



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: 6 grudnia 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	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
4	Analiza wymagań	20
4.1	Przypadki użycia	20
4.1.1	Aktorzy	20
4.1.2	Diagramy przypadków użycia	22
4.1.3	Scenariusz przypadków użycia	31
4.2	Wymagania ogólne i dziedzinowe	31
4.3	Wymagania funkcjonalne	31
4.3.1	Funkcjonalności dla mapy	31
4.3.2	Funkcjonalności dla chatu	31
4.3.3	Funkcjonalności dla forum	31
4.3.4	Funkcjonalności dla konta użytkownika	31
4.3.5	Funkcjonalności dla logowania i rejestracji	41
4.3.6	Funkcjonalności dla wyszukiwarki spotów	42
4.3.7	Funkcjonalności dla motywu	44
4.4	Wymagania pozafunkcjonalne	46
4.5	Wymagania interfejs z otoczeniem	46
4.6	Wymagania na środowisko docelowe	46
5	Projekt	47
5.1	Wzorce projektowe	47
5.2	Architektura systemu	47
5.2.1	Diagram architektury	47
5.2.2	Komponenty systemu	47
5.3	Projekt bazy danych	47
5.3.1	Model danych	47

5.3.2	Diagram ERD	47
5.4	Architektura interfejsu użytkownika	47
5.4.1	Projekt strony głównej	47
5.4.2	Projekt panelu logowania	47
5.4.3	Projekt mapy	47
5.4.4	Projekt chatu	47
5.4.5	Projekt forum	47
5.4.6	Projekt konta użytkownika	47
6	Przebieg realizacji projektu	48
6.1	Faza przedprojektowa (lipiec–wrzesień 2024)	48
6.2	Etap 1 (październik 2024 – styczeń 2025)	51
6.3	Etap 2 (luty 2025 – wrzesień 2025)	55
6.4	Etap 3 (październik 2025 – styczeń 2026)	64
7	Realizacja Projektu	67
7.1	Implementacja backendu	67
7.1.1	Struktura projektu	67
7.1.2	Integracja z bazą danych	67
7.1.3	Obsługa uwierzytelnienia	67
7.1.4	Konteneryzacja	67
7.2	Implementacja frontendu	67
7.2.1	Struktura aplikacji	67
7.2.2	Zarządzanie stanem i przepływ danych	72
7.2.3	Integracja i komunikacja z backendem	74
7.2.4	Style	77
7.2.5	Wyszukiwarka spotów	81
7.2.6	Mapa	88
7.2.7	Chat	88
7.2.8	Forum	88
7.2.9	Konto użytkownika	88
7.2.10	Panel logowania	88

7.3	Implementacja CI/CD	88
8	Testy	89
8.1	Testy jednostkowe	89
8.2	Testy integracyjne	89
8.3	Testy E2E	89
8.4	Wyniki testów i wnioski	89
9	Prezentacja systemu	90
9.1	Strona główna	90
9.2	Strona mapy	90
9.3	Strona chatu	90
9.4	Strona forum	90
9.5	Panel logowania	90
9.6	Panel konta użytkownika	90
10	Nakład pracy	91
10.1	Ogólny nakład pracy	91
10.2	Indywidualne nakłady pracy	91
10.2.1	Adam Langmesser	91
10.2.2	Mateusz Redosz	91
10.2.3	Stanisław Oziemczuk	94
10.2.4	Kacper Badek	94
11	Podsumowanie	95
11.1	Osiągnięte rezultaty	95
11.2	Napotkane wyzwania	95
11.3	Plany na przyszłość	95
12	Słownik pojęć i skrótów	96
	Spis tabel	110
	Bibliografia	112
	Załączniki	114

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 udziałowca:	Zespół projektowy
Opis:	Zespół czterech studentów odpowiedzialnych za analizę, projekt, implementację, testy oraz dokumentację systemu.
Typ:	ożywiony, bezpośredni
Perspektywa:	Techniczna, wykonawcza.
Ograniczenia:	Ograniczone zasoby czasowe i doświadczenie komercyjne.
Powiązane wymagania:	Wymagania funkcjonalne i techniczne systemu, możliwość realizacji w ramach projektu dyplomowego.

Tabela 2.1: Karta udziałowca: Zespół projektowy

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

Tabela 2.2: Karta udziałowca: Promotor

KARTA UDZIAŁOWCA	
Identyfikator:	UO3
Nazwa udziałowca:	Droniarze
Opis:	Główna grupa docelowa systemu – osoby latające dronami rekreacyjnie lub półprofesjonalnie, szukające miejsc do lotów i wymiany doświadczeń.
Typ:	ożywiony, bezpośredni
Perspektywa:	Użytkownik końcowy: prostota obsługi, rzetelne informacje o spotach, wygodne dzielenie się treściami.
Ograniczenia:	Brak wpływu na architekturę techniczną systemu; oczekują intuicyjnego interfejsu.

Powiązane wymagania:	Lista spotów, informacje o ograniczeniach prawnych, oceny i komentarze, dodawanie treści oraz podstawowe funkcje społecznościowe.
-----------------------------	---

Tabela 2.3: Karta udziałowca: [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** [1] w wariacie **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.

2024				2025												2026	
09	10	11	12	01	02	03	04	05	06	07	08	09	10	11	12	01	02
Etap 1																	
				Etap 2													
												Etap 3					

- Etap 1 (październik 2024 – styczeń 2025)
 - Wybór tematu projektu.
 - Analiza grupy docelowej.

- Wstępne opracowanie wymagań.
- Rozpoczęcie prac deweloperskich.
- **Etap 2 (luty 2025 – wrzesień 2025)**
 - Prace deweloperskie nad aplikacją.
 - Ciągła weryfikacja oraz korekcja wymagań postawionych systemowi.
 - Prace nad dokumentacją.
- **Etap 3 (październik 2025 – styczeń 2026)**
 - Finalizacja prac deweloperskich.
 - Implementacja testów automatycznych głównych funkcjonalności.
 - Ukończenie dokumentacji.

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](#)([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

Niniejszy rozdział zawiera spis zewnętrznych [API](#) oraz usług użytych w projekcie.

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ1
Nazwa:	GitHub Actions (CI) [5]
Opis:	Uruchomienia pipeline'ów CI/CD dla repozytorium GitHub.
Limit:	3000 min/mies.

Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ2
Nazwa:	Azure Blob Storage [6]
Opis:	Magazyn plików (m.in. zdjęcia spotów, załączniki z czatu).
Limit:	1 GB/mies.

Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ3
Nazwa:	Mailtrap [7]
Opis:	Środowisko testowe SMTP oraz Email API do wysyłki maili.
Limit:	150 maili/dzień

Tabela 3.3: Usługa zewnętrzna: Mailtrap

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ4

Nazwa:	LocationIQ [8]
Opis:	Geokodowanie adresu przy dodawaniu nowych spotów.
Limit:	5 000 zapytań/dzień

Tabela 3.4: Usługa zewnętrzna: LocationIQ

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ5
Nazwa:	Google Maps (Maps URLs) [9]
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)

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ6
Nazwa:	OpenFreeMap [10]
Opis:	Publiczny serwer kafelków do renderu mapy na froncie.
Limit:	30 000 zapytań/mies.

Tabela 3.6: Usługa zewnętrzna: OpenFreeMap

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ7

Nazwa:	Open-Meteo [11]
Opis:	Prognozy pogody wyświetlane dla spotów.
Limit:	10 000 zapytań/dzień

Tabela 3.7: Usługa zewnętrzna: Open-Meteo

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ8
Nazwa:	Tenor GIF API [12]
Opis:	Wyszukiwanie GIF-ów w czacie.
Limit:	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.8: Usługa zewnętrzna: Tenor GIF API

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ9
Nazwa:	Where the ISS at? [13]
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.

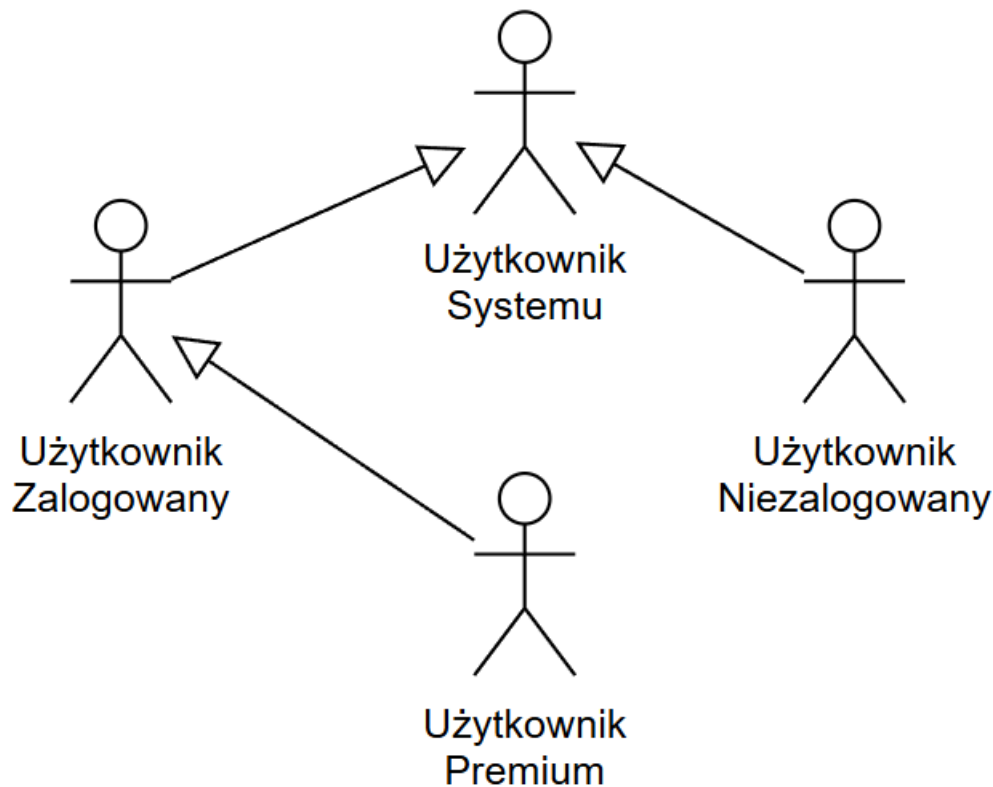
Usługa do przechowywania plików w chmurze - magazyn plików w chmurze służący do przechowywania załączników i multimediów użytkowników.

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.

