użytkownika do wysłania linku resetującego.		
Warunki początkowe:	Użytkownik podał poprawny adres e-mail użyty przy rejestracji.		
Warunki końcowe:	Hasło zresetowane po przejściu całej procedury.		
Sytuacje wyjątkowe:	Niepoprawny adres e-mail; wygasły lub nieprawidłowy token resetu; błąd połączenia z API.		
Szczegóły implementacji:	Frontend: formularz „zapomniałem hasła” (POST do endpointu wysyłającego link resetu) oraz formularz ustawienia nowego hasła (POST/PATCH z tokenem). Wysyłka przez axios ; obsługa komunikatów o powodzeniu/błędach.		
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.12: Resetowanie hasła

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Dodawanie użytkowników do listy znajomych		
Opis:	Jako użytkownik chcę mieć możliwość dodawania innych użytkowników do listy znajomych.		
Kryteria akceptacji:	Użytkownik może dodać innego użytkownika do swojej listy znajomych.		
Dane wejściowe:	Dane użytkownika, którego chcemy dodać do znajomych.		
Warunki początkowe:	Użytkownik jest zalogowany.		
Warunki końcowe:	Znajomy dodany do listy i widoczny w profilu użytkownika.		
Sytuacje wyjątkowe:	Brak uprawnień; użytkownik już jest znajomym; błąd połączenia z API.		
Szczegóły implementacji:	Akcja wysłania zaproszenia do znajomych przez <code>axios</code> ; po akceptacji — aktualizacja listy (odświeżenie <code>react-query</code>).		
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.13: Dodawanie do znajomych

4.3.5 Funkcjonalności dla logowania i rejestracji

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Logowanie i rejestracja		
Opis:	Jako użytkownik chcę mieć możliwość zalogowania się do aplikacji, korzystając z formularza lub poprzez konto Google lub GitHub.		
Kryteria akceptacji:	Użytkownik może zalogować się do aplikacji zarówno za pomocą standardowego formularza, jak i przy użyciu konta w serwisie Google lub GitHub.		
Dane wejściowe:	Dane użytkownika: adres e-mail, hasło; przy rejestracji dodatkowo nazwa użytkownika.		
Warunki początkowe:	Użytkownik niezalogowany.		
Warunki końcowe:	Działające formularze rejestracji i logowania oraz możliwość logowania za pomocą konta Google i GitHub.		
Sytuacje wyjątkowe:	Błędne dane logowania; przerwana lub nieudana autoryzacja u dostawcy (Google/GitHub).		
Szczegóły implementacji:	Frontend: formularze w React; wysyłka żądań przez <code>axios</code> z <code>withCredentials</code> . SSO: integracja z Google i GitHub (OAuth 2.0) z przekierowaniem i ustawieniem sesji po stronie backendu (<code>httpOnly</code> cookie). Obsługa statusu 401 zgodnie z mechanizmem wylogowania.		
Udziałowiec:	Zespół projektowy 2.1 , promotor 2.2 , droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.14: Logowanie i rejestracja

4.3.6 Funkcjonalności dla wyszukiwarki spotów

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Strona główna z podstawowymi filtrami		
Opis:	Jako użytkownik chcę mieć dostęp do strony głównej, która wyświetla karuzelę z najpopularniejszymi spotami oraz listę spotów, które można filtrować.		
Kryteria akceptacji:	Użytkownik widzi karuzelę najpopularniejszych miejsc. Karuzela zawiera zdjęcia, nazwę miejsca i miasto. Użytkownik może filtrować miejsca według lokalizacji (kraj, region, miasto).		
Dane wejściowe:	Lokalizacja użytkownika (kraj, region, miasto); dane z bazy spotów.		
Warunki początkowe:	Użytkownik nie musi być zalogowany.		
Warunki końcowe:	Użytkownik widzi popularne miejsca z wybranego miasta (np. Gdańsk) i może przejść do szczegółów danego miejsca.		
Sytuacje wyjątkowe:	Brak wyników dla wybranych filtrów; błąd połączenia z API.		
Szczegóły implementacji:	Frontend: React + Tailwind. Pobieranie danych przez <code>@tanstack/react-query</code> i <code>axios</code> (GET do backendu z parametrami lokalizacji). Filtry lokalizacji mapowane na parametry zapytania.		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.15: Strona główna — podstawowe filtry

