



POLSKO-JAPONSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Filia w Gdańsku

Langmesser Adam

Nr albumu s27119

Nazwa specjalizacji: Aplikacje Internetowe

Redosz Mateusz

Nr albumu s27094

Nazwa specjalizacji: Aplikacje Internetowe

Oziemczuk Stanisław

Nr albumu s26982

Nazwa specjalizacji: Aplikacje Internetowe

Badek Kacper

Nr albumu s29168

Nazwa specjalizacji: Aplikacje Internetowe

Aplikacja webowa: spotty-na-drony.pl

Rodzaj pracy
inżynierska

Imię i nazwisko promotora
mgr. inż. Adam Urbanowicz

Gdańsk, Luty 2026

Streszczenie: Celem niniejszej pracy było stworzenie w pełni funkcjonalnej aplikacji internetowej umożliwiającej szybkie wyszukiwanie [spotów](#) w okolicy oraz dzielenie się zdjęciami, filmami i doświadczeniami z innymi użytkownikami. W ramach pracy opracowano system składający się z trzech komponentów: warstwy [frontendowej](#), warstwy [backendowej](#) oraz [bazy danych](#). Aplikację internetową wykonano z wykorzystaniem [frameworka React](#) w języku JavaScript oraz jego rozszerzeniu ([TypeScript](#)), a do stylizacji użyto [Tailwind CSS](#). Serwis [backendowy](#) stworzono w języku Java z wykorzystaniem [frameworka Spring Boot](#). Bazę danych oparto na systemie PostgreSQL.

Komunikacja między komponentami odbywała się poprzez [API](#) zgodne ze stylem architektonicznym [REST](#). Projekt zrealizowano w podejściu **Disciplined Agile Delivery** [1] w wariantie **Lean Life Cycle** [2].

Słowa kluczowe: spot, dron, droniarz



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 2024-10-10
Promotor: mgr. inż. Adam Urbanowicz	Konsultanci: 1. — brak —
Cele projektu: Stworzenie w pełni funkcjonalnej aplikacji internetowej wspierającej rozwijanie hobby związanego z lataniem dronem.	
Rezultaty projektu: Aplikacja internetowa, Dokumentacja Interaktywna mapa z wyświetlonymi spotami oraz danymi pogodowymi. Zaawansowana wyszukiwarka spotów. Forum do dzielenia się informacjami na temat dronów. Czat jednoosobowy oraz grupowy. Konto użytkownika z możliwością zapisywania ulubionych spotów.	
Miary sukcesu: W pełni działająca aplikacja. Realizacja projektu w terminie zgodnym z wymaganiami.	
Ograniczenia: Budżetowe: brak środków na komercyjne wdrożenie. Kompetencyjne: ograniczone wcześniejsze doświadczenie zespołu. Czasowe: trzy semestry (09.2024–02.2026). Zasobowe: czteroosobowy zespół.	

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

Data ukończenia projektu: 2026-01-11	Recenzent: dr. Elżbieta Puśniakowska-Gałuch
--	---

Spis treści

1 Wstęp	10
Uwagi redakcyjne	10
1.1 O projekcie	10
1.2 Cel i zakres prac	10
1.3 Geneza pomysłu	10
2 Opis problemu	11
2.1 Rich picture	11
2.2 Udziałowcy	11
2.3 Istniejące rozwiązania	13
2.4 Wizja rozwiązania	13
2.5 Aspekty społeczne i biznesowe	13
2.5.1 Aspekty społeczne	13
2.5.2 Aspekty biznesowe	13
3 Planowanie	14
3.1 Metodologia pracy	14
3.1.1 Przegląd rozważanych podejść	14
3.1.2 Odrzucone podejścia	15
3.1.3 Wybrane podejście: Disciplined Agile Delivery	15
3.1.4 Narzędzia i komunikacja	16
3.1.5 Podział ról w zespole	16
3.2 Harmonogram projektu	16
3.3 Technologie i narzędzia	18
3.3.1 Technologie	18

3.3.2	Narzędzia	26
3.4	Zasoby i ograniczenia	29
3.4.1	Zasoby	29
3.4.2	Ograniczenia	29
3.4.3	Usługi zewnętrzne	30
3.5	Analiza ryzyka	32
4	Analiza wymagań	33
4.1	Przypadki użycia	34
4.1.1	Aktorzy	34
4.1.2	Diagramy przypadków użycia	35
4.1.3	Scenariusze przypadków użycia	39
4.1.3.1	Scenariusze przypadków użycia – funkcje ogólne .	39
4.1.3.2	Scenariusze przypadków użycia dla wyszukiwarki .	44
4.1.3.3	Scenariusze przypadków użycia dla mapy	46
4.1.3.4	Scenariusze przypadków użycia dla czatu	52
4.1.3.5	Scenariusze przypadków użycia dla forum	60
4.1.3.6	Scenariusze przypadków użycia dla panelu użytkownika	69
4.2	Wymagania	78
4.2.1	Wymagania ogólne	79
4.2.1.1	Wymagania ogólne dla czatu	79
4.2.1.2	Wymagania ogólne dla mapy	81
4.2.1.3	Wymagania ogólne dla wyszukiwarki spotów	84
4.2.2	Wymagania funkcjonalne	85
4.2.2.1	Wymagania funkcjonalne dla mapy	85
4.2.2.2	Wymagania funkcjonalne dla czatu	114
4.2.2.3	Funkcjonalności dla forum	124
4.2.2.4	Funkcjonalności dla konta użytkownika	124
4.2.2.5	Funkcjonalności dla logowania i rejestracji	134
4.2.2.6	Wymagania funkcjonalne dla wyszukiwarki spotów	134
4.2.2.7	Funkcjonalności dla motywu	143

4.2.3	Wymagania pozafunkcjonalne	145
4.2.3.1	Wymagania pozafunkcjonalne dla czatu	145
4.2.3.2	Wymagania pozafunkcjonalne dla mapy	151
4.2.3.3	Wymagania pozafunkcjonalne dla wyszukiwarki spo- tów	153
5	Projekt	156
5.1	Architektura systemu	156
5.1.1	Diagram architektury	157
5.1.2	Komponenty systemu	159
5.2	Projekt bazy danych	160
5.2.1	Model danych	160
5.2.2	Diagram ERD	160
5.3	Architektura interfejsu użytkownika	160
5.3.1	Projekt strony głównej	160
5.3.2	Projekt panelu logowania	164
5.3.3	Projekt mapy	166
5.3.4	Projekt chatu	166
5.3.5	Projekt forum	166
5.3.6	Projekt panelu użytkownika	166
5.3.6.1	Profile	166
5.3.6.2	Spots list	172
5.3.6.3	Photos list	173
5.3.6.4	Movies list	175
5.3.6.5	Friends	176
5.3.6.6	Add spot	177
5.3.6.7	Comments	180
5.3.6.8	Settings	180
6	Przebieg realizacji projektu	183
6.1	Elementy niezależne od modułu	183
6.2	Moduły aplikacji	184

6.2.1	Rozwój funkcjonalności w modułach	184
6.2.1.1	Mapa	185
6.2.1.2	Czat	186
6.2.1.3	Forum	186
6.2.1.4	Wyszukiwarka spotów	187
6.2.1.5	Panel użytkownika	187
6.3	Podsumowanie etapu finalizacji	188
7	Realizacja Projektu	189
7.1	Wzorce projektowe	189
7.1.1	Backend	189
7.1.1.1	Fasada	189
7.1.1.2	Singleton	191
7.1.1.3	Builder	193
7.1.1.4	Chain of Responsibility	196
7.1.2	Frontend	196
7.1.2.1	Hooks Pattern	196
7.1.2.2	Optimistic UI	197
7.1.2.3	Protected route	204
7.1.2.4	Portal	206
7.2	Implementacja backendu	209
7.2.1	Struktura projektu	209
7.2.2	Endpointy systemu	213
7.2.2.1	Panel użytkownika	218
7.2.2.2	Spoty i pogoda	223
7.2.2.3	Wyszukiwarka spotów	228
7.2.2.4	Komentarze do spotów	232
7.2.2.5	Forum – posty	237
7.2.2.6	Forum – komentarze do postów	242
7.2.2.7	Konto użytkownika – rejestracja, logowanie, hasło .	247
7.2.2.8	GIF-y (Tenor) – integracja czatu	251
7.2.3	Integracja z bazą danych	259

7.2.4	Obsługa uwierzytelnienia	262
7.2.5	Konteneryzacja	262
7.2.6	Cache	263
7.3	Implementacja frontendu	266
7.3.1	Struktura aplikacji	266
7.3.2	Zarządzanie stanem i przepływ danych	272
7.3.3	Integracja i komunikacja z backendem	275
7.3.4	Style	278
7.3.5	Wyszukiwarka spotów	282
7.3.6	Mapa	289
7.3.6.1	Mapa z zaznaczonymi spotami	289
7.3.6.2	Panel ze szczegółami spota	301
7.3.6.3	Panel z informacjami pogodowymi	319
7.3.6.4	Duża galeria zdjęć i filmów	334
7.3.7	Chat	343
7.3.8	Forum	343
7.3.9	Panel użytkownika	343
7.3.9.1	Profile	344
7.3.9.2	Spots	347
7.3.9.3	Photos	350
7.3.9.4	Movies	354
7.3.9.5	Social	355
7.3.9.6	Add spot	362
7.3.9.7	Comments	366
7.3.9.8	Settings	368
7.3.9.9	Komponenty wspólne	370
7.3.10	Panel logowania	375
7.3.10.1	Logowanie	376
7.3.10.2	Rejestracja	378
7.3.10.3	Reset hasła	381
7.3.10.4	Nowe hasło	384

7.3.10.5	Komponenty wspólne panelu logowania	387
7.3.11	Sidebar	392
7.3.11.1	Komponenty	393
7.3.11.2	Pliki pomocnicze	412
7.3.11.3	Redux	417
7.3.11.4	Użycie w układzie aplikacji	418
7.4	Implementacja CI/CD	419
7.4.1	Pipeline backendu	420
7.4.1.1	Job setup	421
7.4.1.2	Job build	423
7.4.1.3	Job test	425
7.4.2	Pipeline frontendu	427
7.5	Implementacja WebSocket	429
7.5.1	Charakterystyka protokołu WebSocket	429
7.5.2	Zastosowanie WebSocket w naszym projekcie	430
7.5.3	Implementacja na backendzie	431
7.5.3.1	Konfiguracja WebSocket	432
7.5.3.2	Kontroler STOMP	434
7.5.3.3	Serwis dystrybucji wiadomości	434
7.5.3.4	Zestawienie wykorzystywanych destynacji	436
7.5.4	Implementacja na frontendzie	436
7.5.4.1	Serwis komunikacji WebSocket	438
7.5.4.2	Mechanizm rejestracji subskrypcji	444
7.5.4.3	Kontekst i zarządzanie połączeniem	446
7.5.4.4	Definicja subskrypcji czatu	448
7.5.5	Przebieg komunikacji	449
8	Testy	450
8.1	Testy jednostkowe	450
8.2	Testy integracyjne	453
8.3	Testy end-to-end (E2E)	455
8.4	Scenariusze testów end-to-end (E2E)	457