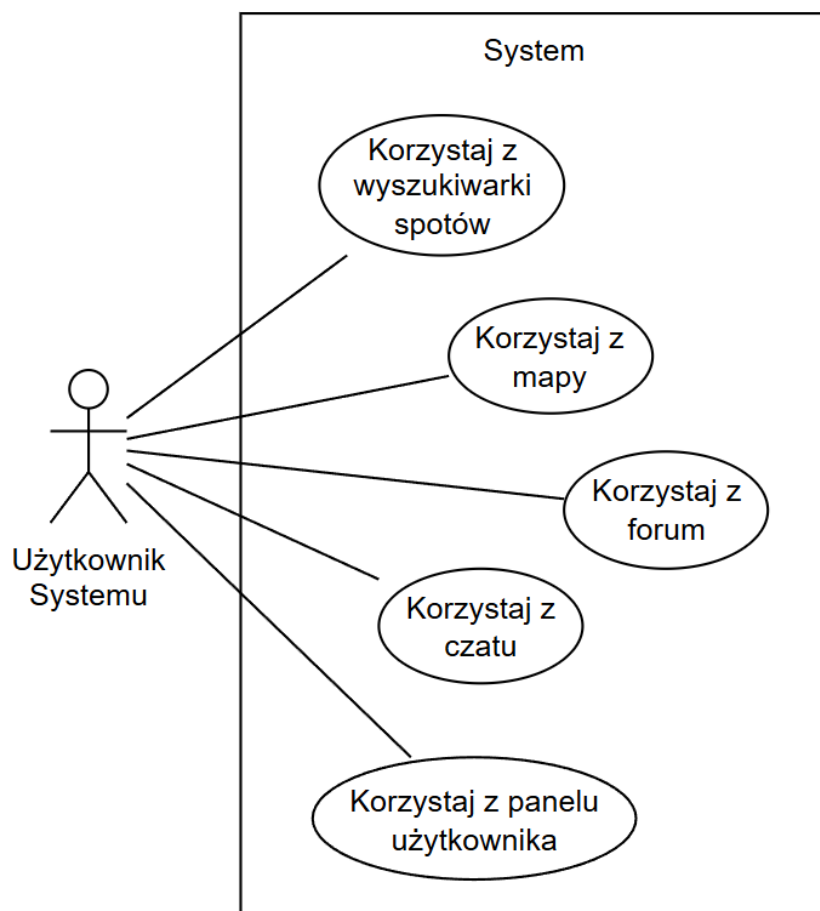


Rysunek 4.1: Diagram hierarchii użytkowników systemu

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). Użytkownik premium jest wyspecjalizowaną wersją użytkownika zalogowanego i oprócz standardowych możliwości ma także dostęp do opcji premium.

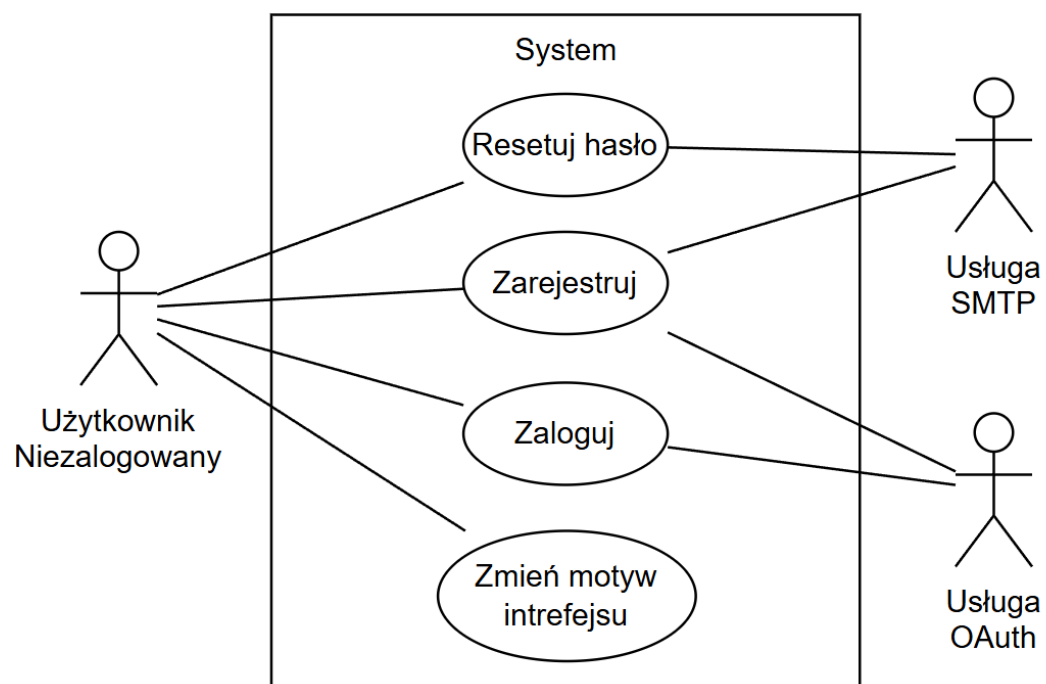
4.1.2 Diagramy przypadków użycia

Niniejszy rozdział przedstawia diagramy przypadków użycia.

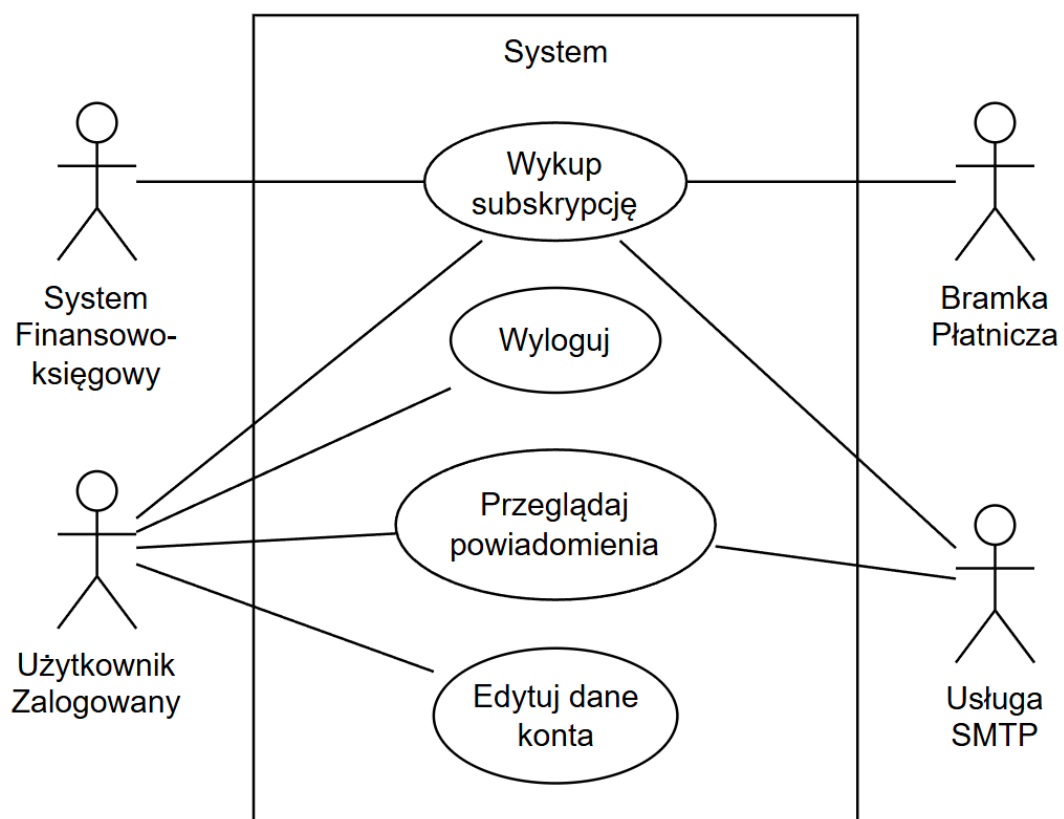


Rysunek 4.2: Wysokopoziomowy diagram przypadków użycia

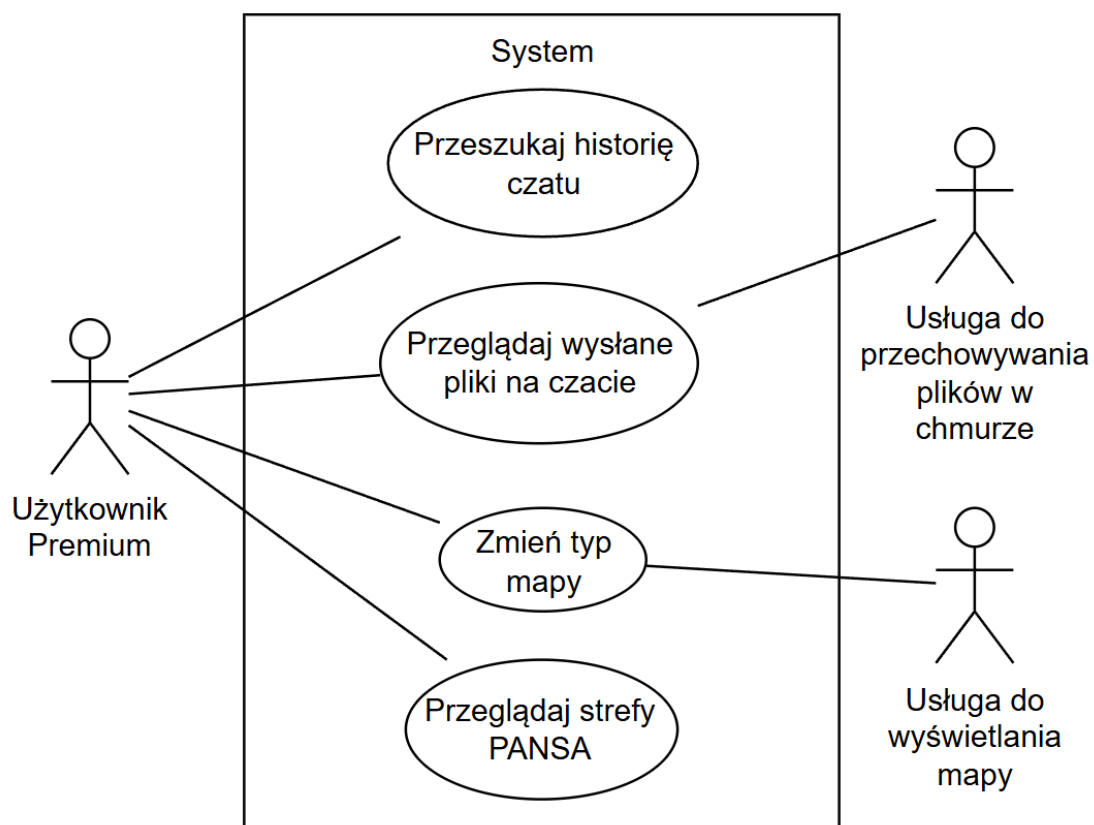
Diagram przedstawia podstawowe interakcje użytkownika z systemem. Na jego podstawie zespół projektowy podzielił architekturę aplikacji na 5 modułów: wyszukiwarkę spotów, mapę spotów, forum, czat oraz profil użytkownika. Pozostałe diagramy są bardziej szczegółowe.