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Strona główna z zaawansowanymi filtrami		
Opis:	Jako użytkownik chcę mieć dostęp do strony głównej, która wyświetla listę spotów, które można filtrować i sortować.		
Kryteria akceptacji:	Użytkownik widzi listę, którą może filtrować według miasta, tagów i oceny spotu, a także sortować po ocenie i popularności.		
Dane wejściowe:	Lokalizacja użytkownika (miasto), wartości filtrów i sortowania; dane z bazy spotów.		
Warunki początkowe:	Użytkownik nie musi być zalogowany.		
Warunki końcowe:	Użytkownik widzi wyniki zgodne z zastosowanymi filtrami i sortowaniem oraz może przejść do szczegółów danego miejsca.		
Sytuacje wyjątkowe:	Brak wyników po zastosowaniu filtrów; błąd połączenia z API.		
Szczegóły implementacji:	Frontend: React + Tailwind. Pobieranie danych przez <code>@tanstack/react-query</code> i <code>axios</code> z parametrami: lokalizacja, tagi, minimalna ocena oraz kryterium sortowania.		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.		
Wymagania powiązane:	SPXX		

Tabela 4.16: Strona główna — zaawansowane filtry

4.3.7 Funkcjonalności dla motywu

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Ustawienia motywu		
Opis:	Jako użytkownik chcę móc zmienić motyw aplikacji.		
Kryteria akceptacji:	Dostępna jest opcja przełączenia motywu na <i>jaśny</i> lub <i>ciemny</i> ; zmiana następuje bez przeładowania strony; ustawienie działa we wszystkich widokach.		
Dane wejściowe:	Preferencje użytkownika dotyczące motywu.		
Warunki początkowe:	Brak.		
Warunki końcowe:	Zmiana motywu widoczna jest natychmiast po kliknięciu przycisku.		
Sytuacje wyjątkowe:	Brak.		
Szczegóły implementacji:	Tailwind CSS z <code>darkMode: 'class'</code> ; motyw przełączany przez dodanie/usunięcie klasy <code>dark</code> na elemencie <code><html></code> ;		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.17: Ustawienia motywu (ręczna zmiana)

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Zapamiętywanie preferencji motywu		
Opis:	Jako użytkownik chcę, aby moja preferencja motywu była zapamiętana i przywracana przy kolejnym użyciu aplikacji.		
Kryteria akceptacji:	Wybrany motyw jest przywracany po ponownym włączeniu i odświeżeniu strony; preferencja jest zapamiętywana lokalnie w przeglądarce.		
Dane wejściowe:	Preferencje użytkownika zapisane lokalnie.		
Warunki początkowe:	FOXX dostępne.		
Warunki końcowe:	Motyw po uruchomieniu odpowiada ostatniej decyzji użytkownika.		
Sytuacje wyjątkowe:	Brak dostępu do magazynu trwałego — preferencja przechowywana w local storage.		
Szczegóły implementacji:	Zapis w <code>localStorage</code> pod kluczem <code>theme</code> (<code>dark</code> lub <code>light</code>); krótki skrypt umieszczony w <code>App.jsx</code> przed startem odczytuje <code>localStorage</code> i odpowiednio dodaje lub usuwa klasę <code>dark</code> na <code><html></code> (eliminuje mignięcie stylów).		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.18: Zapamiętanie preferencji motywu