8.4.1	Account (logowanie i rejestracja)	457
8.4.2	User dashboard – Add spot	459
8.4.3	User dashboard – Comments	462
8.4.4	User dashboard – Favorite spots	464
8.4.5	User dashboard – Movies	466
8.4.6	User dashboard – Photos	468
8.4.7	User dashboard – Profile	470
8.4.8	User dashboard – Settings	474
8.4.9	User dashboard – Social (friends/followed/followers)	477
8.5	Wyniki testów i wnioski	480
9	Prezentacja systemu	481
9.1	Strona główna	481
9.2	Strona mapy	481
9.3	Strona czatu	481
9.3.1	Widok bazowy modułu czatu	481
9.4	Strona forum	484
9.5	Panel logowania	484
9.6	Panel konta użytkownika	484
10	Nakład pracy	485
10.1	Ogólny nakład pracy	485
10.2	Indywidualne nakłady pracy	485
10.2.1	Adam Langmesser	485
10.2.2	Mateusz Redosz	485
10.2.3	Stanisław Oziemczuk	488
10.2.4	Kacper Badek	488
11	Podsumowanie	489
11.1	Osiągnięte rezultaty	489
11.2	Napotkane wyzwania	490
11.3	Plany na przyszłość	490
12	Słownik pojęć i skrótów	491

Spis rysunków	513
Spis tabel	521
Bibliografia	533
Załączniki	535

Rozdział 1

Wstęp

Uwagi redakcyjne

O ile nie wskazano inaczej, wszystkie rysunki, diagramy i ilustracje zamieszczone w pracy stanowią opracowanie własne. W przypadku materiałów pochodzących ze źródeł zewnętrznych informacja o źródle została podana w podpisie pod ilustracją.

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:	U1
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:	U2
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 załatwia.
Powiązane wymagania:	Czytelna dokumentacja, zgodność z wytycznymi kierunku oraz odpowiedni poziom techniczny rozwiązania.

Tabela 2.2: Karta udziałowca: Promotor

KARTA UDZIAŁOWCA	
Identyfikator:	U3
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

W niniejszym rozdziale przedstawiono sposób zaplanowania realizacji projektu oraz założenia organizacyjne. W pierwszej kolejności opisano proces wyboru metodyki pracy oraz uzasadniono przyjęte podejście spośród rozważanych alternatyw. Następnie zaprezentowano harmonogram projektu, obejmujący podział działań w czasie oraz kluczowe etapy realizacji. Kolejna część rozdziału przedstawia technologie i narzędzia wykorzystywane do komunikacji w zespole oraz zarządzania przepływem pracy. Rozdział uzupełniają informacje o dostępnych zasobach i ograniczeniach projektowych, a także analiza ryzyka wraz z omówieniem potencjalnych zagrożeń i sposobów ich ograniczania.

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, retrospektyna). 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

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

Kluczowe argumenty wyboru:

- **Brak sprintów.** Zastosowano przepływ ciągły, co pozwala dopasować tempo do zmiennej dostępności zespołu i unikać sztucznego „domykania” iteracji.
- **Rozbudowana faza startowa.** Na początku przewidziano większy wysiłek planistyczny: doprecyzowanie zakresu, wstępna wizja architektury, identyfikacja ryzyk, plan publikacji oraz kryteria jakości – bez zamrażania szczegółowo.
- **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.

- 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

W ramach prac nad projektem 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

Do realizacji projektu zespół wspólnie wytypował główne technologie części [backendowej](#), [frontendowej](#) oraz dokumentacji. Natomiast poszczególne biblioteki i rozwiązania były wybierane indywidualnie lub po konsultacjach przez osobę wykonującą dane zadanie. Poniżej przedstawiono stos technologiczny zastosowany w projekcie.

- **Backend**

Na główny [framework](#) został wybrany SpringBoot, ponieważ spośród innych dostępnych opcji, członkowie zespołu mieli z nim największe doświadczenie nabyte zarówno podczas studiów, jak i później pracę komercyjną. Językiem programowania wykorzystywanym w SpringBoot'cie jest Java, z którym zespół zapoznał się w ramach programu nauczania.

- **Java** – obiektowy język programowania, cechujący się silnym typowaniem. Programy napisane w Javie są uruchamiane na maszynie wirtualnej Java ([JVM](#)), dzięki czemu można je bezproblemowo przenosić między różnymi platformami wyposażonymi w to środowisko.
- **SpringBoot** – [framework](#) służący do tworzenia aplikacji opartych na [Spring Framework](#). Wykorzystuje strategię [Convention Over Configuration](#), która zmniejsza czas na konfigurowanie Springa, pozwalając skupić się na implementacji logiki. Służy do tworzenia między innymi aplikacji internetowych czy mikroserwisów.

Zestawienie używanych bibliotek na backendzie	
Biblioteka	Opis
angus-mail	Wysyłanie i odbiór wiadomości e-mail w aplikacjach Java.
azure-storage-blob	Operacje na blobach w Microsoft Azure Blob Storage, np. upload, download.
GeographicLib-Java	Precyzyjne obliczenia geodezyjne i konwersja współrzędnych.
h2	Lekka baza danych H2 uruchamiana w pamięci RAM używana w testach jako zastępstwo prawdziwej.
httpclient5	Zaawansowany klient HTTP do wykonywania żądań i obsługi odpowiedzi.
httpcore5	Niskopoziomowe elementy HTTP wykorzystywane przez httpclient.
jjwt-api	Tworzenie i parsowanie JWT.
jjwt-impl	Implementacja funkcjonalna biblioteki JJWT.
jjwt-jackson	Integracja JJWT z Jacksonem dla serializacji i deserializacji zawartości JWT.
jsoup	Parsowanie, manipulacja i ekstrakcja danych z dokumentów HTML.
junit-jupiter	Integracja biblioteki Testcontainers z frameworkiem testowym JUnit ułatwiająca pisanie testów integracyjnych z użyciem kontenerów.
lombok	Biblioteka ułatwiająca generowanie kodu (gettery, settery, buildery, konstruktor itp.) przez adnotacje zmniejszające ilość boilerplate'u w kodzie.
postgresql	Umożliwia połączenie i komunikację z bazą danych PostgreSQL.
shedlock-spring	Zarządzanie zadaniami okresowymi (cron/scheduled).

Biblioteka	Opis
spring-boot-starter-websocket	Wsparcie Spring Boot dla komunikacji WebSocket – konfiguracja i zależności umożliwiające dwukierunkową komunikację w czasie rzeczywistym w aplikacjach webowych.
spring-security-messaging	Integracja Spring Security z warstwą messaging (STOMP/WebSocket) – uwierzytelnianie i autoryzacja komunikatów.
spring-boot-starter-cache	Abstrakcje i automatyczna konfiguracja cache w Spring Boot ułatwiające włączenie mechanizmów cache'owania.
spring-boot-starter-data-redis	Integracja Spring Data Redis dodające klienta, repozytoria i konfiguracje do współpracy z Redis.
spring-boot-starter-data-jpa	Warstwa dostępu do relacyjnej bazy danych przez JPA.
spring-boot-starter-web	Podstawowy starter webowy Spring Boot: Spring MVC, Jackson, wbudowany serwer aplikacyjny dla REST/HTTP.
spring-boot-starter-validation	Wsparcie walidacji Bean Validation (Jakarta Validation / Hibernate Validator) dla danych wejściowych.
spring-boot-starter-aop	Obsługa programowania aspektowego (AOP) w Springu – aspekty, przechwytywanie wywołań, transakcyjne zachowania.
spring-boot-starter-actuator	Monitoring, zbieranie metryk aplikacji Spring Boot.
spring-boot-starter-oauth2-resource-server	Wsparcie serwera zasobów OAuth2 w Spring Boot – walidacja tokenów i konfiguracja zabezpieczeń zasobów.
spring-boot-configuration-processor	Procesor adnotacji dla konfiguracji Spring Boot – generacja metadanych dla właściwości konfiguracyjnych.

Biblioteka	Opis
spring-boot-starter-test	Zestaw narzędzi testowych (JUnit, Mockito, AssertJ itp.) do testów jednostkowych i integracyjnych.
spring-security-test	Narzędzia pomocnicze i rozszerzenia do testowania konfiguracji Spring Security.
spring-boot-starter-oauth2-client	Wsparcie klienta OAuth2 w Spring Boot – logowanie/połączenie z zewnętrznymi providerami OAuth2/OIDC.
spring-boot-starter-security	Podstawowe komponenty Spring Security do zabezpieczania aplikacji.
spring-boot-starter-webflux	Budowanie reaktywnych (asynchronicznych / nieblokujących) aplikacji webowych.
spring-retry	Biblioteka do automatycznego ponawiania operacji z konfiguracją i adnotacjami.
spring-boot-starter-thymeleaf	Integracja Thymeleaf z Spring Boot – silnik szablonów do generowania widoków HTML po stronie serwera.
testcontainers	Uruchamianie izolowanych kontenerów Docker w testach integracyjnych.

Tabela 3.1: Zestawienie wszystkich bibliotek użytych na backendzie.

- **Frontend**

Do realizacji tej części projektu wybrano [bibliotekę React](#) JavaScript, którą wszyscy członkowie zespołu poznali w trakcie studiów w ramach wybranej specjalizacji Aplikacje Internetowe.

- [React](#) – [biblioteka](#) JavaScript służąca do budowania interaktywnych interfejsów użytkownika ([UI](#)). Polega na programowaniu deklaratywnym oraz tworzeniu komponentów wielokrotnego użytku. Nie manipuluje bezpośrednio [DOM](#), lecz tworzy swój wirtualny [DOM](#) i porównuje jego wersje. Po wykryciu zmian aktualizuje tylko te części [DOM](#), które

tego wymagają, co przekłada się na wydajną interakcję z aplikacją. Często jest wykorzystywany do tworzenia aplikacji typu [SPA](#).

Zestawienie używanych bibliotek na frontendzie	
Biblioteka / plugin	Opis / przeznaczenie
@ferrucc-io/emoji-picker	Komponent umożliwiający wybór emotikon w aplikacji.
@hookform/resolvers	Adaptery validatorów dla <code>react-hook-form</code> – ułatwia integrację z bibliotekami validacji (Zod, Yup).
@reduxjs/toolkit	Narzędzie upraszczające konfigurację i używanie Reduxa.
@stomp/stompjs	Klient protokołu STOMP do komunikacji poprzez WebSocket obsługujący sesje, subskrypcje i wymiany komunikatów.
@tailwindcss/vite	Wtyczka integrująca Tailwind CSS z bundlerem Vite.
@tanstack/react-query	Zarządzanie asynchronicznymi danymi: pobieranie, cache'owanie, synchronizacja i obsługa błędów zapytań HTTP.
@tiptap/extension-file-handler	Rozszerzenie edytora Tiptap dodające obsługę wstawiania i zarządzania plikami.
@tiptap/extension-image	Rozszerzenie Tiptap pozwalające na wstawianie i obsługę obrazów w edytorze.
@tiptap/extension-placeholder	Rozszerzenie Tiptap dodające placeholder w polach edytora.
@tiptap/extension-text-align	Rozszerzenie Tiptap umożliwiające wyrównywanie tekstu.
@tiptap/pm	Silnik edytora (ProseMirror) używany wewnętrznie przez Tiptap obsługujący model dokumentu i transformacji.