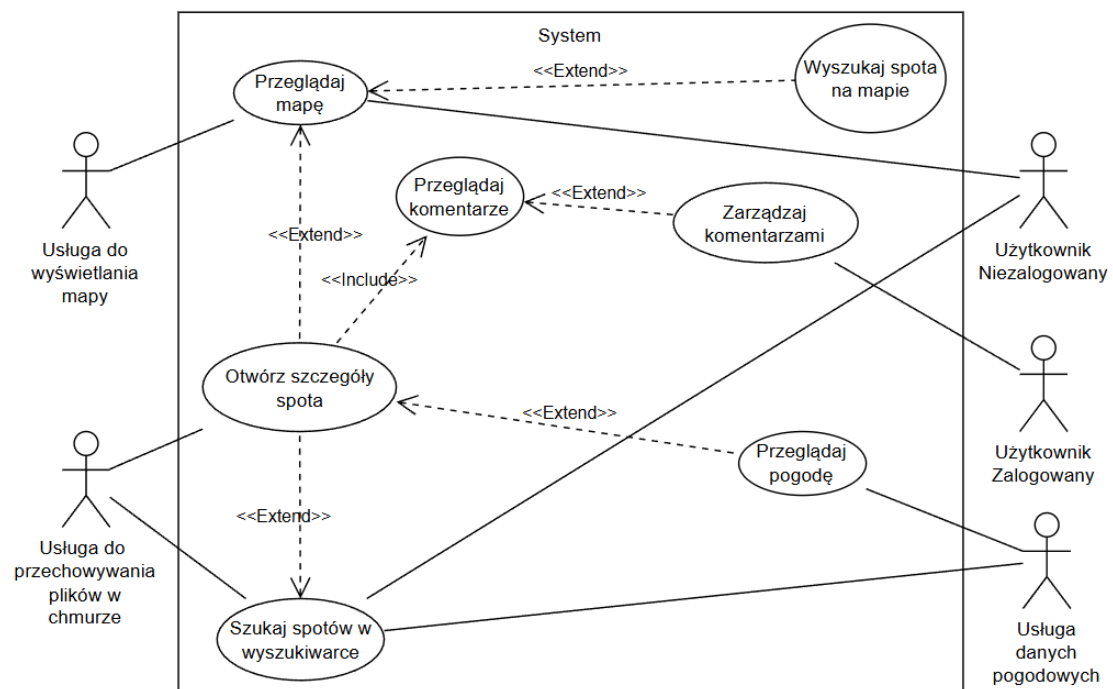
Rysunek 4.3: Diagram przypadków użycia dla użytkownika niezalogowanego



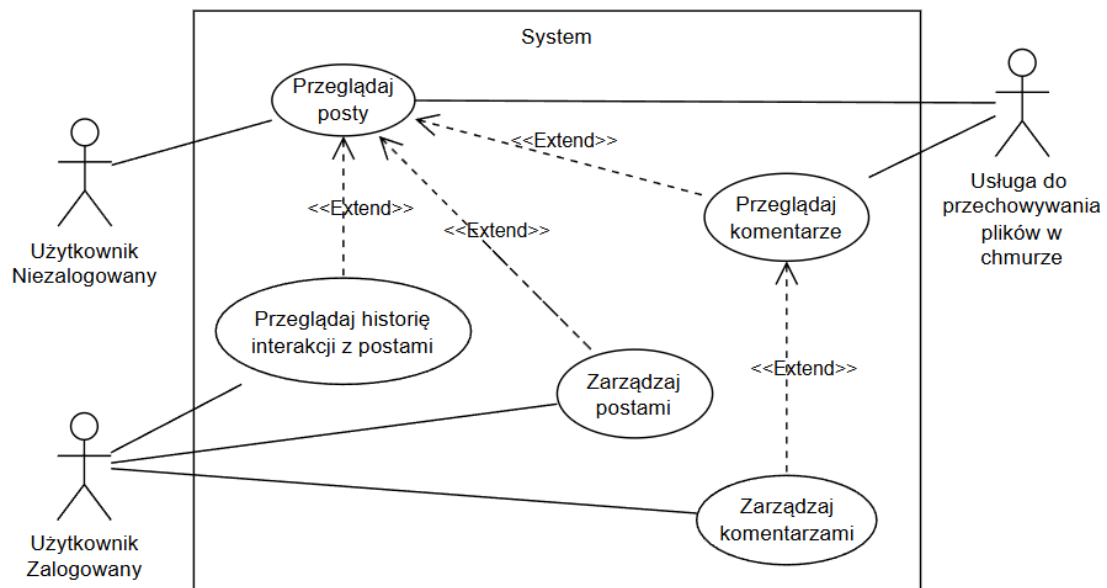
Rysunek 4.4: Diagram przypadków użycia dla użytkownika zalogowanego



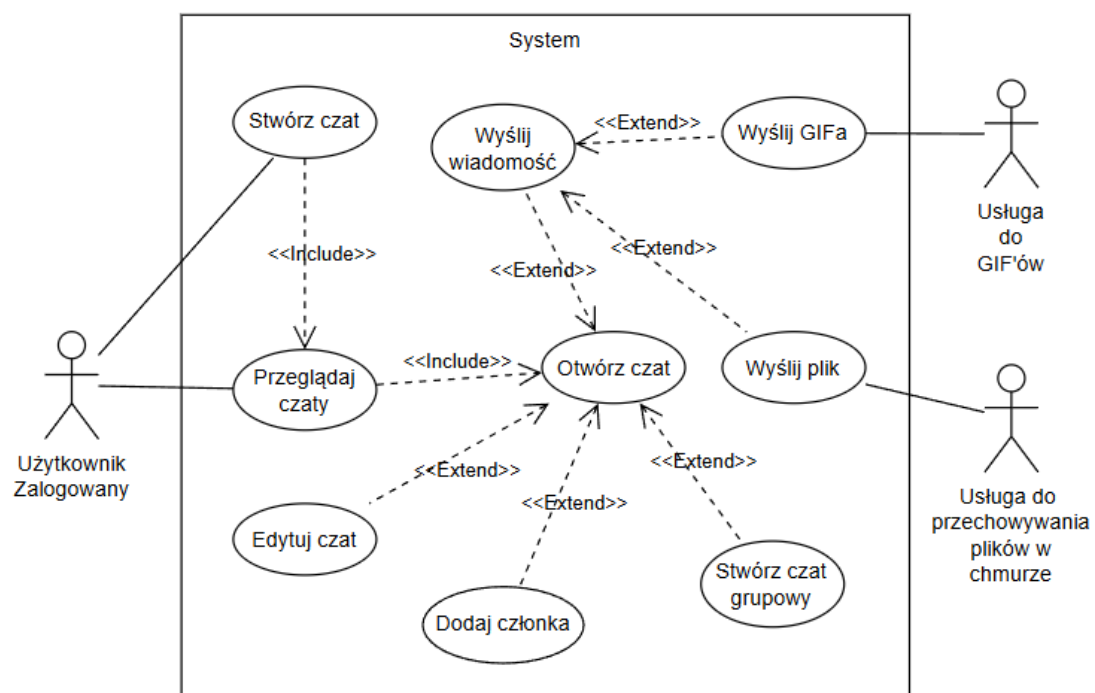
Rysunek 4.5: Diagram przypadków użycia dla użytkownika premium



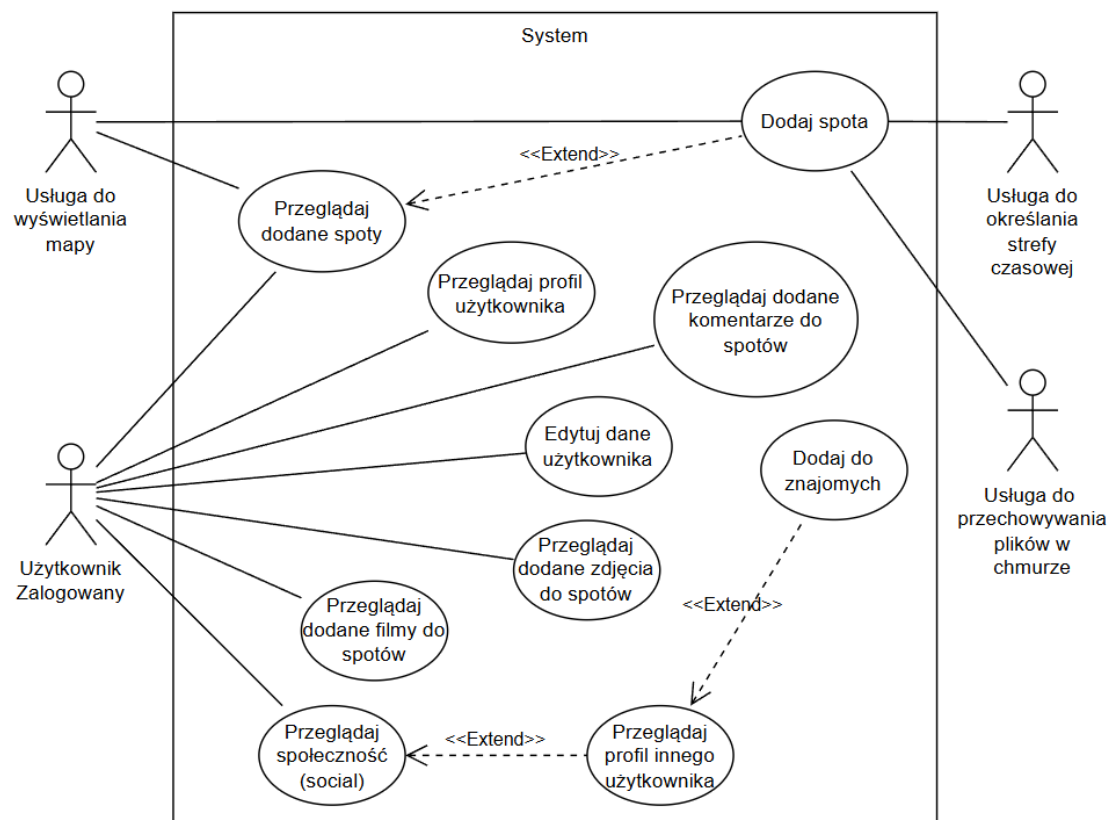
Rysunek 4.6: Diagram przypadków użycia wyszukiwarki spotów oraz mapy



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

W niniejszym rozdziale przedstawiono przebieg realizacji projektu w kolejnych Etapach zdefiniowanych w harmonogramie. Sposób realizacji projektu odzwierciedla model pracy zespołu zgodny z metodyką [Disciplined Agile Delivery - Lean Life Cycle](#), w której prace deweloperskie, planowanie i doprecyzowywanie wymagań przebiegają iteracyjnie i równolegle.