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	S
Nazwa:	Przełącznik motywu w Sidebar		
Opis:	Jako użytkownik chcę szybko zmieniać motyw bez wchodzenia w ustawienia.		
Kryteria akceptacji:	W Sidebar dostępny jest przełącznik <i>Jasny-/Ciemny</i> ; posiada odpowiednio ikony <i>słońca/księżyca</i> ; zmiana następuje natychmiast.		
Dane wejściowe:	Bieżąca preferencja motywu.		
Warunki początkowe:	FOXX, FOXX dostępne.		
Warunki końcowe:	Motyw zmieniony; preferencja zaktualizowana.		
Sytuacje wyjątkowe:	Brak.		
Szczegóły implementacji:	Przycisk typu <i>toggle</i> wywołuje funkcję, która przełącza klasę <code>dark</code> na <code>document.documentElement</code> oraz aktualizuje <code>localStorage</code> (<code>theme = 'dark' 'light'</code>); brak przeładowania strony.		
Udziałowiec:	Zespół projektowy 2.1 , promotor 2.2 , droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.19: Szybki przełącznik motywu w interfejsie

4.4 Wymagania pozafunkcjonalne

4.5 Wymagania interfejs z otoczeniem

4.6 Wymagania na środowisko docelowe

Rozdział 5

Projekt

5.1 Wzorce projektowe

5.2 Architektura systemu

5.2.1 Diagram architektury

5.2.2 Komponenty systemu

5.3 Projekt bazy danych

5.3.1 Model danych

5.3.2 Diagram ERD

5.4 Architektura interfejsu użytkownika