Biblioteka / plugin	Opis / przeznaczenie
@tiptap/react	Integracja edytora Tiptap z Reactem dodająca komponenty i hooki edytora.
@tiptap/starter-kit	Podstawowe rozszerzenia i konfiguracja Tiptap.
@vis.gl/react-maplibre	Narzędzia do implementacji map.
antd	Komponenty UI dla React o ustandaryzowanym wyglądzie.
axios	Obiektowy klient HTTP służący do wykonywania zapytań oraz obsługi odpowiedzi i błędów.
date-fns	Funkcje do manipulacji i formatowania dat.
dotenv	Ładowanie zmiennych środowiskowych z pliku .env.
maplibre-gl	Silnik renderowania map – warstwy, kafelki i interakcje geograficzne.
media-chrome	Elementy UI obsługujące odtwarzanie multimedialów w przeglądarce.
motion	Narzędzia służące do animacji interfejsu użytkownika.
query-string	Narzędzia do parsowania i generowania parametrów zapytań w URL.
react	Biblioteka do budowy deklaratywnych interfejsów użytkownika oparta na komponentach.
react-dom	Pakiet odpowiedzialny za renderowanie elementów React w przeglądarkowym DOM.
react-hook-form	Obsługa formularzy: zarządzanie stanem, walidacje i wydajność.
react-icons	Zbiór ikon udostępnionych jako komponenty React.
react-intersection-observer	Obserwowanie wejścia/wyjścia elementów z pola widzenia.

Biblioteka / plugin	Opis / przeznaczenie
react-paginate	Komponent pomocniczy do paginacji list i tabel w aplikacji React.
react-player	Odtwarzanie multimedialnych w aplikacji.
react-redux	Integracja Redux i Reacta.
react-router-dom	Routing w aplikacjach React.
react-select	Rozszerzony komponent <code>select</code> z opcjami wyszukiwania, grupowania i stylizacji.
sockjs-client	Klient WebSocket z fallbackami umożliwiający stabilniejszą komunikację w czasie rzeczywistym.
tailwindcss	Definiowanie wyglądu przy pomocy gotowych klas.
uuid	Generator unikalnych identyfikatorów.
victory	Tworzenie wykresów i wizualizacji danych w React.
victory-chart	Moduł biblioteki Victory zawierający komponenty i narzędzia do rysowania wykresów.
zod	Typowana walidacja danych.
@testing-library/jest-dom	Rozszerzenia matcherów dla Jesta do asercji DOM.
@testing-library/react	Narzędzia do testowania komponentów React.
@testing-library/user-event	Symulacja zdarzeń użytkownika (np. kliknięcie) w testach integracyjnych.
@types/react	Typy TypeScript dla biblioteki React.
@types/react-dom	Typy TypeScript dla <code>react-dom</code> .
@types/sockjs-client	Typy TypeScript dla klienta SockJS.
@typescript-eslint/eslint-plugin	Zestaw reguł ESLint specyficznych dla TypeScript.
@typescript-eslint/parser	Parser ESLint umożliwiający analizę kodu TypeScript.

Biblioteka / plugin	Opis / przeznaczenie
@vitejs/plugin-react	Wtyczka Vite zapewniająca obsługę JSX/React Fast Refresh i optymalizacje.
cypress	Narzędzie do testów end-to-end.
eslint	Narzędzie do analizy statycznej kodu na podstawie zdefiniowanych reguł.
eslint-plugin-react	Wtyczka ESLint z regułami dedykowanymi dla aplikacji React.
eslint-plugin-react-hooks	Reguły ESLint dotyczące hooków React.
eslint-plugin-react-refresh	Wtyczka wspomagająca integrację z React Fast Refresh.
jest	Tworzenie testów jednostkowych JavaScript.
jsdoc	Narzędzie do generowania dokumentacji API z adnotacjami w kodzie źródłowym.
jsdom	Implementacja DOM w Node.js używana w testach do symulacji środowiska przeglądarki.
prettier	Formatowanie i ujednolicenie stylu kodu.
prettier-plugin-tailwindcss	Wtyczka Prettiera sortująca klasy Tailwind CSS.
tailwind-scrollbar	Plugin Tailwind CSS dodający klasy pomocnicze do stylizacji pasków przewijania.
typescript	Nakładka JavaScript z systemem typów.
vite	Bundler zoptymalizowany pod nowoczesne aplikacje webowe.
vitest	Narzędzie do uruchamiania testów zoptymalizowane pod bundler Vite.

Tabela 3.2: Zestawienie wszystkich bibliotek i pluginów użytych na frontendzie.

- **Cache**

- **Redis** – z ang. REmote DIctionary Server, nierelacyjna (NoSQL) baza danych przechowująca dane jako pary klucz - wartość. Działa w pamięci RAM, dzięki czemu pozwala na bardzo szybki odczyt danych.

- **Konteneryzacja**

- **Docker** – to silnik do konteneryzacji oprogramowania. Tworzy środowisko oddzielone od systemu hosta, w którym na podstawie obrazów programów tworzone są ich kontenery, czyli niezależne ich instancje. Docker pobiera wszystkie niezbędne dependencje dla danego kontenera, bez potrzeby uruchamiania osobnego systemu operacyjnego, dzięki czemu kontenery są znacznie lżejsze niż maszyny wirtualne. Konteneryzacja pozwala na uruchomienie oprogramowania na różnych komputerach dokładnie w taki sam sposób, rozwiązuje to problem „na mojej maszynie działa”. Członkowie zespołu zapoznali się tą z technologią w trakcie realizacji specjalizacji Aplikacje Internetowe.

- **Baza danych**

- **PostgreSQL** – system zarządzania relacyjnymi bazami danych, zgodny ze standardem SQL. Wybrano relacyjną bazę danych, ponieważ doskonale wpisuje się ona w planowaną strukturę danych.

3.3.2 Narzędzia

Do niektórych płatnych narzędzi otrzymano bezpłatny dostęp za pośrednictwem uczelni, w innych istniała możliwość założenia konta edukacyjnego, które oferowało dostęp do wszystkich funkcji narzędzia. Gdy żadna z wymienionych opcji nie była udostępniona, wybierano 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. Używano 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. Wykorzystywano je do lokalnego uruchamiania bazy danych oraz serwisu do cachowania.

- **Docker Compose**

Narzędzie, które pozwala definiować oraz uruchamiać aplikacje składające się z wielu kontenerów Docker. Konfiguracja serwisów, sieci i [wolumenów](#) jest ustawiana w pliku (lub plikach) YAML. Zastosowano je do skonfigurowania bazy danych i serwisu do [cache'owania](#) w środowisku deweloperskim.

- **One Drive**

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

- **Azure Blob Storage**

To rozwiązanie chmurowe Microsoft, służące do bezpiecznego przechowywania dużej ilości danych niestrukturyzowanych, takich jak pliki multimedialne, dokumenty czy kopie zapasowe. Dane są dostępne poprzez interfejs [REST API](#) usługi Azure Storage. Wykorzystywano je 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 metodach zwinnych. Do [Backlogu](#) wpisywano zadania, a na [tablicy Kanbanowej](#) rejestrowano ich statusy oraz poświęcony czas.

- **GitHub**

Zdalne repozytorium służące do przechowywania i wersjonowania kodu aplikacji. Zamieszczono tam kod naszego projektu. Do każdego zadania tworzono osobną gałąź z właściwą nazwą, a po zakończeniu prac przeprowadzano [review kodu](#). Następnie łączono 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. Stosowano 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. Wykorzystywano go podczas [review kodu](#). Copilot skanuje wszystkie pliki i w komentarzach opisuje sugerowane zmiany lub potencjalne błędy.

- **Discord**

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

- **Messenger**

Komunikator będący usługą Facebooka. Daje możliwość tworzenia czatów grupowych lub prywatnych, a także udostępniania plików. Używano 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. Wykorzystywano go do testowania stworzonych endpointów oraz debugowania.

- **Figma**

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

- **Visual Paradigm**

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

- **Xmind**

Narzędzie służące do tworzenia mapy myśli. Wykorzystano je w celu lepszego zrozumienia problemów poprzez przeniesienie ich na diagram.

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:	UZ3
Nazwa:	GitHub Actions (CI) [5]
Opis:	Uruchomienia pipeline'ów CI/CD dla repozytorium GitHub.
Limit:	3000 min/mies.

Tabela 3.3: Usługa zewnętrzna: GitHub Actions (CI)

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ4
Nazwa:	Azure Blob Storage [6]
Opis:	Magazyn plików (m.in. zdjęcia spotów, załączniki z czatu).
Limit:	1 GB/mies.

Tabela 3.4: Usługa zewnętrzna: Azure Blob Storage

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ5
Nazwa:	Mailtrap [7]
Opis:	Środowisko testowe SMTP oraz Email API do wysyłki maili.
Limit:	150 maili/dzień

Tabela 3.5: Usługa zewnętrzna: Mailtrap

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ6
Nazwa:	LocationIQ [8]
Opis:	Geokodowanie adresu przy dodawaniu nowych spotów.
Limit:	5 000 zapytań/dzień

Tabela 3.6: Usługa zewnętrzna: LocationIQ

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ7
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 3.7: Usługa zewnętrzna: Google Maps (Maps URLs)

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ8
Nazwa:	OpenFreeMap [10]
Opis:	Publiczny serwer kafelków do renderu mapy na froncie.
Limit:	30 000 zapytań/mies.

Tabela 3.8: Usługa zewnętrzna: OpenFreeMap

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ9
Nazwa:	Open-Meteo [11]
Opis:	Prognozy pogody wyświetlane dla spotów.
Limit:	10 000 zapytań/dzień

Tabela 3.9: Usługa zewnętrzna: Open-Meteo

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ10
Nazwa:	Tenor GIF API [12]
Opis:	Wyszukiwanie GIF-ów w czacie.
Limit:	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.10: Usługa zewnętrzna: Tenor GIF API

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ11
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 3.11: Usługa zewnętrzna: Where the ISS at?

3.5 Analiza ryzyka

Rozdział 4

Analiza wymagań

Niniejszy rozdział zawiera analizę wymagań postawionych systemowi.