Warto podkreślić, że przez cały czas trwania projektu członkowie zespołu poświęcali znaczącą część czasu na wzajemne przeglądy kodu ([Review kodu](#)) oraz ciągły feedback dotyczący implementacji poszczególnych funkcjonalności. Taki sposób pracy pozwolił na szybkie wychwytywanie błędów, ujednolicenie stylu implementacji oraz bieżące korygowanie wymagań.

Dla przejrzystości opisu, w ramach każdego miesiąca przedstawiono zadania w podziale na poszczególnych członków zespołu.

6.1 Faza przedprojektowa (lipiec–wrzesień 2024)

Faza przedprojektowa, wyróżniona w harmonogramie jako okres od lipca do września 2024 roku, poprzedzała formalne zatwierdzenie tematu pracy oraz opracowanie szczegółowego harmonogramu.

Na początku fazy przedprojektowej doprecyzowano koncepcję systemu i wybrano główny stos technologiczny, co umożliwiło szybkie przejście do bardziej zaawansowanych działań opisanych w kolejnych podsekcjach. Niezależnie od ostatecznego sformułowania tematu zakładano stworzenie aplikacji webowej wymaga-

jącej kont użytkowników, co wynikało wprost ze specyfiki specjalizacji „Aplikacje Internetowe”, na której studiowali wszyscy członkowie zespołu.

Lipiec 2024

Cały zespół

- Określenie struktury plików na [backendzie](#).
- Wstępny wybór stosu technologicznego: [Backend](#) w [Spring Boot](#), [Frontend](#) w [React](#) z wykorzystaniem [TypeScript](#) oraz system kontroli wersji oparty na [GitHub](#), a także ogólne założenia dotyczące dalszego rozwoju architektury.

Adam Langmesser

- Przygotowanie lokalnego środowiska deweloperskiego dla [backendu](#) i [frontendu](#) (konfiguracja [IDE](#), testowe projekty, podstawowe ustawienia narzędzi budujących), co ułatwiło płynne przejście do właściwych prac w kolejnych miesiącach.

Sierpień 2024

Cały zespół

- Utworzenie projektu w narzędziu [Jira](#), zdefiniowanie typów zgłoszeń (epik, zadanie, podzadanie) oraz przygotowanie podstawowego workflow z wykorzystaniem [tablicy kanban](#).

Mateusz Redosz

- Konfiguracja biblioteki [Tailwind CSS](#) w części [frontendowej](#).
- Konfiguracja narzędzia [Prettier](#) do automatycznego formatowania kodu [frontendu](#).
- Implementacja rejestracji użytkownika zarówno na [backendzie](#) jak i [frontendzie](#).

- Dodanie i konfiguracja biblioteki [TanStack Query](#).
- Stworzenie wstępnej struktury [routingu](#) na [frontendzie](#).

Adam Langmesser

- Implementacja podstawowej strony powitalnej na [frontendzie](#).
- Implementacja podstaw logiki logowania i rejestracji użytkownika po stronie [backendu](#).
- Dodanie obsługi przypadku, w którym podczas rejestracji wybrana nazwa użytkownika jest już zajęta.

Kacper Badek

- Stworzenie formularza logowania użytkownika po stronie [frontendu](#) oraz jego integracja z przygotowanym [backendowym API](#) logowania.

Stanisław Oziemczuk

- Integracja systemu z zewnętrzną usługą wysyłania wiadomości e-mail oraz implementacja wysyłania wiadomości powitalnej do nowo zarejestrowanego użytkownika.
- Dodanie pliku konfiguracyjnego do formatowania kodu w [IntelliJ IDEA](#).

Wrzesień 2024

Adam Langmesser

- Dodanie [Docker Compose](#) dla [backendu](#) i bazy danych.
- Dodanie cyklicznego testu sprawdzającego, czy [backend](#) uruchamiany w środowisku [Docker Compose](#) działa poprawnie i odpowiada na podstawowe zapytania.

Mateusz Redosz

- Dodanie przycisków na [frontendzie](#) do rejestracji kontem Google lub [GitHub](#).
- Poprawa logiki obsługi [JWT](#).

Kacper Badek

- Implementacja funkcjonalności resetowania hasła użytkownika, obejmującej generowanie i weryfikację tokenów resetu oraz formularz zmiany hasła.

6.2 Etap 1 (październik 2024 – styczeń 2025)

Etap 1 był przede wszystkim poświęcony dopracowaniu wymagań wstępnych, stabilizacji modułu uwierzytelniania oraz pierwszym eksperymentom z mapą [spotów](#). W tym okresie zespół łączył prace deweloperskie z działaniami analitycznymi i planistycznymi (opracowanie harmonogramu, założeń oraz wymagań w ramach przedmiotów projektowych na uczelni).

Październik 2024

Cały zespół

- Formalny wybór tematu projektu inżynierskiego.

Adam Langmesser

- Zaprezentowanie zespołowi przykładowych [testów integracyjnych](#) oraz [testów E2E](#) na [backendzie](#).

Mateusz Redosz

- Poprawa konfiguracji [CORS](#) na [backendzie](#), tak aby aplikacja [frontendowa](#) mogła komunikować się z serwerem w sposób bezpieczny i zgodny z przeglądarkowymi ograniczeniami.
- Zmiana sposobu przechowywania tokena [JWT](#) – umieszczenie go w ciasteczku [Ciasteczko HttpOnly](#), co poprawiło bezpieczeństwo aplikacji.

Stanisław Oziemczuk

- Implementacja logowania [OAuth](#) z wykorzystaniem Google i [GitHub](#) po stronie [backendu](#) oraz integracja z [frontendem](#).
- Napisanie testów na [backendzie](#) do [OAuth](#).

Kacper Badek

- Uporządkowanie pliku `.gitignore` oraz struktury repozytorium.

Listopad 2024

Adam Langmesser

- Implementacja demonstracyjnej mapy z wykorzystaniem biblioteki [Leaflet](#) – prototyp miał na celu pokazanie zespołowi możliwości interaktywnej mapy.
- Poprawki konfiguracji narzędzia [ESLint](#) po stronie [frontendu](#).

Mateusz Redosz

- Dalsze poprawki obsługi [JWT](#) na [backendzie](#) oraz dodanie logowania błędów związanych z procesem logowania i rejestracji.
- Implementacja przykładowych [testów E2E](#) na [frontendzie](#).

Stanisław Oziemczuk

- Poprawa logiki logowania użytkownika po stronie [backendu](#) i [frontendu](#).

Kacper Badek

- Implementacja logiki cyklicznego usuwania przeterminowanych tokenów resetu hasła.

Cały zespół

- Opracowanie harmonogramu projektu w formie opisowej oraz w postaci wykresu Gantta.
- Przygotowanie wstępnych założeń i wymagań w ramach przedmiotu [PRO](#).

Grudzień 2024

Adam Langmesser

- Dalsze poprawki konfiguracji [Spring Security](#) na [backendzie](#), w tym doprecyzowanie ról i uprawnień.
- Dodanie nowych użytkowników do bazy danych, w celach deweloperskich.
- Implementacja testów automatycznych związanych z bezpieczeństwem (scenariusze logowania i rejestracji użytkownika po stronie [backendu](#)).

Mateusz Redosz

- Dodanie biblioteki [Redux](#) do [frontendu](#) i wstępna konfiguracja store.
- Implementacja automatycznego wylogowywania użytkownika po wygaśnięciu tokena [JWT](#).
- Stworzenie komponentu odpowiedzialnego za prezentację błędów systemowych użytkownikowi (globalny mechanizm powiadomień).

Stanisław Oziemczuk

- Stworzenie komponentu [frontendu](#) odpowiedzialnego za wyświetlanie szczegółów pojedynczego [spota](#).

Kacper Badek

- Poprawa logowania błędów związanych z resetowaniem hasła użytkownika po stronie [backendu](#).
- Dostosowanie wyglądu i treści wiadomości e-mail wysyłanych przez system (np. w procesie resetu hasła).

Styczeń 2025

Adam Langmesser

- Zaprezentowanie zespołowi działania [testów integracyjnych](#) oraz [testów E2E](#) na [backendzie](#).
- Poprawa testów logowania i rejestracji na [backendzie](#) oraz ujednolicenie asercji.
- Rozwiązanie problemu z relacjami między obiektami w testach integracyjnych.

Mateusz Redosz

- Rozszerzenie stanu [Redux](#) o informację o tym, czy użytkownik jest aktualnie zalogowany.
- Implementacja testów dla procesów logowania i rejestracji użytkownika.
- Implementacja [testów integracyjnych](#) i [testów jednostkowych](#) dla [sidebara](#).
- Dodanie uruchamiania testów [frontendu](#) do [CI/CD](#).
- Poprawa sposobu wyświetlania szczegółów [spota](#).

Stanisław Oziemczuk

- Poprawki logowania i rejestracji z wykorzystaniem [OAuth](#) (Google/[GitHub](#)) – dopracowanie scenariuszy brzegowych.
- Implementacja logiki filtrowania [spotów](#) na mapie po różnych kryteriach (np. nazwa) po stronie [backendu](#).
- Dalsze dopracowanie logiki filtrowania [spotów](#) po nazwie po stronie [backendu](#).
- Implementacja logiki zmiany danych konta użytkownika zarówno na [backendzie](#) jak i na [frontendzie](#).

Kacper Badek

- Dodanie danych deweloperskich dla mapy (przykładowe [spotify](#), dane do prezentacji i testów).
- Implementacja logiki dodawania [spota](#) do ulubionych po stronie [backendu](#).