5.4.1 Projekt strony głównej

5.4.2 Projekt panelu logowania

5.4.3 Projekt mapy

5.4.4 Projekt chatu

5.4.5 Projekt forum

5.4.6 Projekt konta użytkownika

Rozdział 6

Przebieg realizacji projektu

6.1 Sprint 1

6.2 Sprint 2

Rozdział 7

Realizacja Projektu

7.1 Implementacja backendu

7.1.1 Struktura projektu

7.1.2 Integracja z bazą danych

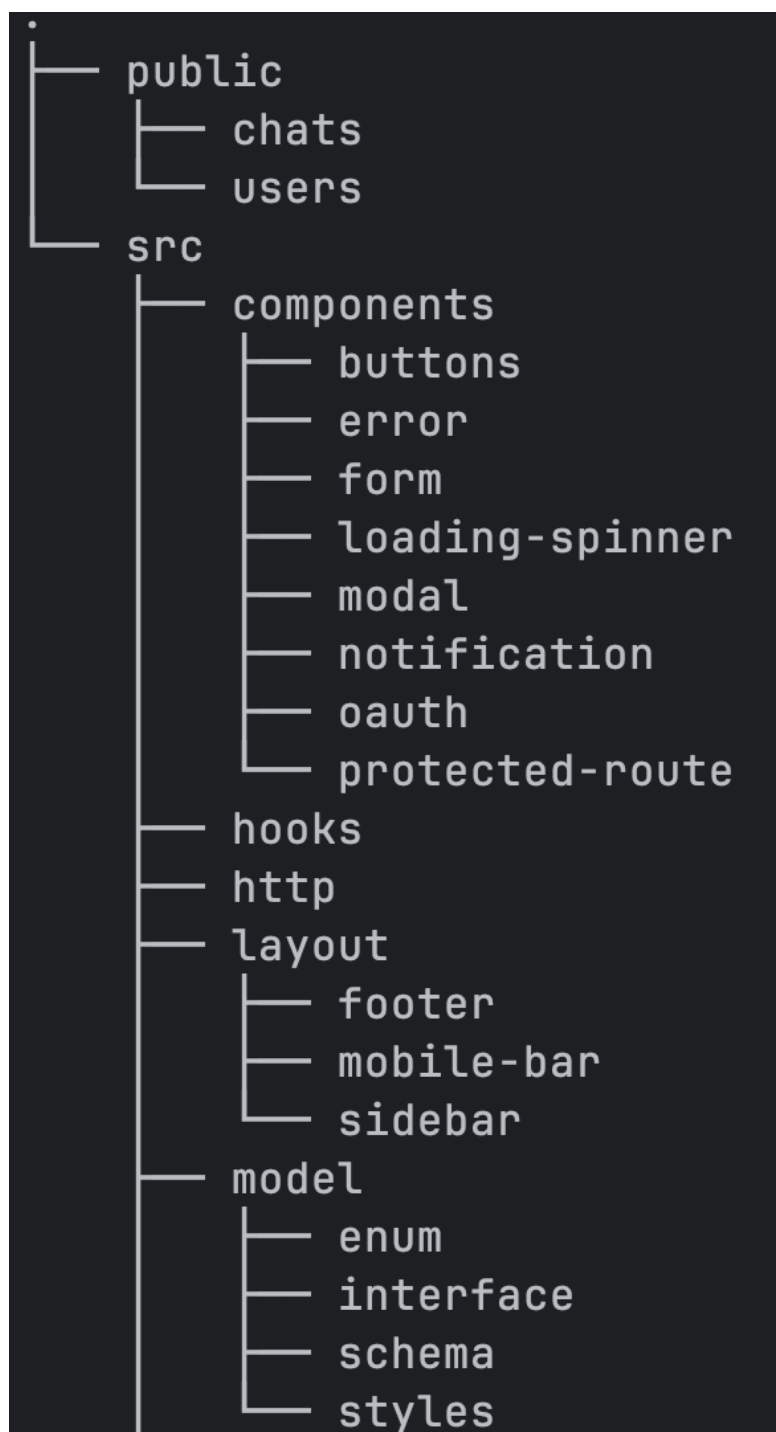
7.1.3 Obsługa uwierzytelnienia

7.1.4 Konteneryzacja

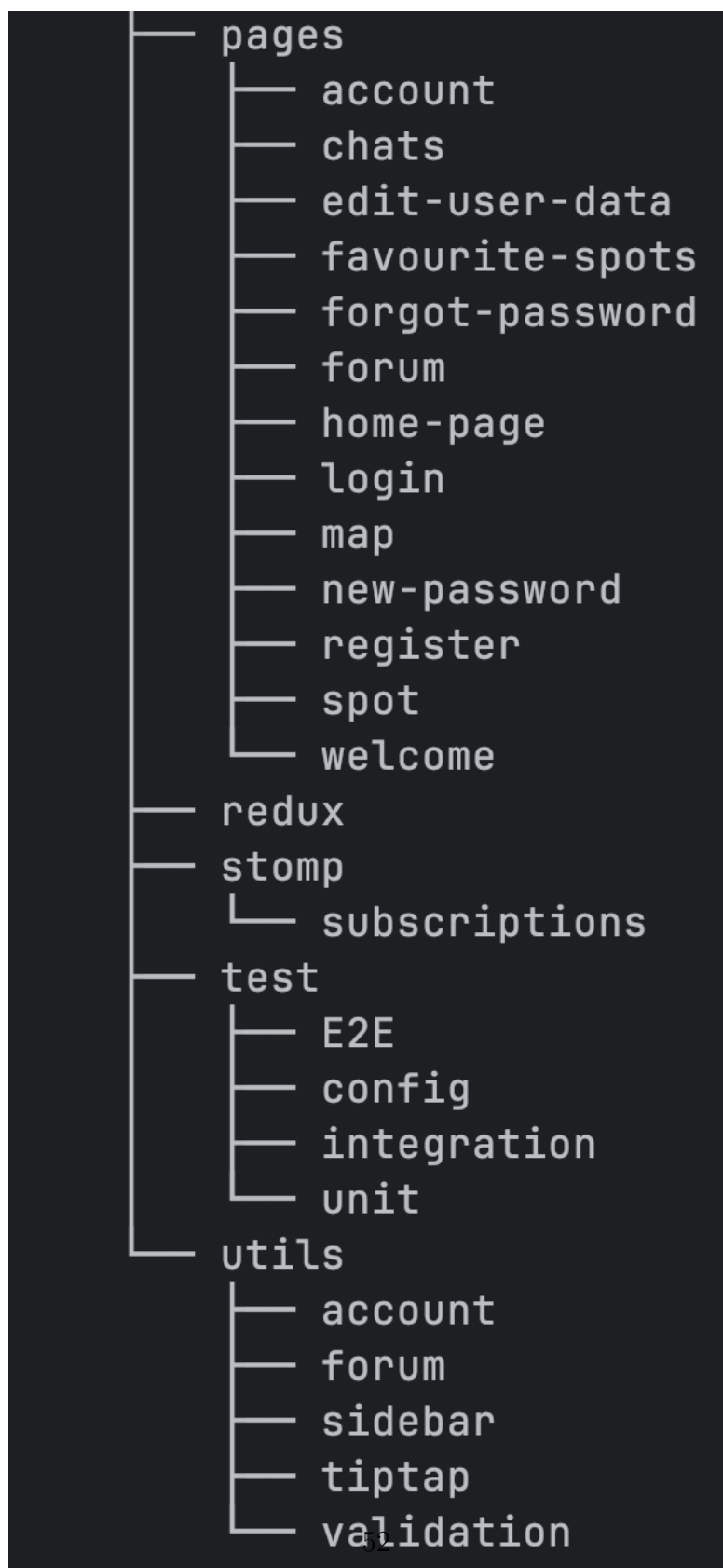
7.2 Implementacja frontendu

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`) [[react-use-state](#)], jak i ze [Stanu](#) globalnego, utrzymywanego przez [Bibliotekę React Redux](#) [[redux](#)]. Globalny [Stan](#) został wprowadzony po to, aby możliwie najbardziej ograniczyć przekazywanie [Propsów](#) w głąb drzewa komponentów oraz uniknąć niepotrzebnych ponownych renderów.

Do przechowywania [Stanu](#) lokalnego, ograniczonego tylko do danego komponentu (lub jego najbliższych elementów podrzędnych), wykorzystujemy [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` [axios] oraz Biblioteki TanStack Query [tanstack-query]. 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 [[tailwind](#)]. 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](#) [**motion**]. 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 wyszukiwarkę, 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