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

W niniejszej pracy przyjęto następującą interpretację priorytetów MoSCoW:

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

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

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

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

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

4.1 Przypadki użycia

W niniejszym rozdziale przedstawiono przypadki użycia systemu. W pierwszej kolejności scharakteryzowano aktorów. Następnie zaprezentowano diagramy przypadków użycia, które ilustrują relacje między aktorami a funkcjami aplikacji. Jako ostatni punkt rozdziału przedstawiono uszczegółowienie przypadków użycia w postaci scenariuszy.

4.1.1 Aktorzy

Użytkownik - Reprezentuje każdą osobę korzystającą z aplikacji.

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

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 multimedialnych użytkowników.

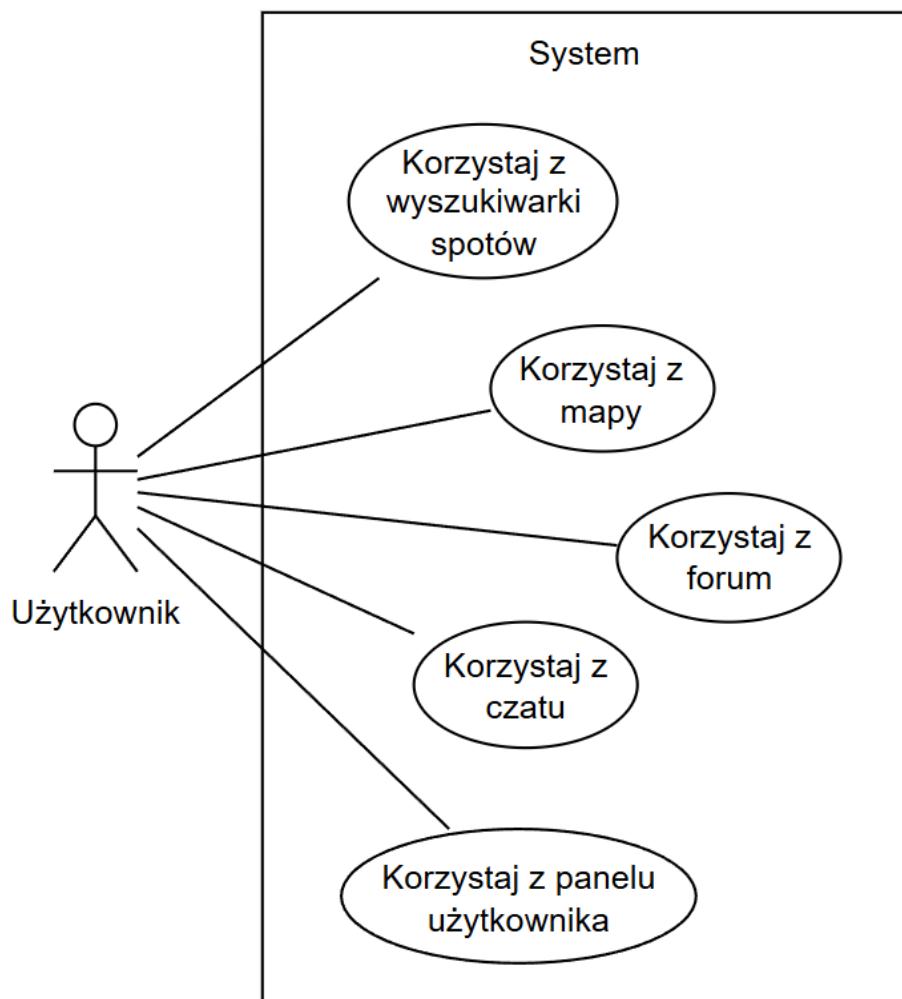
Usługa do wyświetlania mapy - zewnętrzne API dostarczające kafelki map, nawigację oraz dane geolokalizacyjne.

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.

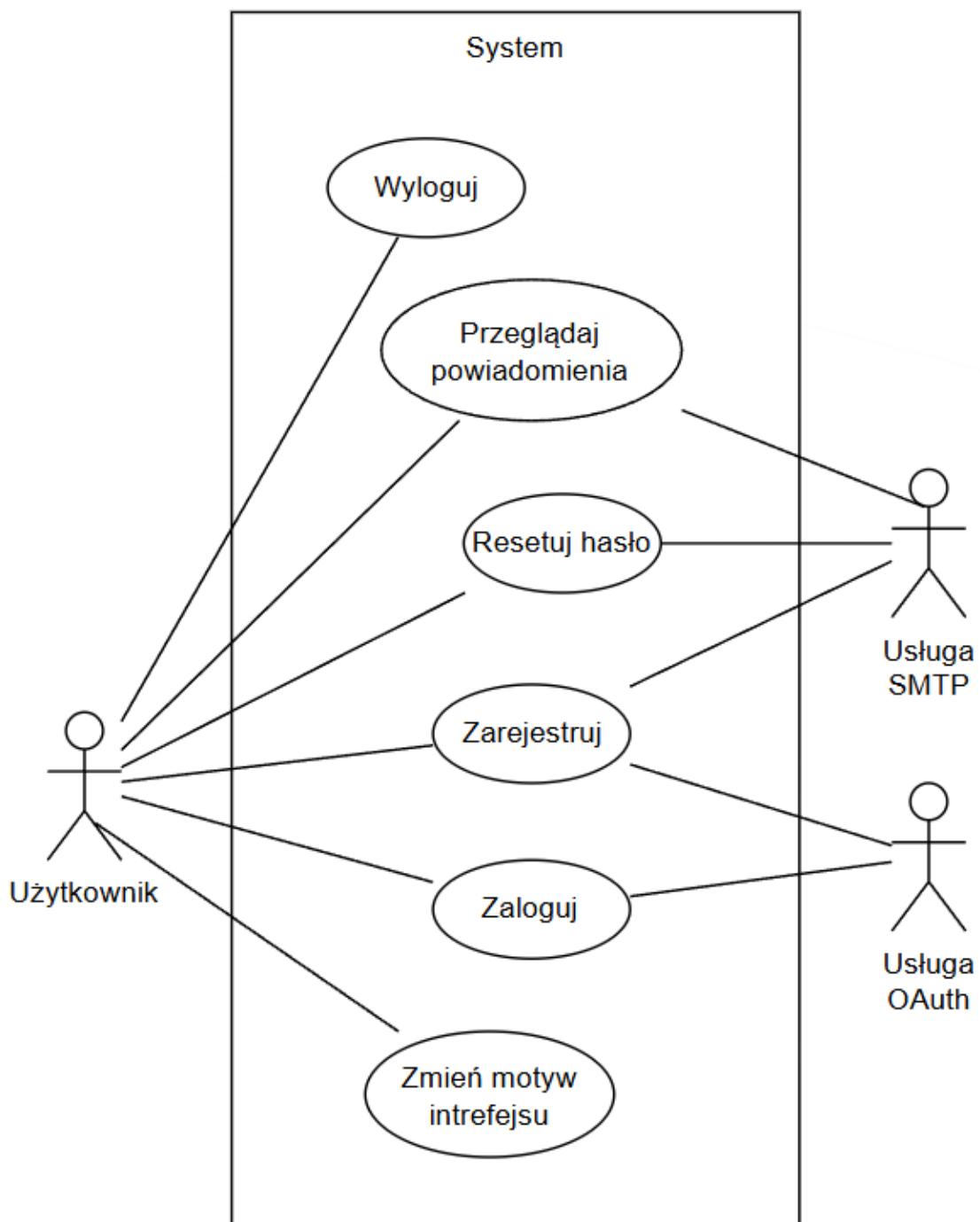
Usługa do określania strefy czasowej - usługa ustalająca strefę czasową spoden podstawie jego współrzędnych geograficznych.

4.1.2 Diagramy przypadków użycia

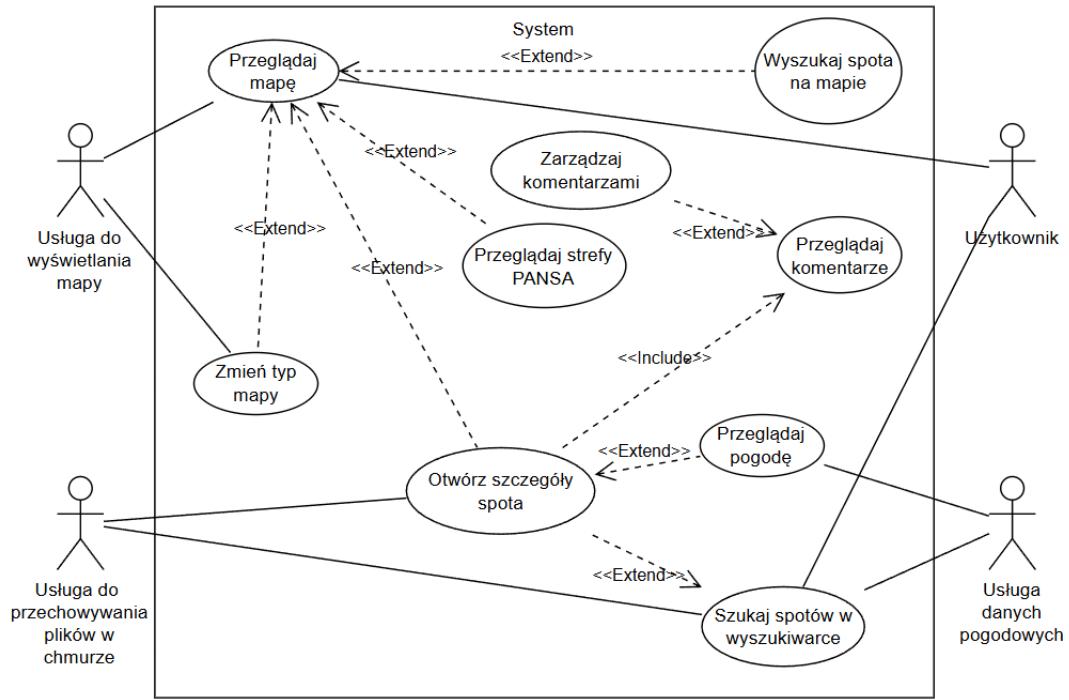


Rysunek 4.1: Wysokopoziomowy diagram przypadków użycia

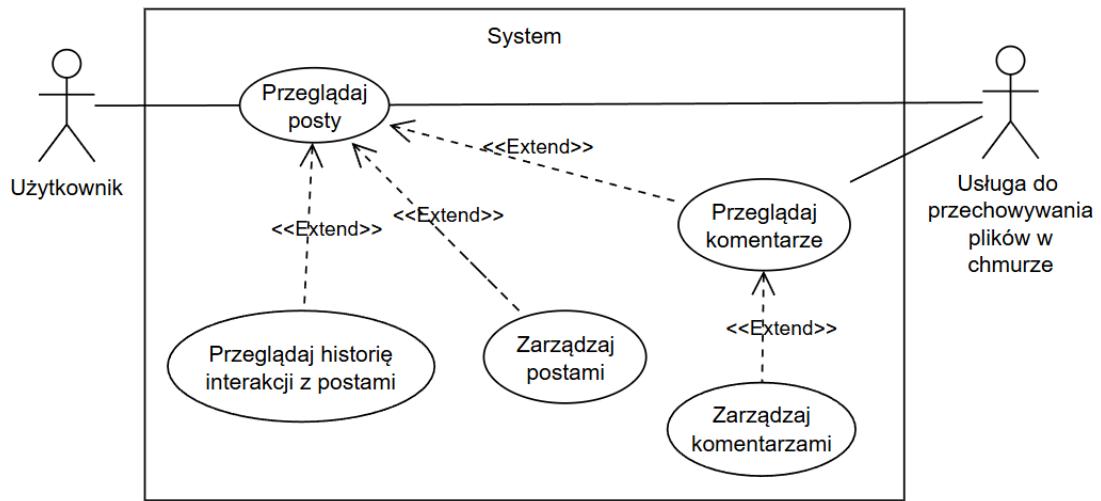
Diagram przedstawia podstawowe interakcje użytkownika z systemem. Na jego podstawie zespół projektowy podzielił architekturę aplikacji na 5 modułów: wyszukiwarkę spotów, mapę spotów, forum, czat oraz profil użytkownika. Pozostałe diagramy są bardziej szczegółowe.