6.3 Etap 2 (luty 2025 – wrzesień 2025)

Etap 2 obejmował zasadniczą część prac deweloperskich nad aplikacją. W tym okresie rozwijano kolejne moduły (mapa, forum, czat, panel użytkownika), równolegle doprecyzowując dokumentację oraz weryfikując i aktualizując wymagania na podstawie bieżących doświadczeń zespołu.

Równolegle, w ramach przedmiotu [PRZ 1](#) opracowano projekt interfejsu użytkownika, konsultowany z prowadzącym zajęcia, mgr. inż. Adamem Urbanowiczem, który na bieżąco przekazywał zespołowi uwagi i rekomendacje dotyczące ergonomii oraz spójności interfejsu z założeniami funkcjonalnymi.

Luty 2025

Mateusz Redosz

- Poprawa wyglądu strony logowania, w tym dopracowanie trybu jasnego tejże strony.
- Implementacja automatycznego wylogowania użytkownika.
- Implementacja obsługi błędów [jakarta validation](#).
- Refactor wyglądu komponentu pogodowego na [frontendzie](#).
- Poprawa działania [CI/CD](#).
- Wypracowanie propozycji palety kolorystycznej interfejsu użytkownika.
- Implementacja integracji z zewnętrznym [API](#) pogodowym służącym do pobierania podstawowych danych pogodowych dla danego [spota](#).

Stanisław Oziemczuk

- Implementacja logiki dodawania [spota](#) do ulubionych po stronie [frontendu](#) oraz integracja z [backendem](#).
- Drobną refaktoryzację kodu na [frontendzie](#).
- Obsługa błędów przy wysyłaniu maili.
- Zmiana dostawy usługi do wysyłania maili.
- Optymalizacja zapytań do bazy danych.
- [Testy integracyjne](#) logiki logowania i rejestracji [OAuth](#) na [backendzie](#) oraz filtrowania [spotów](#).
- Poprawa plików [Dockerfile](#) oraz [Docker Compose](#).
- Przygotowanie skryptu uruchamiającego wszystkie wymagane kontenery [Docker](#) (bazy danych i usługi pomocnicze) na potrzeby środowiska deweloperskiego, co uprościło proces uruchamiania projektu na nowych stanowiskach.

Adam Langmesser

- Dalsze poprawki konfiguracji [Spring Security](#) na [backendzie](#), obejmujące doprecyzowanie reguł autoryzacji i konfiguracji filtrów.
- Poprawa konfiguracji pliku `.gitignore` w repozytorium, tak aby obejmował również nowe katalogi i pliki generowane przez narzędzia deweloperskie.
- Poprawa działania [CI/CD](#) dla [backendu](#).
- Poprawa struktury plików na [backendzie](#).
- Dodanie cachowania na [backendzie](#) w postaci [Redis](#).
- Poprawa konfiguracji poziomów logowania o błędach na [backendzie](#).

Marzec 2025

Cały Zespół

- Wypracowanie wstępnej wersji wyglądu interfejsu użytkownika w ramach [PRZ 1](#).

Kacper Badek

- Refaktoryzacja szablonów wiadomości e-mail związanych z rejestracją i resetowaniem hasła – ujednolicenie struktury HTML, metadanych oraz stylu logo aplikacji.
- Dodanie możliwości edycji, usuwania i głosowania na komentarze [spota](#) oraz lepsza integracja z widokiem szczegółów [spota](#) ([paginacja](#), powiadomienia, prezentacja ocen).

Stanisław Oziemczuk

- Przygotowanie instrukcji uruchamiania projektu w README.

Mateusz Redosz

- Migracja wszystkich adresów obrazów (galerie zdjęć [spotów](#) oraz logotypy w szablonach e-mail) z usługi Google Drive do [CDN](#) [ucarecdn.com](#).

Kwiecień 2025

Cały Zespół

- Wypracowanie finalnej wersji wyglądu interfejsu użytkownika w ramach [PRZ 1](#).

Adam Langmesser

- Konfiguracja projektu dokumentacji w [LaTeX](#), utworzenie pliku bibliografii oraz głównego dokumentu z przykładową strukturą rozdziałów.

- Implementacja podstaw funkcjonalności czatu w warstwie [backendowej](#) i [frontendowej](#), a także dostosowanie layoutu i paska bocznego do nowej sekcji.

Mateusz Redosz

- Integracja [frontendu](#) z [TypeScript](#).
- Integracja [Tailwind CSS](#) z projektem [frontendu](#) oraz dodanie predefiniowanych kolorów zgodnych z nową szatą kolorystyczną interfejsu użytkownika.
- Poprawa i rozbudowa [sidebara](#).
- Implementacja strony profilu użytkownika.

Maj 2025

Stanisław Oziemczuk

- Migracja mapy z biblioteki [Leaflet](#) na [React-MapLibre](#), oraz implementacja od nowa podstaw wyświetlania [spotów](#) i interakcji użytkownika z mapą.
- Zmiana wyglądu znacznika lokalizacji użytkownika na mapie.
- Migracja funkcji [API](#) związanych ze [spotami](#) do [TypeScript](#), rozszerzenie modelu szczegółów [spota](#) o dodatkowe informacje lokalizacyjne i statystyczne (np. kraj, miasto, tagi, liczniki ocen) oraz refaktoryzacja logiki interakcji z markerami [React-MapLibre](#).

Mateusz Redosz

- Refactor panelu bocznego, dodanie ogólnych hooków do zarządzania stanem (w tym trybem ciemnym) oraz wprowadzenie zależności do biblioteki [motion](#) na potrzeby animacji.
- Dodanie sekcji „*Znajomi*” w panelu konta, obejmującej obsługę znajomych, obserwowanych i obserwujących, wprowadzenie nowych modeli i [endpointów](#) do zarządzania relacjami oraz dodanie testów jednostkowych.

- Refaktoryzacja [endpointów](#) użytkownika tak, aby login był pobierany z kontekstu uwierzytelnienia zamiast z parametru ścieżki, dostosowanie do tego wywołań na froncie oraz przeniesienie [slice Redux](#) odpowiedzialnego za konto użytkownika na [TypeScript](#) z uproszczonym stanem i uporządkowanymi importami.
- Poprawa wyglądu formularzy logowania i rejestracji.
- Poprawa zarządzania stanem [sidebara](#).

Adam Langmesser

- Wprowadzenie usprawnień w potokach [CI/CD backendu](#), obejmujących optymalizację czasu budowania oraz doprecyzowanie kroków uruchamiania testów, co zwiększyło niezawodność procesu wdrażania.

Kacper Badek

- Dodanie pełnoprawnego forum po stronie [frontendu](#), obejmującego obsługę postów, kategorii, tagów i [paginacji](#), oraz integracja [backendu](#) z usługą [Azure Blob Storage](#) do uploadu i przechowywania mediów.

Czerwiec 2025

Mateusz Redosz

- Przebudowa profilu użytkownika poprzez rozdzielenie widoków własnego profilu i profili innych użytkowników, dodanie przycisków akcji (follow/unfollow, zaproszenie do znajomych), usprawnienie nawigacji z kart znajomych i [sidebara](#) oraz aktualizacja testów.
- Dodanie sekcji „*Ulubione [spoty](#)*” w panelu konta (lista, filtrowanie, usuwanie oraz podgląd na mapie), wprowadzenie współdzielonych komponentów i modeli dla tej funkcji oraz uporządkowanie przestarzałego kodu i modeli współrzędnych.

- Przebudowa funkcjonalności społecznościowej: rozdzielenie widoków sekcji *social* dla właściciela profilu i osoby oglądającej, aktualizacja modelu danych i routingu oraz dodanie nawigacji z liczników profilu do odpowiednich list (friends, followers, followed), wraz z wprowadzeniem osobnego [slice Redux](#) do zarządzania aktywnym typem listy social.

Stanisław Oziemczuk

- Refaktoryzacja komentarzy do [spotów](#) – zastąpienie ogólnego modelu szczegółowym, spot-specyficznym typowaniem i nowymi komponentami UI, uporządkowanie warstwy [API](#) oraz poprawa wyglądu i układu sekcji komentarzy.

Lipiec 2025

Mateusz Redosz

- Dodanie nowej sekcji „*Zdjęcia*” w panelu użytkownika, z możliwością sortowania i filtrowania po dacie, aktualizacja routingu i stylu oraz dopisanie testów.
- Dodanie sekcji komentarzy w panelu konta (z grupowaniem po dacie i [spocie](#) oraz filtrowaniem/sortowaniem po dacie).
- Dodanie strony ustawień konta umożliwiającej edycję nazwy użytkownika, adresu e-mail i hasła. Aktualizacja routingu i modeli (nowe enumy/interfejsy, wariant przycisku), usunięcie starych [endpointów](#) edycji danych oraz dodanie zależności potrzebnych do obsługi formularzy.
- Ujednolicenie obsługi zdjęć i filmów do wspólnego modelu *media*, dodanie nowej sekcji „*Filmy*” w panelu użytkownika (routing, [endpointy](#), komponenty UI) oraz refaktoryzacja istniejących widoków i DTO tak, aby korzystały z nowych, współdzielonych struktur.

Adam Langmesser

- Dodanie pobierania danych czatu i wysyłania wiadomości po [WebSocket](#).

- Uporządkowanie formatowania kodu z wykorzystaniem narzędzia [Prettier](#), dodanie skryptu `format:check` oraz sprawdzania formatowania w potokach [CI/CD](#).
- Ujednolicenie modelu danych czatu, uproszczenie logiki komunikacji z [API](#) i struktury store'a [Redux](#), dostosowanie komponentów czatu do nowego modelu oraz poprawa wyglądu czatu.
- Poprawa danych deweloperskich w bazie (m.in. przykładowych [spotów](#) i użytkowników).
- Usprawnienie mechanizmu nieskończonego przewijania i nazewnictwa listy czatów.
- Stworzenie ogólnej logiki i komponentów pomocniczych umożliwiających wszystkim członkom zespołu wygodne korzystanie z [websocketów](#) na froncie.
- Dodanie grupowania wiadomości na czacie po ich dacie wysłania, implementacja wielowierszowego pola tekstowego.
- Wprowadzenie drobnych poprawek w potokach [CI/CD](#).