Spis tabel

2.1	Zespół projektowy	7
2.2	Promotor	8
2.3	Droniarze	8
Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)		18
Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage		18
Tabela 3.3: Usługa zewnętrzna: Mailtrap		18
Tabela 3.4: Usługa zewnętrzna: LocationIQ		18
Tabela 3.5: Usługa zewnętrzna: Google Maps (Maps URLs)		19
Tabela 3.6: Usługa zewnętrzna: OpenFreeMap		19
Tabela 3.7: Usługa zewnętrzna: Open-Meteo		19
Tabela 3.8: Usługa zewnętrzna: Tenor GIF API		19
Tabela 3.9: Usługa zewnętrzna: Where the ISS at?		20
4.1	Profil użytkownika	32
4.2	Lista dodanych spotów	33
4.3	Dodanie spotu	34
4.4	Lista zdjęć	35
4.5	Lista filmów	35
4.6	Lista znajomych	36
4.7	Lista obserwujących	36
4.8	Lista obserwowanych	37
4.9	Lista polubionych/odwiedzonych/planowanych spotów	37
4.10	Lista komentarzy	38
4.11	Ustawienia profilu	39

4.12	Resetowanie hasła	40
4.13	Dodawanie do znajomych	41
4.14	Logowanie i rejestracja	42
4.15	Strona główna — podstawowe filtry	43
4.16	Strona główna — zaawansowane filtry	44
4.17	Ustawienia motywu (ręczna zmiana)	45
4.18	Zapamiętanie preferencji motywu	46
4.19	Szybki przełącznik motywu w interfejsie	47

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.