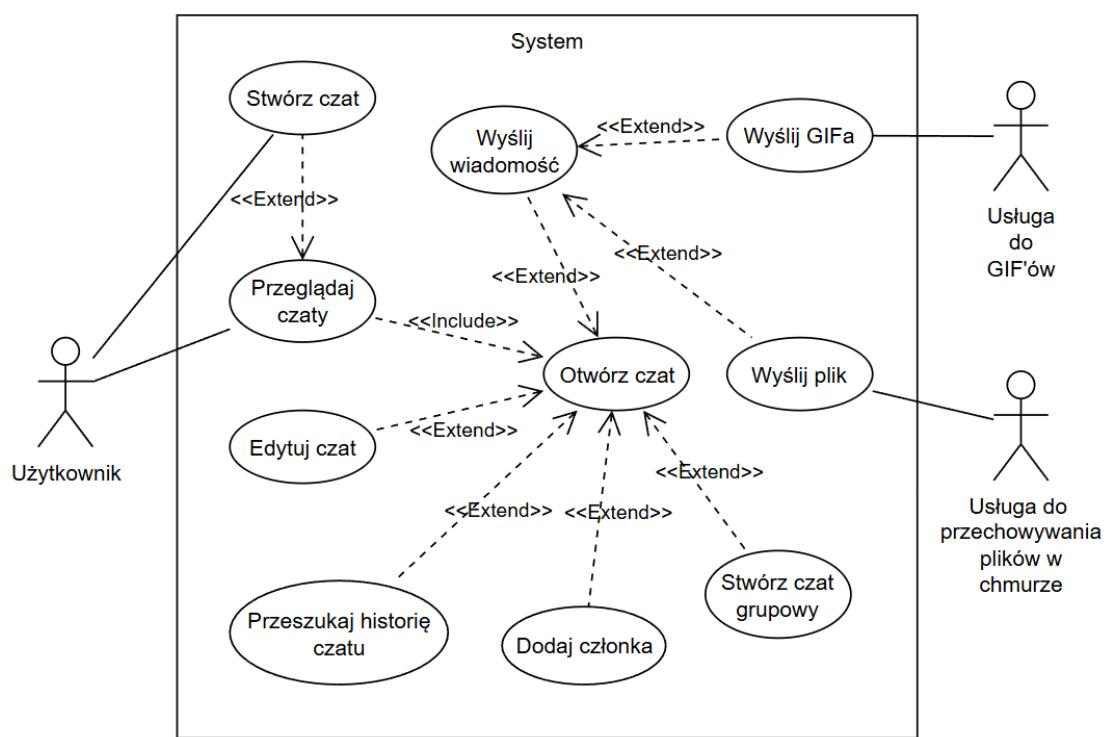
Rysunek 4.2: Ogólny diagram przypadków użycia



Rysunek 4.3: Diagram przypadków użycia wyszukiwarki spotów oraz mapy



Rysunek 4.4: Diagram przypadków użycia forum



Rysunek 4.5: Diagram przypadków użycia czatu



Rysunek 4.6: Diagram przypadków użycia profilu użytkownika

4.1.3 Scenariusze przypadków użycia

4.1.3.1 Scenariusze przypadków użycia – funkcje ogólne

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU1
Nazwa:	Rejestracja użytkownika
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik zakłada konto poprzez formularz rejestracji.

Warunki wstępne:	Użytkownik znajduje się na stronie z formularzem rejestracji.
Warunki końcowe:	Użytkownik posiada konto w systemie.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wypełnia formularz rejestracyjny. 2. Użytkownik naciska przycisk rejestracji. 3. System tworzy konto użytkownika. 4. System loguje użytkownika i przenosi go na ostatnio doowiedzoną podstronę.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 1a. Podane dane są niepoprawne – system wyświetla komunikat o błędzie oraz podświetla pola wymagające poprawy. 2a. Nazwa użytkownika jest już zajęta – system wyświetla komunikat o błędzie. 2b. Adres email jest już zajęty – system wyświetla komunikat o błędzie.

Tabela 4.1: Scenariusz przypadku użycia: Rejestracja użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU2
Nazwa:	Logowanie użytkownika
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik loguje się do systemu, podając login i hasło.
Warunki wstępne:	Użytkownik znajduje się na stronie logowania.

Warunki końcowe:	Użytkownik jest zalogowany i przeniesiony na ostatnio do-wiedzoną podstronę.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wypełnia formularz logowania. 2. Użytkownik naciska przycisk logowania. 3. System loguje użytkownika i przenosi go na ostatnio do-wiedzoną podstronę.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> <li value="2">2a. Podane dane są niepoprawne – system wyświetla komunikat o błędzie.

Tabela 4.2: Scenariusz przypadku użycia: Logowanie użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU3
Nazwa:	Resetowanie hasła
Priorytet:	S
Aktorzy:	Użytkownik, Usługa SMTP
Opis:	Użytkownik inicjuje reset hasła, aby odzyskać dostęp do konta.
Warunki wstępne:	Użytkownik znajduje się na ekranie resetu hasła.
Warunki końcowe:	Użytkownik otrzymuje wiadomość e-mail z linkiem do ustalenia nowego hasła.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wpisuje adres e-mail powiązany z kontem. 2. Użytkownik zatwierdza żądanie resetu hasła. 3. System generuje token resetu hasła. 4. System wysyła e-mail z linkiem do zmiany hasła.

Alternatywne przepływy zdarzeń:	<p>2a. Nie istnieje konto dla podanego adresu – system wyświetla komunikat o błędzie.</p> <p>4a. Występuje błąd połączenia z usługą SMTP – system informuje użytkownika o problemie technicznym.</p>
--	--

Tabela 4.3: Scenariusz przypadku użycia: Resetowanie hasła

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU4
Nazwa:	Zmiana hasła w ustawieniach konta
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik zmienia hasło do konta z poziomu ustawień profilu.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się na ekranie zmiany danych konta.
Warunki końcowe:	Hasło do konta użytkownika zostało zaktualizowane.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik wpisuje aktualne hasło. Użytkownik wpisuje nowe hasło i powtarza je. Użytkownik zatwierdza formularz zmiany hasła. System zapisuje nowe hasło i informuje o powodzeniu operacji.

Alternatywne przepływy zdarzeń:	3a. Aktualne hasło jest nieprawidłowe – system wyświetla komunikat i nie zapisuje zmian. 3b. Nowe hasło nie spełnia wymagań bezpieczeństwa – system informuje o błędzie i podświetla pola do poprawy.
--	--

Tabela 4.4: Scenariusz przypadku użycia: Zmiana hasła w ustawieniach konta

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU5
Nazwa:	Wylogowanie użytkownika
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik wylogowuje się z aplikacji.
Warunki wstępne:	Użytkownik jest zalogowany.
Warunki końcowe:	Sesja użytkownika została zakończona, użytkownik widzi stronę główną dla niezalogowanych.
Główny przepływ zdarzeń:	1. Użytkownik wybiera opcję wylogowania z menu. 2. System unieważnia token dostępu użytkownika. 3. System przenosi użytkownika na stronę główną aplikacji.
Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów.

Tabela 4.5: Scenariusz przypadku użycia: Wylogowanie użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU6
Nazwa:	Przeglądanie powiadomień
Priorytet:	W
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda listę powiadomień.
Warunki wstępne:	Użytkownik jest na ekranie centra powiadomień.
Warunki końcowe:	Powiadomienia zostały wyświetlane, a wybrane oznaczone jako przeczytane.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> System wyświetla powiadomienia w odwróconym porządku chronologicznym. Użytkownik otwiera wybrane powiadomienie. System oznacza powiadomienie jako przeczytane i ewentualnie przenosi użytkownika do powiązanego widoku.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> System nie może pobrać powiadomień (błąd serwera) – użytkownik otrzymuje komunikat o błędzie i może spróbować ponownie.

Tabela 4.6: Scenariusz przypadku użycia: Przeglądanie powiadomień

4.1.3.2 Scenariusze przypadków użycia dla wyszukiwarki

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU7
Nazwa:	Wyszukiwanie spota w globalnej wyszukiwarce
Priorytet:	S

Aktorzy:	Użytkownik, Usługa do wyświetlania mapy, Usługa do pogody
Opis:	Użytkownik wyszukuje spoty za pomocą globalnej wyszukiwarki w aplikacji.
Warunki wstępne:	Użytkownik znajduje się na stronie głównej z wyszukiwarką.
Warunki końcowe:	Użytkownik otrzymuje listę znalezionych spotów.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wpisuje frazę w globalnej wyszukiwarce. 2. System wyszukuje spoty spełniające kryteria. 3. System wyświetla listę wyników.
Alternatywne przepływy zdarzeń:	<p>3a. System informuje o braku wyników spełniających kryteria.</p>

Tabela 4.7: Scenariusz przypadku użycia: Wyszukiwanie spota w globalnej wyszukiwarce

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU8
Nazwa:	Przejście do spota na mapie z wyszukiwarki
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik przechodzi z wyników wyszukiwarki do widoku mapy ustalonego na konkretny spot.
Warunki wstępne:	Wyświetlona jest lista wyników wyszukiwania spotów.

Warunki końcowe:	Mapa jest przybliżona do wybranego spota, a jego szczegóły są dostępne.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera spota z listy wyników. 2. System przełącza widok na moduł mapy. 3. System ustawia mapę na lokalizację spota i otwiera jego szczegóły.
Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów.