Stanisław Oziemczuk

- Zastąpienie dotychczasowych filtrów wyszukiwania [spotów](#) na mapie nowym paskiem wyszukiwania po nazwie z bocznym panelem wyników i [paginacją](#) oraz dopracowanie interfejsu (gwiazdki ocen) i logiki cache'owania zapytań.
- Ujednolicenie obsługi mediów dla [spotów](#) i komentarzy oraz implementacja wyświetlania zdjęć i filmów dla [spotów](#).
- Refaktoryzacja struktury plików na [backendzie](#) związanych z modułem mapy.
- Dodanie możliwości wyszukiwania [spotów](#) na mapie w obecnie widocznym dla użytkownika obszarze.
- Dodanie zdjęć i filmów do [spotów](#) w celach deweloperskich.

Kacper Badek

- Zapoznanie się z dokumentacją [TinyMCE Rich text editor](#).

Sierpień 2025

Adam Langmesser

- Dodanie do czatu możliwości wyszukiwania i wysyłania animowanych obrazów ([gifów](#)) oraz wysyłania [emoji](#).
- Usprawnienie czatu poprzez wprowadzenie optymistycznego wysyłania wiadomości, dodanie nowych [endpointów](#) do stronicowanego pobierania starszych wiadomości.

Stanisław Oziemczuk

- Dodanie jednoznacznego „punktu środka” [spota](#) i konsekwentne wykorzystywanie go w [backendzie](#) i [frontendzie](#).

Mateusz Redosz

- Dodanie na stronie głównej karuzeli z najpopularniejszymi [spotami](#) oraz wyszukiwarki [spotów](#) po lokalizacji, z listą wyników i dystansem od użytkownika.
- Dodanie zaawansowanego wyszukiwania [spotów](#) na stronie głównej.
- Wprowadzenie [paginacji](#) do [endpointów](#) panelu użytkownika i przebudowa powiązanych komponentów [frontendu](#) na nieskończone przewijanie.
- Dodanie sekcji „*Zdjęcia*” w panelu społecznościowym użytkownika oraz uproszczenie logiki nieskończonego przewijania dla różnych zakładek social.
- Dodanie do panelu użytkownika funkcji dodawania własnych [spotów](#) oraz widoku listy dodanych [spotów](#).

Kacper Badek

- Przygotowanie formularza dodawania postów z wykorzystaniem edytora [TinyMCE](#).
- Skonfigurowanie biblioteki [jsoup](#) na [backendzie](#).

Wrzesień 2025

Adam Langmesser

- Wprowadzenie możliwości rozpoczynania lub kontynuowania prywatnych rozmów czatowych bezpośrednio z list znajomych i obserwujących w zakładce „Social”.
- Dodanie obsługi [emoji](#) na czacie oraz poprawa wyglądu okna do wysyłania [gifów](#).

Mateusz Redosz

- Refaktoryzacja systemu informacji o błędach i sukcesach na [frontendzie](#), umożliwiającą jednoczesne wyświetlanie wielu komunikatów oraz zwiększającą modularność i możliwość ponownego wykorzystania komponentów.
- Wprowadzenie [paginacji](#) dla wyszukiwarki [spotów](#).
- Dodanie walidacji formularza dodawania [spota](#), a także wyświetlanie komunikatów błędów.
- Rozszerzenie komponentu przesyłania multimediów o podgląd wybranych obrazów i materiałów wideo.
- Wprowadzenie obsługi zmiany zdjęcia profilowego użytkownika.
- Refaktoryzacja struktury projektu części opisowej pracy inżynierskiej – zastąpienie dotychczasowej treści demonstracyjnej rzeczywistymi rozdziałami opisującymi projekt.
- Rozszerzenie zaawansowanych możliwości wyszukiwania [spotów](#) o sortowanie wyników według popularności lub oceny oraz filtrowanie po minimalnej ocenie.

Stanisław Oziemczuk

- Implementacja kompletnej funkcjonalności pogody dla [spotów](#): dodanie podstawowego i szczegółowego modalu pogodowego ([Modal](#)) oraz zestawu komponentów interfejsu prezentujących m.in. temperaturę, prędkość wiatru, opady oraz dodatkowe parametry meteorologiczne.
- Przebudowa obsługi pogody tak, aby zamiast bezpośrednich wywołań publicznego [API](#) wykorzystywany był [backend](#) jako warstwa pośrednia.
- Implementacja wyświetlania zdjęcia profilowego użytkownika w komentarzach [spota](#).

Kacper Badek

- Zamiana edytora [TinyMCE](#) na [Tiptap](#).
- Implementacja przeglądania postów na forum oraz ich sortowania.
- Wprowadzenie czytelniejszych adresów [URL](#) opartych na [Slugach](#), poprawa walidacji dodawania nowego posta.
- Poprawa ergonomii poruszania się po forum.

6.4 Etap 3 (październik 2025 – styczeń 2026)

Etap 3 obejmował finalizację prac nad systemem, dopracowanie dokumentacji technicznej i tekstowej pracy inżynierskiej oraz przygotowanie projektu do oddania. W tym czasie większość nowych funkcjonalności była już zaimplementowana, a nacisk położono na stabilizację, testy oraz spójny opis w dokumentacji.

W ramach przedmiotu [PSEM](#), prowadzonego przez dr. hab. Marka Bednarczyka, postępy w przygotowywaniu dokumentacji były na bieżąco konsultowane, a na podstawie uzyskiwanej informacji zwrotnej wprowadzano kolejne poprawki i uzupełnienia. Analogiczny tryb pracy przyjęto w ramach przedmiotu [PRZ 2](#), którego opiekunem był promotor pracy, mgr. inż. Adam Urbanowicz.

Październik 2025

Adam Langmesser

- Wprowadzenie możliwości tworzenia czatów grupowych, w tym obsługi wyboru uczestników, komunikacji z [backendem](#) oraz integracji z istniejącą listą czatów.
- Dodanie funkcjonalności wysyłania i wyświetlania załączników w wiadomościach czatu (pliki oraz obrazy), wraz z logiką wyboru, podglądu i wysyłania samych plików bez treści tekstowej.
- Rozszerzenie istniejącej funkcjonalności czatów grupowych o możliwość edycji nazwy oraz obrazu czatu po stronie [backendu](#) i [frontendu](#).
- Dodanie możliwości dołączania nowych użytkowników do istniejących czatów grupowych.

Mateusz Redosz

- Rozbudowa systemu statusów znajomych w części społecznościowej aplikacji, w tym rozróżnienie zaproszeń wysłanych, otrzymanych oraz relacji zakończonych, a także dostosowanie interfejsu do prezentacji odpowiednich komunikatów i akcji.
- Wprowadzenie zaawansowanego zarządzania zaproszeniami do znajomych: dodanie modalnego widoku listy zaproszeń, obsługi akceptowania i odrzucania oraz integracji z dedykowanymi [endpointami backendowymi](#).
- Dodanie funkcji „*Dodaj znajomego*” w sekcji społecznościowej, obejmującej wyszukiwarke użytkowników ([paginacja](#), wyszukiwanie po nazwie użytkownika) oraz spójny wygląd modalnego okna wyszukiwania.
- Rozbudowa komponentu przycisku przesyłania plików o możliwość podglądu wielu plików, nadawania im unikalnych identyfikatorów oraz usuwania pojedynczych plików przed wysłaniem.

- Przygotowanie struktury rozdziałów w [latexie](#).
- Opracowanie rozdziału [10.2.2 Mateusz Redosz](#).

Stanisław Oziemczuk

- Wprowadzenie rozszerzonej galerii multimediiów dla [spotów](#), obejmującej obsługę [paginacji](#), podglądu w trybie pełnoekranowym oraz dodatkowych akcji (np. udostępnianie odnośnika do zasobu).
- Dostosowanie modułu mapy oraz widoku szczegółów [spotów](#) pod kątem responsywności i skalowania na dużych ekranach.
- Poprawa responsywności panelu pogodowego.

Kacper Badek

- Dodanie obsługi komentarzy do postów (dodawanie, edycję, usuwanie oraz głosowanie).
- Zastąpienie klasycznych wskaźników ładowania loaderami typu [Skeleton loader](#) dla listy postów oraz paneli kategorii i tagów.
- Poprawa działania stanu formularza dodawania postów.
- Dodanie możliwości zgłaszania postów i komentarzy.
- Implementacja możliwości obserwowania postów.

Listopad 2025

TODO: do uzupełnienia

Grudzień 2025

TODO: do uzupełnienia

Styczeń 2026

TODO: do uzupełnienia

Na zakończenie prac projektowych przyjęto datę **10 stycznia 2026 roku**.

Rozdział 7

Realizacja Projektu

7.1 Implementacja backendu

7.1.1 Struktura projektu

7.1.2 Integracja z bazą danych

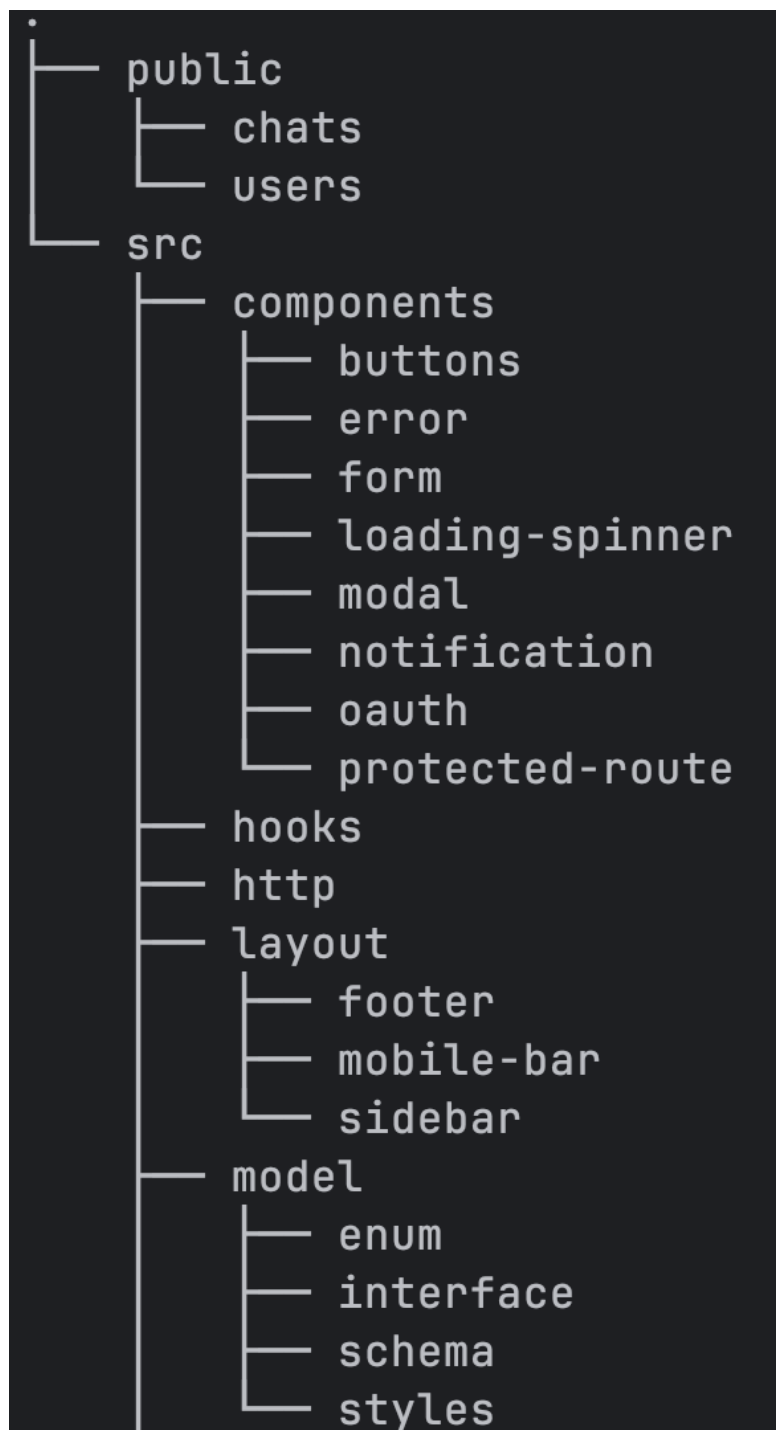
7.1.3 Obsługa uwierzytelnienia

7.1.4 Konteneryzacja

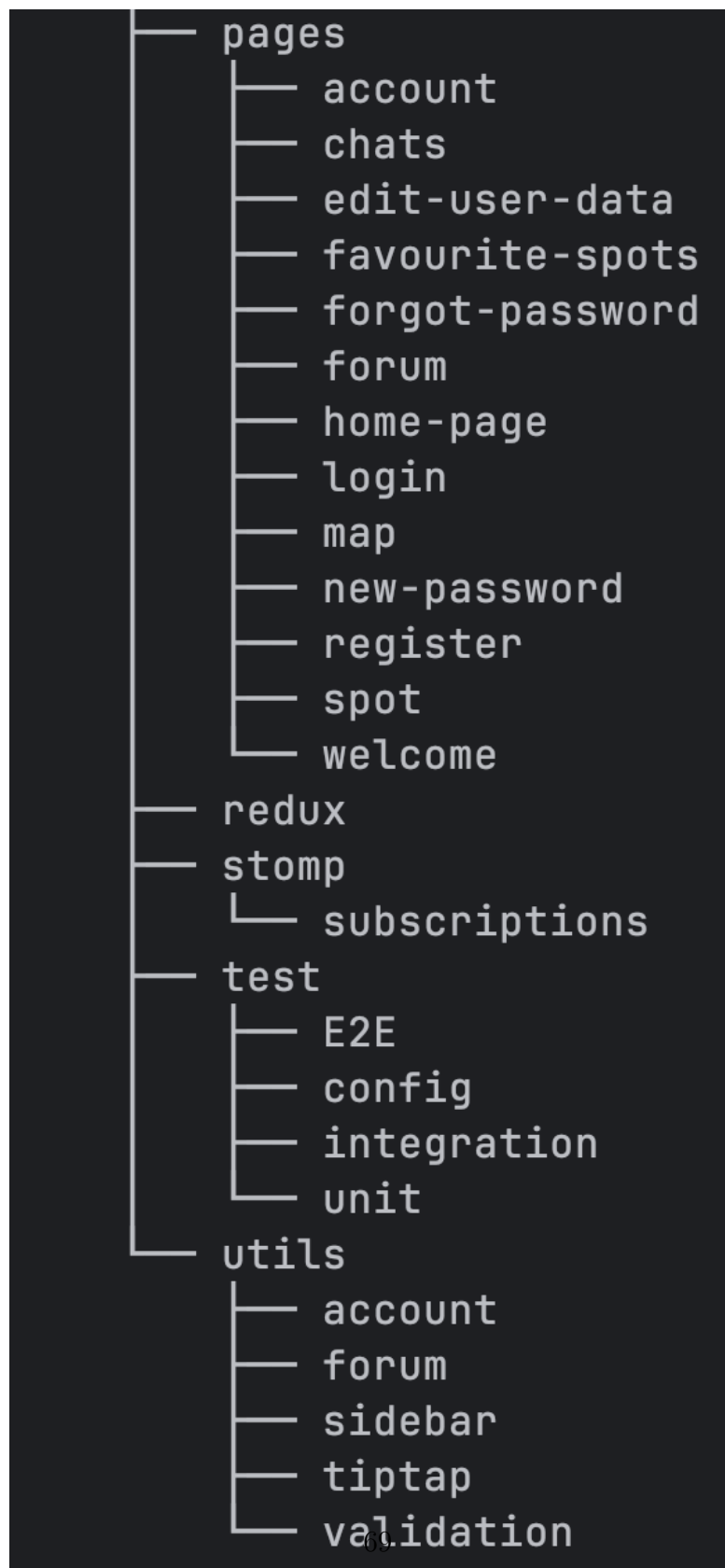
7.2 Implementacja frontendu

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`) [14], jak i ze [Stanu](#) globalnego, utrzymywanego przez [Bibliotekę React Redux](#) [15]. 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](#) [16] oraz [Biblioteki TanStack Query](#) [17]. 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 [18]. 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](#) [19]. 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 wyszukiwarke, 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](#), [16](#), [50](#), [55](#), [58](#), [60](#), [61](#), [64](#), [99](#)

Azure Blob Storage

Usługa magazynu obiektowego w chmurze Microsoft Azure do przechowywania nieustrukturyzowanych danych (*blobs*) takich jak obrazy, wideo i pliki. Udostępnia kontenery, warstwy dostępu, wersjonowanie oraz tokeny SAS; często używana do hostowania multimediów w aplikacjach webowych.. [59](#)

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](#), [14](#), [49–56](#), [58](#), [59](#), [61](#), [62](#), [64](#), [65](#), [92](#), [98](#)

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. [70](#), [72](#), [74](#), [76](#), [79](#), [80](#), [85](#)

BPMN

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

CDN

Skrót od *Content Delivery Network*. Rozproszona sieć serwerów służąca do szybkiego dostarczania statycznych zasobów (np. obrazów, arkuszy CSS, skryptów JavaScript) z węzłów geograficznie najbliższych użytkownikowi, co zmniejsza opóźnienia i obciąża serwer aplikacji. [57](#)

CI/CD

Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. [15](#), [54–56](#), [59](#), [61](#), [91](#), [94](#)

Ciasteczko HttpOnly

Ciasteczko HTTP ustawione z flagą `HttpOnly`, dzięki czemu nie jest dostępne z poziomu JavaScriptu. Zmniejsza ryzyko kradzieży tokenów (np. JWT) w przypadku ataków typu XSS. [51](#)

CORS

Skrót od *Cross-Origin Resource Sharing*. Mechanizm bezpieczeństwa w przeglądarkach, który kontroluje, czy aplikacja z jednej domeny może wykonywać zapytania HTTP do serwera w innej domenie; konfigurowany za pomocą nagłówków HTTP. [51](#)

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.. [79](#), [80](#)

Design

Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). [91](#)

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. . [10](#), [48](#)

Docker

Platforma do konteneryzacji aplikacji. Pozwala uruchamiać oprogramowanie w lekkich, izolowanych kontenerach tworzonych na podstawie obrazów, co upraszcza wdrażanie i utrzymanie spójnego środowiska. [56](#), [98](#)

Docker Compose

Narzędzie do definiowania i uruchamiania wielokontenerowych aplikacji [Docker](#) za pomocą pliku konfiguracyjnego (np. `docker-compose.yml`). Umożliwia jednoczesne uruchamianie powiązanych usług (np. [Backend](#), baza danych, usługi pomocnicze) jednym poleceniem. [50](#), [56](#)

Dockerfile

Plik tekstowy zawierający instrukcje opisujące, jak zbudować obraz Dockera (jakiej podstawy użyć, jakie pliki skopiować, jakie polecenia uruchomić). Na jego podstawie narzędzie Docker tworzy gotowy obraz kontenera. [56](#)

Droniarz

Potoczne określenie osoby, która jest jednocześnie pilotem oraz operatorem drona. Zwykle entuzjasta dronów.. [8](#), [9](#), [110](#)

Droniarz foto/video

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

emoji

Małe graficzne ikonki używane do wyrażania emocji lub pojęć w komunikacji cyfrowej (np. uśmiechnięta buźka, kciuk w górę, symbol serca).. [62](#), [63](#)

endpoint

Endpoint to konkretny adres (np. [URL](#)) i metoda protokołu HTTP w [API](#), które razem odpowiadają za realizację jednej, dobrze zdefiniowanej operacji (np. pobrania listy spotów, dodania komentarza, wyszukania spotów).. [58](#)–[60](#), [62](#), [65](#)

ESLint

Statyczny analizator kodu JavaScript/TypeScript. Umożliwia wykrywanie błędów, niespójności stylu oraz potencjalnych problemów poprzez zestaw reguł, które można dostosować do projektu. [52](#)

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. [67](#)

Framework

Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. [2](#), [77](#)

Frontend

Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. [2](#), [14](#), [49–56](#), [58](#), [59](#), [62](#), [63](#), [65](#), [92](#)

GIF

Format graficzny *Graphics Interchange Format* obsługujący krótkie, zapętlone animacje. W aplikacjach czatowych wykorzystywany do wysyłania „reakcji” w postaci ruchomych obrazków. [62](#), [63](#)

GitHub

Platforma hostingu repozytoriów *Git* w chmurze, oferująca m.in. pull requesty, system zgłoszeń (issues), zarządzanie wersjami oraz integrację z narzędziami CI/CD. [49](#), [51](#), [52](#), [54](#)

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`).. [72](#), [76](#)

IDE

(ang. *integrated development environment*); to zintegrowane środowisko programistyczne, służące do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. [14](#), [49](#)

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. [86](#)

IntelliJ IDEA

Zintegrowane środowisko programistyczne (IDE) firmy JetBrains, szeroko stosowane przy tworzeniu aplikacji backendowych w ekosystemie Spring. Oferuje m.in. podpowiedzi składni, refaktoryzację kodu, debugger oraz integrację z systemami kontroli wersji. [50](#)

jakarta validation

Jakarta Validation to specyfikacja (i zestaw adnotacji, typu @NotNull, @Size itd.) służąca do automatycznego sprawdzania poprawności danych w aplikacjach stworzonych za pomocą Java/Jakarta EE/Spring, np. przy walidacji pól DTO, encji czy parametrów metod.. [55](#)