Tabela 4.8: Scenariusz przypadku użycia: Przejście do spota na mapie z wyszukiwarki

4.1.3.3 Scenariusze przypadków użycia dla mapy

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU9
Nazwa:	Przeglądanie mapy spotów
Priorytet:	S
Aktorzy:	Użytkownik, Usługa do wyświetlania mapy
Opis:	Użytkownik przegląda mapę spotów.
Warunki wstępne:	Użytkownik znajduje się w module mapy.
Warunki końcowe:	Mapa ze spotami została wyświetlona, a użytkownik może przybliżać, oddalać i przesuwać widok.

Główny przepływ zdarzeń:	1. System inicjuje widok mapy z domyślnym obszarem. 2. System pobiera listę spotów. 3. System rysuje znaczniki spotów na mapie.
Alternatywne przepływy zdarzeń:	2a. Usługa mapy jest niedostępna – system wyświetla komunikat o błędzie.

Tabela 4.9: Scenariusz przypadku użycia: Przeglądanie mapy spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU10
Nazwa:	Otwarcie szczegółów spota
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik otwiera widok szczegółów wybranego spota.
Warunki wstępne:	Użytkownik widzi mapę spotów.
Warunki końcowe:	Wyświetlony został widok szczegółów spota z podstawowymi informacjami oraz jego lokalizacją na mapie.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik wybiera spota z mapy. System pobiera dane szczegółowe spota (informacje opisowe, lokalizacja). System otwiera widok szczegółów spota.

Alternatywne przepływy zdarzeń:	<p>2a. Spot nie istnieje (został usunięty lub ukryty) – system informuje użytkownika i powraca do poprzedniego widoku.</p> <p>2b. Wystąpił błąd podczas pobierania danych spota – system wyświetla komunikat o błędzie i umożliwia ponowną próbę.</p>
--	---

Tabela 4.10: Scenariusz przypadku użycia: Otwarcie szczegółów spota

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU11
Nazwa:	Przeglądanie komentarzy do spota
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik czyta komentarze pod wybranym spotem.
Warunki wstępne:	Wyświetlany jest widok szczegółów spota.
Warunki końcowe:	Lista komentarzy do spota została wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> System pobiera komentarze powiązane ze spotem. System wyświetla komentarze w kolejności chronologicznej lub według popularności. Użytkownik przewija listę komentarzy.
Alternatywne przepływy zdarzeń:	<p>1a. Spot nie ma jeszcze komentarzy – system wyświetla odpowiednią informację.</p>

Tabela 4.11: Scenariusz przypadku użycia: Przeglądanie komentarzy do spota

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU12
Nazwa:	Przeglądanie pogody na spocie
Priorytet:	S
Aktorzy:	Użytkownik, Usługa danych pogodowych
Opis:	Użytkownik sprawdza prognozę pogody dla lokalizacji spota.
Warunki wstępne:	Wyświetlany jest widok szczegółów spota.
Warunki końcowe:	Prognoza pogody dla spota została wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik otwiera zakładkę pogody. System wysyła zapytanie do usługi pogodowej z lokalizacją spota. System odbiera prognozę i prezentuje ją (temperatura, prędkość wiatru, opady).
Alternatywne przepływy zdarzeń:	2a. Usługa pogodowa jest niedostępna – system wyświetla komunikat o braku danych pogodowych.

Tabela 4.12: Scenariusz przypadku użycia: Przeglądanie pogody na spocie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU13
Nazwa:	Wyszukiwanie spota na mapie
Priorytet:	C
Aktorzy:	Użytkownik

Opis:	Użytkownik wyszukuje spota po nazwie korzystając z pola wyszukiwania na mapie.
Warunki wstępne:	Użytkownik widzi mapę spotów.
Warunki końcowe:	Mapa zostaje ustawiona na wybranego spota lub listę dopasowań.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wpisuje frazę w polu wyszukiwania na mapie. 2. System podpowiada listę pasujących spotów. 3. Użytkownik wybiera spota z listy. 4. System przenosi użytkownika na mapie do wybranego spota.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> <li value="2">2a. Brak wyników dla podanej frazy – system informuje użytkownika o braku dopasowań.

Tabela 4.13: Scenariusz przypadku użycia: Wyszukiwanie spota na mapie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU14
Nazwa:	Zmiana typu mapy
Priorytet:	W
Aktorzy:	Użytkownik
Opis:	Użytkownik zmienia typ mapy (np. standardowa, satelitarna, hybrydowa).
Warunki wstępne:	Użytkownik jest na ekranie mapy.

Warunki końcowe:	Mapa jest wyświetlana w wybranym typie.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera ustawienia widoku mapy. 2. Użytkownik wybiera typ mapy z dostępnej listy. 3. System przełącza widok mapy na wybrany typ.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 3a. Wybrany typ mapy nie jest dostępny (błąd usługi mapowej) – system przywraca poprzedni typ i informuje o błędzie.

Tabela 4.14: Scenariusz przypadku użycia: Zmiana typu mapy

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU15
Nazwa:	Przeglądanie stref PANSA
Priorytet:	W
Aktorzy:	Użytkownik, Usługa do wyświetlania mapy
Opis:	Użytkownik wyświetla na mapie strefy przestrzeni powietrznej PANSA .
Warunki wstępne:	Użytkownik ma otwarty moduł mapy.
Warunki końcowe:	Strefy PANSA zostały zwizualizowane na mapie.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik włącza warstwę „Strefy PANSA”. 2. System pobiera dane o strefach. 3. System nakłada kontury stref na mapę.

Alternatywne przepływy zdarzeń:	2a. Dane o strefach są chwilowo niedostępne – system komunikuje problem i nie włącza warstwy.
--	---

Tabela 4.15: Scenariusz przypadku użycia: Przeglądanie stref [PANSA](#)

4.1.3.4 Scenariusze przypadków użycia dla czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU16
Nazwa:	Utworzenie prywatnego czatu
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik tworzy prywatną konwersację z innym użytkownikiem.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w zakładce społeczność.
Warunki końcowe:	Nowy czat prywatny został utworzony i wyświetlony użytkownikowi.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik wybiera opcję utworzenia nowego czatu. System tworzy nowy czat (jeśli nie istnieje). System otwiera widok nowego czatu.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> a. Taki czat już istnieje – system zamiast tworzyć nowy, otwiera istniejącą konwersację.

Tabela 4.16: Scenariusz przypadku użycia: Utworzenie prywatnego czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU17
Nazwa:	Otworzenie czatu
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik otwiera wybrany czat, aby wyświetlić historię rozmowy i móc wysyłać kolejne wiadomości.
Warunki wstępne:	Użytkownik jest zalogowany i widzi listę swoich czatów lub otrzymał powiadomienie prowadzące do czatu.
Warunki końcowe:	Wybrany czat został otworzony, a historia rozmowy jest wiadoma dla użytkownika.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik wybiera czat z listy czatów lub z powiadomienia. System pobiera dane czatu (uczestników, ostatnie wiadomości). System oznacza nieprzeczytane wiadomości na czacie jako przeczytane. System wyświetla widok czatu wraz z historią rozmowy.
Alternatywne przepływy zdarzeń:	<p>2a. Czat nie jest już dostępny (np. został usunięty lub użytkownik utracił do niego dostęp) – system wyświetla komunikat o braku dostępu i powraca do listy czatów.</p> <p>2b. Wystąpił błąd podczas pobierania danych czatu – system wyświetla komunikat o błędzie i umożliwia ponowną próbę.</p>

Tabela 4.17: Scenariusz przypadku użycia: Otworzenie czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU18
Nazwa:	Utworzenie czatu grupowego
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik tworzy nowy czat grupowy z kilkoma uczestnikami.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się na dowolnym czasie prywatnym.
Warunki końcowe:	Czat grupowy został utworzony i wyświetlony na ekranie.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję utworzenia czatu grupowego. 2. Użytkownik wybiera uczestników grupy. 3. Użytkownik zatwierdza utworzenie czatu. 4. System tworzy czat grupowy i dodaje do niego wskazanych użytkowników. 5. System otwiera widok nowego czatu grupowego.
Alternatywne przepływy zdarzeń:	<p>3a. System nie może utworzyć czatu – aplikacja informuje o błędzie.</p>

Tabela 4.18: Scenariusz przypadku użycia: Utworzenie czatu grupowego

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU19
Nazwa:	Przeglądanie listy czatów
Priorytet:	S

Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda listę swoich czatów prywatnych i grupowych.
Warunki wstępne:	Użytkownik jest zalogowany i otwiera moduł czatu.
Warunki końcowe:	Lista czatów użytkownika została wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. System pobiera listę czatów użytkownika. 2. System wyświetla listę czatów z podstawowymi informacjami. 3. Użytkownik wybiera czat z listy. 4. System otwiera widok wybranego czatu.
Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów.

Tabela 4.19: Scenariusz przypadku użycia: Przeglądanie listy czatów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU20
Nazwa:	Wysyłanie wiadomości na czacie
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik wysyła wiadomość tekstową na czacie.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku konkretnego czatu.

Warunki końcowe:	Nowa wiadomość jest zapisana i widoczna w historii czatu.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wpisuje treść wiadomości. 2. Użytkownik wysyła wiadomość. 3. System zapisuje wiadomość i dostarcza ją do uczestników czatu. 4. System wyświetla wiadomość na liście wiadomości.
Alternatywne przepływy zdarzeń:	<p>2a. Treść wiadomości jest pusta – system blokuje wysłanie i pozostaje w tym samym widoku.</p>

Tabela 4.20: Scenariusz przypadku użycia: Wysyłanie wiadomości na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU21
Nazwa:	Wysyłanie GIF-a na czacie
Priorytet:	S
Aktorzy:	Użytkownik, Usługa GIF-ów
Opis:	Użytkownik wysyła animację GIF w konwersacji czatowej.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku czatu.
Warunki końcowe:	Wybrany GIF został dodany jako wiadomość w czacie.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję dodania GIF-a. 2. System otwiera okno wyszukiwarki GIF-ów. 3. Użytkownik wybiera lub wyszukuje GIF-a. 4. Użytkownik zatwierdza wysłanie GIF-a. 5. System dodaje GIF-a jako wiadomość na czacie.
Alternatywne przepływy zdarzeń:	<p>2a. Usługa GIF-ów jest niedostępna – system informuje o braku możliwości wysłania GIF-a.</p>

Tabela 4.21: Scenariusz przypadku użycia: Wysyłanie GIF-a na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU22
Nazwa:	Wysyłanie pliku na czacie
Priorytet:	S
Aktorzy:	Użytkownik, Usługa do przechowywania plików w chmurze
Opis:	Użytkownik wysyła plik (np. zdjęcie, film) na czacie.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku czatu.
Warunki końcowe:	Plik został zapisany w chmurze i powiązany z wiadomością na czacie.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję dodania pliku. 2. Użytkownik wybiera plik z urządzenia. 3. System przesyła plik do usługi przechowywania w chmurze. 4. System tworzy wiadomość z odnośnikiem do pliku. 5. System wyświetla wiadomość na liście czatu.

Alternatywne przepływy zdarzeń:	3a. Przesyłanie pliku nie powiodło się – system informuje użytkownika i umożliwia ponowną próbę.
--	--

Tabela 4.22: Scenariusz przypadku użycia: Wysyłanie pliku na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU23
Nazwa:	Edycja ustawień czatu
Priorytet:	C
Aktorzy:	Użytkownik
Opis:	Użytkownik modyfikuje ustawienia czatu (np. nazwę, avatar, tryb powiadomień).
Warunki wstępne:	Użytkownik jest zalogowany i ma uprawnienia do edycji danego czatu.
Warunki końcowe:	Zaktualizowane ustawienia czatu są zapisane i widoczne dla uczestników.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik otwiera panel ustawień czatu. Użytkownik wprowadza zmiany (np. nazwę, opis, avatar). Użytkownik zapisuje zmiany. System waliduje dane i aktualizuje konfigurację czatu.
Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów poza walidacją pól.

Tabela 4.23: Scenariusz przypadku użycia: Edycja ustawień czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU24
Nazwa:	Dodanie członka do czatu grupowego
Priorytet:	C
Aktorzy:	Użytkownik
Opis:	Użytkownik dodaje nowego uczestnika do czatu grupowego.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w czacie grupowym.
Warunki końcowe:	Nowy uczestnik został dodany do czatu grupowego.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik otwiera listę uczestników czatu grupowego. Użytkownik wybiera opcję dodania nowego członka. Użytkownik wskazuje użytkownika do dodania i zatwierdza wybór. System dodaje wskazanego użytkownika do czatu grupowego.
Alternatywne przepływy zdarzeń:	3a. Operacja nie powiodła się – system informuje o błędzie.

Tabela 4.24: Scenariusz przypadku użycia: Dodanie członka do czatu grupowego

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU25
Nazwa:	Przeszukiwanie historii czatu
Priorytet:	W
Aktorzy:	Użytkownik

Opis:	Użytkownik wyszukuje konkretne wiadomości w historii czatu.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku czatu.
Warunki końcowe:	Wiadomości spełniające kryteria wyszukiwania zostały wyświetlane.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera pole wyszukiwania historii w czacie. 2. Użytkownik wpisuje frazę lub filtr (np. zakres dat, autor). 3. System filtryuje wiadomości zgodnie z kryteriami. 4. System prezentuje listę dopasowanych fragmentów rozmowy.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 4a. Nie znaleziono wiadomości spełniających kryteria – system informuje o braku wyników wyszukwania.

Tabela 4.25: Scenariusz przypadku użycia: Przeszukiwanie historii czatu

4.1.3.5 Scenariusze przypadków użycia dla forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU26
Nazwa:	Przeglądanie postów na forum
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda listę postów na forum z możliwością sortowania wyników.

Warunki wstępne:	Użytkownik znajduje się w module forum.
Warunki końcowe:	Lista postów forum jest wyświetlona, a użytkownik może przechodzić do szczegółów wybranego posta.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik przechodzi do widoku listy postów forum. 2. System pobiera listę postów i domyślnie wyświetla je w kolejności od najnowszych. 3. Użytkownik wybiera sposób sortowania listy. 4. System aktualizuje listę postów zgodnie z wybranym kryterium sortowania. 5. Użytkownik wybiera post, który chce przeczytać. 6. System otwiera szczegółowy widok wybranego posta.
Alternatywne przepływy zdarzeń:	<p>6a. System nie może pobrać szczegółów posta – system wyświetla komunikat o błędzie.</p>

Tabela 4.26: Scenariusz przypadku użycia: Przeglądanie postów na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU27
Nazwa:	Wyszukiwanie postów na forum
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik wyszukuje posty na forum na podstawie tytułu, kategorii, tagów oraz autora.
Warunki wstępne:	Użytkownik znajduje się w module forum lub na stronie wyszukiarki postów.

Warunki końcowe:	Lista postów spełniających zadane kryteria wyszukiwania jest wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera panel wyszukiwania postów na forum. 2. Użytkownik określa kryteria wyszukiwania (np. tytuł, kategoria, tagi, autor). 3. Użytkownik uruchamia wyszukiwanie. 4. System filtryuje posty zgodnie z podanymi kryteriami i wyświetla listę wyników.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 4a. Brak postów spełniających zadane kryteria – system wyświetla informację o braku wyników. 4b. Wystąpił błąd podczas wyszukiwania – system wyświetla komunikat o błędzie i umożliwia ponowną próbę.

Tabela 4.27: Scenariusz przypadku użycia: Wyszukiwanie postów na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU28
Nazwa:	Dodanie posta na forum
Priorytet:	S
Aktorzy:	Użytkownik, Usługa do przechowywania plików w chmurze
Opis:	Użytkownik publikuje nowy post na forum, określając jego treść, kategorię, tagi oraz opcjonalne załączniki.
Warunki wstępne:	Użytkownik znajduje się w module forum.
Warunki końcowe:	Nowy post jest poprawnie zapisany i widoczny na forum.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję dodania nowego posta. 2. Użytkownik wpisuje tytuł i treść posta. 3. Użytkownik wybiera kategorię posta. 4. (Opcjonalnie) Użytkownik wybiera tagi przypisane do posta. 5. (Opcjonalnie) Użytkownik dodaje załączniki (zdjęcia/filmy) do posta. 6. Użytkownik publikuje posta. 7. System zapisuje posta (oraz poprawne załączniki w chmurze) i wyświetla go na liście postów.
Alternatywne przepływy zdarzeń:	<p>6a. Załącznik nie może zostać zapisany lub nie spełnia wymagań (np. zbyt duży rozmiar, nieobsługiwany format) – system informuje użytkownika o błędzie, blokuje publikację posta i wymaga usunięcia lub podmiany problematycznego pliku.</p> <p>6b. Formularz zawiera błędne lub niekompletne dane (np. brak tytułu lub treści) – system wyświetla komunikat i prosi o poprawę danych przed publikacją.</p>

Tabela 4.28: Scenariusz przypadku użycia: Dodanie posta na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU29
Nazwa:	Dodanie komentarza na forum
Priorytet:	S
Aktorzy:	Użytkownik

Opis:	Użytkownik dodaje komentarz pod postem na forum, opcjonalnie z załącznikami (zdjęcia/filmy).
Warunki wstępne:	Użytkownik jest zalogowany i widzi szczegóły posta.
Warunki końcowe:	Nowy komentarz został zapisany i jest widoczny pod postem.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wpisuje treść komentarza w formularzu pod postem. 2. (Opcjonalnie) Użytkownik dodaje załączniki (zdjęcia/filmy) do komentarza. 3. Użytkownik publikuje komentarz. 4. System zapisuje komentarz (oraz poprawne załączniki) i odświeża listę komentarzy.
Alternatywne przepływy zdarzeń:	<p>3a. Treść komentarza lub załączniki są niepoprawne (np. naruszają validację) – system wyświetla komunikat o błędzie i blokuje publikację do czasu poprawy danych.</p>

Tabela 4.29: Scenariusz przypadku użycia: Dodanie komentarza na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU30
Nazwa:	Przeglądanie historii interakcji z postami
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda historię swoich aktywności na forum (dodane posty, komentarze, reakcje).

Warunki wstępne:	Użytkownik jest zalogowany.
Warunki końcowe:	Lista interakcji użytkownika z postami jest wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik przechodzi do sekcji historii aktywności. System pobiera historię interakcji użytkownika. System wyświetla listę interakcji z możliwością filtrowania.
Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów.

Tabela 4.30: Scenariusz przypadku użycia: Przeglądanie historii interakcji z postami

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU31
Nazwa:	Zarządzanie komentarzami na forum
Priorytet:	C
Aktorzy:	Użytkownik
Opis:	Użytkownik zarządza komentarzami pod postami forum (edykcja, usuwanie, zgłaszanie komentarzy innych użytkowników).
Warunki wstępne:	Użytkownik jest zalogowany i ma dostęp do danego wątku forum.
Warunki końcowe:	Komentarze zostały zaktualizowane zgodnie z działaniami użytkownika.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera widok komentarzy pod postem. 2. Użytkownik wybiera komentarz i odpowiednią akcję (edytacja, usunięcie, zgłoszenie). 3. System weryfikuje uprawnienia użytkownika oraz zgodność akcji z jego rolą. 4. System wykonuje wybraną akcję (np. zapisuje zmiany, usuwa komentarz lub przygotowuje zgłoszenie) i aktualizuje widok.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 2a. Użytkownik próbuje zgłosić własny komentarz – system blokuje operację i informuje, że nie można zgłaszać własnych treści. 3a. Użytkownik nie ma wymaganych uprawnień do wykonania wybranej akcji – system blokuje operację i informuje o braku uprawnień.

Tabela 4.31: Scenariusz przypadku użycia: Zarządzanie komentarzami na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU32
Nazwa:	Zgłoszenie komentarza naruszającego regulamin
Priorytet:	C
Aktorzy:	Użytkownik
Opis:	Użytkownik zgłasza komentarz na forum jako naruszający regulamin.
Warunki wstępne:	Użytkownik widzi komentarz w aplikacji.

Warunki końcowe:	Zgłoszenie komentarza zostało zapisane i trafiło do kolejki moderacyjnej.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję „Zgłoś komentarz”. 2. Użytkownik wybiera kategorię naruszenia, podaje szczegóły i potwierdza zgłoszenie. 3. System zapisuje zgłoszenie i wiąże je z komentarzem oraz zgłaszającym użytkownikiem.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 1a. Użytkownik próbuje zgłosić własny komentarz – system blokuje operację i wyświetla komunikat, że nie można zgłaszać własnych treści. 3a. Komentarz został już wcześniej zgłoszony – system informuje użytkownika, że komentarz znajduje się już w kolejce moderacyjnej i nie zapisuje kolejnego zgłoszenia.

Tabela 4.32: Scenariusz przypadku użycia: Zgłoszenie komentarza naruszającego regulamin

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU33
Nazwa:	Zgłoszenie posta na forum
Priorytet:	C
Aktorzy:	Użytkownik
Opis:	Użytkownik zgłasza post forum jako naruszający regulamin lub tematykę.
Warunki wstępne:	Wyświetlony jest widok posta na forum.

Warunki końcowe:	Zgłoszenie posta zostało zapisane
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję „Zgłoś post”. 2. Użytkownik wybiera kategorię naruszenia, podaje szczegóły i potwierdza zgłoszenie. 3. System zapisuje zgłoszenie i oznacza post jako zgłoszony.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 1a. Użytkownik próbuje zgłosić własny post – system blokuje operację i wyświetla komunikat, że nie można zgłaszać własnych treści. 3a. Post został już wcześniej zgłoszony – system informuje użytkownika, że post jest już zgłoszony i nie zapisuje kolejnego zgłoszenia.