Jira

Narzędzie firmy Atlassian do zarządzania projektami i zadaniami, szeroko stosowane w metodykach zwinnych. Umożliwia pracę z epikami, taskami, podtaskami oraz tablicami Scrum i Kanban. [49](#)

jsoup

Biblioteka *Java* do przetwarzania dokumentów HTML, umożliwiająca parsowanie, przeszukiwanie i modyfikowanie struktury dokumentu w sposób zbliżony do pracy z DOM-em i selektorami CSS.. [62](#)

JWT

Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. [51–53](#), [74](#), [91](#)

LaTeX

System składu tekstu wykorzystywany do przygotowywania profesjonalnych dokumentów technicznych i naukowych. Umożliwia precyzyjne formatowanie, zarządzanie odwołaniami, bibliografią i wzorami matematycznymi. [57](#), [66](#)

Leaflet

Lekka biblioteka JavaScript do tworzenia interaktywnych map w przeglądarce, często używana z danymi z OpenStreetMap. Umożliwia dodawanie znaczników, warstw oraz obsługę interakcji użytkownika. [52](#), [58](#)

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*).. [104](#)

Modal

Okno dialogowe (okno modalne), które pojawia się na wierzchu interfejsu i blokuje interakcję z resztą aplikacji, dopóki użytkownik go nie zamknie. Służy do prezentowania ważnych komunikatów lub formularzy. [64](#)

OAuth

Standard autoryzacji umożliwiający aplikacjom zewnętrznym uzyskanie dostępu do zasobów użytkownika bez przekazywania jego hasła, często wykorzystywany przy logowaniu za pomocą dostawców takich jak Google czy GitHub. [52](#), [54](#), [56](#)

paginacja

Mechanizm dzielenia dużych zbiorów danych (np. list postów, wyników wyszukiwania, komentarzy) na mniejsze strony, które są pobierane i wyświetlane stopniowo, zamiast ładowania wszystkich elementów jednocześnie.. [57](#), [59](#), [61–63](#), [65](#), [66](#)

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. . [20](#)

Prettier

Narzędzie do automatycznego formatowania kodu (np. JavaScript, TypeScript, CSS, HTML). Narzuca spójny styl formatowania, zastępując ręczne ustawianie wcięć i łamań linii. [49](#), [61](#)

PRO

Przedmiot realizowany na 5. semestrze studiów, prowadzony przez dr. inż. Martę Łabudę. W ramach przedmiotu wybrano temat projektu oraz wytworzono wstępną dokumentację projektu w tym m.in. wymagania.. [52](#)

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.. [72](#)

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). [71](#)

PRZ 1

Przedmiot realizowany na 6. semestrze studiów, prowadzony w przypadku zespołu projektowego przez mgr. inż. Adama Urbanowicza. W ramach przedmiotu wytworzono projekt interfejsu użytkownika.. [55](#), [57](#)

PRZ 2

Przedmiot realizowany na 7. semestrze studiów, prowadzony w przypadku zespołu projektowego przez mgr. inż. Adama Urbanowicza. W ramach przedmiotu dokończono prace nad pracą inżynierską. Pan Adam Urbanowicz jako promotor doradzał zespołowi projektowemu.. [64](#)

PSEM

Przedmiot realizowany na 7. semestrze studiów, prowadzony w przypadku zespołu projektowego przez dr. inż. Marka Bednarczyka. W ramach przedmiotu dokończono wytwarzanie dokumentacji.. [64](#)

React

Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. [49](#), [72](#)

React-MapLibre

Otwartoźródłowa biblioteka do renderowania interaktywnych map wektorowych w przeglądarce, rozwijana jako niezależna kontynuacja Mapbox GL JS. Umożliwia wyświetlanie kafelków mapowych, znaczników i warstw z danymi geoprzestrzennymi. [58](#)

Redis

Szybka baza danych typu klucz–wartość przechowywana głównie w pamięci operacyjnej. Często wykorzystywana jako pamięć podręczna (cache), magazyn sesji lub prosty mechanizm komunikatów między usługami. [56](#)

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.. [53](#), [54](#), [61](#), [72](#), [73](#), [106](#)

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.. [80](#)

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.

Review kodu

Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. [14](#), [15](#), [48](#), [91](#)

Rich text editor

Edytor treści, który zamiast „surowego” tekstu umożliwia stosowanie formatowania (np. pogrubienie, kursywa, listy, nagłówki, linki), dzięki czemu użytkownik może tworzyć czytelne, sformatowane wpisy. [62](#)

routing

Routing w [SPA](#) to warstwa w kliencie odpowiedzialna za zarządzanie stanem “aktualnej strony” na podstawie URL-a, zwykle z wykorzystaniem historii przeglądarki, tak aby interfejs reagował na zmianę ścieżki bez przeładowań z serwera.. [50](#)

Sidebar

Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. [46](#), [54](#), [58](#), [59](#), [91–93](#)

Skeleton loader

Wzorzec prezentowania stanu ładowania, w którym zamiast klasycznego „spinera” wyświetlane są szare prostokąty imitujące docelowy układ treści. Poprawia subiektywne odczucie szybkości działania aplikacji. [66](#)

slice Redux

Slice Redux to wydzielona część globalnego stanu w [Redux](#), wraz z powiązanymi akcjami i reduktorami, odpowiedzialna za jeden obszar domeny (np. konto użytkownika, czat, mapę czy listę znajomych).. [59](#), [60](#)

Slug

Przyjazny dla użytkownika fragment adresu URL, zwykle oparty na tytule (np. `/post/jak-zaczac-latac-dronem`), ułatwiający identyfikację treści i

pozycjonowanie w wyszukiwarkach. [64](#)

SPA

SPA (Single Page Application) to aplikacja webowa, w której cała strona ładuje się raz, a późniejsze zmiany widoku odbywają się dynamicznie po stronie przeglądarki bez pełnego przeładowania strony.. [106](#)

Spot

Potencjalne miejsce do latania dronem, zaznaczone na mapie.. [51](#), [53–64](#), [66](#), [83–88](#)

Spring Boot

Framework w ekosystemie Spring dla języka Java, ułatwiający tworzenie aplikacji backendowych dzięki automatycznej konfiguracji, wbudowanemu serwerowi aplikacyjnemu oraz zestawowi gotowych starterów. [49](#)

Spring Security

Moduł bezpieczeństwa w ekosystemie Spring odpowiedzialny za uwierzytelnianie i autoryzację użytkowników. Zapewnia obsługę różnych mechanizmów logowania, ról i uprawnień oraz integrację z różnymi źródłami danych. [53](#), [56](#)

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).. [72](#)

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](#), [49](#)

Tailwind CSS

Framework CSS typu *utility-first*, dostarczający gotowe klasy narzędziowe do określania wyglądu (kolory, odstępy, layout). Umożliwia szybkie prototypowanie i spójne stylowanie komponentów bez pisania rozbudowanych arkuszy CSS. [49](#), [58](#)

TanStack Query

Biblioteka do obsługi zapytań do serwera i cachowania danych w aplikacjach frontendowych (m.in. React). Ułatwia zarządzanie stanem danych z backendu: pobieranie, odświeżanie, invalidację i obsługę błędów. [50](#)

Testy E2E

Testy *end-to-end*, które sprawdzają działanie systemu od strony użytkownika, przechodząc przez wszystkie warstwy aplikacji (frontend, backend, baza danych) i symulując rzeczywiste scenariusze użycia. [51](#), [52](#), [54](#)

Testy integracyjne

Testy sprawdzające, czy połączone ze sobą moduły lub usługi współpracują poprawnie — na przykład czy warstwa backendowa poprawnie komunikuje się z bazą danych, warstwą sieciową i pozostałymi komponentami systemu. [51](#), [54](#), [56](#)

testy jednostkowe

Testy sprawdzające poprawność działania pojedynczych, małych fragmentów kodu (np. funkcji, metod, klas) w izolacji od reszty systemu.. [54](#)

TinyMCE

Popularny edytor *rich text* osadzany w przeglądarce. Pozwala użytkownikowi formatować tekst (pogrubienia, listy, nagłówki, linki) w sposób przypominający klasyczny edytor tekstu, zapisując wynik zwykle w HTML. [62](#), [64](#)

Tiptap

Nowoczesny, rozszerzalny edytor *rich text* dla aplikacji webowych oparty na silniku ProseMirror. Umożliwia budowanie rozbudowanych, modularnych edytorów treści, np. do postów na forum. [64](#)

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.. [49](#), [58](#), [59](#), [72](#)

UI

Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. [15](#), [72](#)

UML

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

URL

Adres zasobu w internecie (ang. *Uniform Resource Locator*), np. adres strony, widoku w aplikacji webowej lub konkretnego posta na forum. [64](#), [99](#)

WebSocket

Protokół komunikacyjny umożliwiający dwukierunkową komunikację w czasie rzeczywistym między przeglądarką a serwerem po pojedynczym, utrzymanym połączeniu TCP. Często wykorzystywany m.in. w czatach i aplikacjach działających w czasie rzeczywistym.. [60](#), [61](#)

Spis tabel

Tabela 2.1: Karta udziałowca: Zespół projektowy	7
Tabela 2.2: Karta udziałowca: Promotor	8
Tabela 2.3: Karta udziałowca: Droniarze	9
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	18
Tabela 3.5: Usługa zewnętrzna: Google Maps (Maps URLs)	18
Tabela 3.6: Usługa zewnętrzna: OpenFreeMap	18
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?	19
4.1 Profil użytkownika	31
4.2 Lista dodanych spotów	32
4.3 Dodanie spota	33
4.4 Lista zdjęć	34
4.5 Lista filmów	34
4.6 Lista znajomych	35
4.7 Lista obserwujących	35
4.8 Lista obserwowanych	36
4.9 Lista polubionych/odwiedzonych/planowanych spotów	36
4.10 Lista komentarzy	37
4.11 Ustawienia profilu	38

4.12	Resetowanie hasła	39
4.13	Dodawanie do znajomych	40
4.14	Logowanie i rejestracja	41
4.15	Strona główna — podstawowe filtry	42
4.16	Strona główna — zaawansowane filtry	43
4.17	Ustawienia motywu (ręczna zmiana)	44
4.18	Zapamiętanie preferencji motywu	45
4.19	Szybki przełącznik motywu w interfejsie	46

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