Tabela 4.33: Scenariusz przypadku użycia: Zgłoszenie posta na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU34
Nazwa:	Przeglądanie komentarzy pod postem
Priorytet:	S
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda komentarze dodane pod wybranym postem na forum z możliwością zmiany kolejności ich wyświetlania.
Warunki wstępne:	Wyświetlany jest szczegółowy widok posta na forum.
Warunki końcowe:	Lista komentarzy powiązanych z postem została wyświetlona zgodnie z wybranym kryterium sortowania.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. System pobiera komentarze powiązane z wybranym postem i domyślnie wyświetla je w kolejności od najnowszych. 2. Użytkownik wybiera sposób sortowania komentarzy. 3. System aktualizuje listę komentarzy zgodnie z wybranym kryterium sortowania. 4. Użytkownik przewija listę komentarzy i zapoznaje się z ich treścią.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 1a. Post nie ma jeszcze komentarzy – system wyświetla informację o braku komentarzy. 1b. Wystąpił błąd podczas pobierania komentarzy – system wyświetla komunikat o błędzie i umożliwia ponowną próbę.

Tabela 4.34: Scenariusz przypadku użycia: Przeglądanie komentarzy pod postem

4.1.3.6 Scenariusze przypadków użycia dla panelu użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU35
Nazwa:	Dodanie spota w panelu użytkownika
Priorytet:	M
Aktorzy:	Użytkownik, Usługa do wyświetlania mapy, Usługa do przechowywania plików w chmurze
Opis:	Użytkownik dodaje nowy spot poprzez panel.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku panelu użytkownika.

Warunki końcowe:	Nowy spot został zapisany i jest widoczny na mapie oraz w panelu użytkownika.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję „Dodaj spota”. 2. Użytkownik uzupełnia podstawowe informacje o spocie (nazwa, opis, tagi). 3. Użytkownik wskazuje lokalizację spota na mapie. 4. Użytkownik dodaje zdjęcia/filmy do spota. 5. Użytkownik zapisuje spota. 6. System zapisuje dane spota (oraz pliki w chmurze) i aktualizuje mapę.
Alternatywne przepływy zdarzeń:	<p>3a. Podane dane wejściowe są niepoprawne – system wyświetla komunikat i zaznacza wymagające poprawy pola.</p>

Tabela 4.35: Scenariusz przypadku użycia: Dodanie spota w panelu użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU36
Nazwa:	Przeglądanie profilu użytkownika
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda swój profil (lista spotów, media, podstawowe dane).
Warunki wstępne:	Użytkownik jest zalogowany.
Warunki końcowe:	Wyświetlony jest widok profilu użytkownika wraz z jego zawartością.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera swój profil. 2. System pobiera dane profilu (informacje podstawowe, spoty, media). 3. System wyświetla dane w odpowiednich sekcjach (spoty, zdjęcia, filmy, komentarze).
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 2a. Wystąpił błąd podczas pobierania danych użytkownika – system wyświetla informację o błędzie.

Tabela 4.36: Scenariusz przypadku użycia: Przeglądanie profilu użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU37
Nazwa:	Przeglądanie profilu innego użytkownika
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik ogląda profil innego użytkownika (np. z mapy, forum lub społeczności).
Warunki wstępne:	Użytkownik jest zalogowany i ma dostęp do odnośnika do profilu innego użytkownika.
Warunki końcowe:	Profil innego użytkownika został wyświetlony.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera odnośnik do profilu innego użytkownika. 2. System pobiera dane profilu docelowego użytkownika. 3. System wyświetla profil (media, podstawowe informacje).

Alternatywne przepływy zdarzeń:	2a. Wystąpił błąd podczas pobierania danych użytkownika – system wyświetla informację o błędzie.
--	--

Tabela 4.37: Scenariusz przypadku użycia: Przeglądanie profilu innego użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU38
Nazwa:	Dodanie użytkownika do znajomych
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik wysyła lub akceptuje zaproszenie do znajomych.
Warunki wstępne:	Użytkownik jest zalogowany i przegląda profil innego użytkownika.
Warunki końcowe:	Relacja „znajomy” została utworzona lub zaproszenie czeka na akceptację.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik kliką przycisk „Dodaj do znajomych”. System sprawdza, czy relacja już istnieje. System tworzy nowe zaproszenie. System informuje o statusie o wysłaniu zaproszenia.
Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów.

Tabela 4.38: Scenariusz przypadku użycia: Dodanie użytkownika do znajomych

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU39
Nazwa:	Przeglądanie społeczności
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda społeczności, grupy lub listy znajomych powiązane z aplikacją.
Warunki wstępne:	Użytkownik jest zalogowany.
Warunki końcowe:	Lista społeczności lub znajomych została wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik przechodzi do sekcji społeczności. System pobiera listę społeczności i znajomych użytkownika. System wyświetla listę z możliwością przechodzenia do profili i czatów.
Alternatywne przepływy zdarzeń:	2a. Nie udało się pobrać danych – system wyświetla komunikat o błędzie.

Tabela 4.39: Scenariusz przypadku użycia: Przeglądanie społeczności

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU40
Nazwa:	Przeglądanie dodanych spotów
Priorytet:	M
Aktorzy:	Użytkownik, Usługa do wyświetlania mapy

Opis:	Użytkownik przegląda listę/siatkę spotów, które sam dodał.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku panelu użytkownika lub sekcji „Moje spoty”.
Warunki końcowe:	Lista dodanych spotów użytkownika została wyświetlona (np. na mapie i/lub w formie listy).
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik przechodzi do sekcji „Moje spoty”. 2. System pobiera listę spotów dodanych przez użytkownika. 3. System wyświetla listę spotów oraz znaczniki na mapie. 4. Użytkownik wybiera spota, aby przejść do jego szczegółów.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 2a. Użytkownik nie dodał jeszcze żadnego spota – system wyświetla komunikat i proponuje dodanie pierwszego spota.

Tabela 4.40: Scenariusz przypadku użycia: Przeglądanie dodanych spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU41
Nazwa:	Edycja danych użytkownika
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik modyfikuje swoje dane profilu (np. nazwę, opis, avatar).
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w widoku edycji profilu.

Warunki końcowe:	Zaktualizowane dane profilu są zapisane i widoczne w aplikacji.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera widok edycji profilu. 2. Użytkownik wprowadza zmiany w danych profilu. 3. Użytkownik zapisuje zmiany. 4. System waliduje dane i zapisuje zaktualizowany profil.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 4a. Dane są niepoprawne lub niekompletne – system wyświetla komunikat o błędzie i zaznacza pola do poprawy.

Tabela 4.41: Scenariusz przypadku użycia: Edycja danych użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU42
Nazwa:	Przeglądanie dodanych zdjęć do spotów
Priorytet:	M
Aktorzy:	Użytkownik, Usługa do przechowywania plików w chmurze
Opis:	Użytkownik przegląda wszystkie zdjęcia powiązane ze spotami, które dodał.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w sekcji mediów (np. „Moje zdjęcia”).
Warunki końcowe:	Lista lub galeria zdjęć powiązanych ze spotami użytkownika została wyświetlona.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera sekcję przeglądania zdjęć ze spotów. 2. System pobiera metadane zdjęć z usługi przechowywania plików. 3. System wyświetla galerię zdjęć z podstawowymi informacjami (np. nazwa spota, data dodania).
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 2a. Użytkownik nie dodał jeszcze zdjęć – system wyświetla informację o braku zdjęć. 2b. Nie udało się pobrać danych – system wyświetla komunikat o błędzie.

Tabela 4.42: Scenariusz przypadku użycia: Przeglądanie dodanych zdjęć do spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU43
Nazwa:	Przeglądanie dodanych filmów do spotów
Priorytet:	M
Aktorzy:	Użytkownik, Usługa do przechowywania plików w chmurze
Opis:	Użytkownik przegląda filmy powiązane ze spotami, które dał.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w sekcji mediów (np. „Moje filmy”).
Warunki końcowe:	Lista lub galeria filmów powiązanych ze spotami użytkownika została wyświetlona.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera sekcję przeglądania filmów ze spotów. 2. System pobiera metadane filmów z usługi przechowywania plików. 3. System wyświetla listę/galerię filmów z podstawowymi informacjami. 4. Użytkownik wybiera film, aby go odtworzyć.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> <li value="2">2a. Użytkownik nie dodał jeszcze filmów – system wyświetla informację o braku filmów. <li value="3">3a. Nie udało się pobrać filmów – system wyświetla komunikat o błędzie.

Tabela 4.43: Scenariusz przypadku użycia: Przeglądanie dodanych filmów do spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU44
Nazwa:	Przeglądanie dodanych komentarzy do spotów
Priorytet:	M
Aktorzy:	Użytkownik
Opis:	Użytkownik przegląda komentarze dodane do spotów, które sam utworzył.
Warunki wstępne:	Użytkownik jest zalogowany i otwiera sekcję komentarzy do swoich spotów.
Warunki końcowe:	Lista komentarzy do spotów użytkownika została wyświetlona.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik przechodzi do sekcji komentarzy do własnych spotów. 2. System pobiera komentarze powiązane ze spotami użytkownika. 3. System wyświetla komentarze (np. w kolejności chronologicznej).
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 2a. Żaden z spotów użytkownika nie ma komentarzy – system wyświetla odpowiednią informację. 3a. Nie udało się pobrać komentarzy – system wyświetla komunikat o błędzie.

Tabela 4.44: Scenariusz przypadku użycia: Przeglądanie dodanych komentarzy do spotów

4.2 Wymagania

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

W pracy wyróżniono następujące typy wymagań:

- **Wymagania ogólne** – opisują system na wysokim poziomie, z perspektywy celu biznesowego i użytkownika, bez wchodzenia w szczegóły techniczne. Określają, jakie główne problemy ma rozwiązywać system i jakie korzyści ma dostarczać interesariuszom.
- **Wymagania dziedzinowe** – wynikają ze specyfiki obszaru, w którym wykorzystywany jest system (domena problemu). Odzwierciedlają reguły biznesowe, ograniczenia prawne oraz ustaloną praktykę w danej dziedzinie i muszą być spełnione niezależnie od przyjętych rozwiązań technicznych.

- **Wymagania funkcjonalne** – opisują konkretne funkcje systemu widziane z perspektywy użytkownika lub innego aktora. Definiują, jakie operacje system ma umożliwiać, jakie dane przetwarzać oraz jak reagować na określone zdarzenia i scenariusze użycia.
- **Wymagania pozafunkcjonalne** – określają, *jak* system ma realizować swoje funkcje, a nie *co* ma robić. Obejmują m.in. wymagania dotyczące wydajności, bezpieczeństwa, niezawodności, użyteczności, skalowalności oraz utrzymywalności rozwiązania.
- **Wymagania dotyczące interfejsu z otoczeniem** – opisują sposób komunikacji systemu z innymi systemami, urządzeniami lub usługami zewnętrznymi. Określają format wymienianych danych, wykorzystywane protokoły, kierunek i częstotliwość wymiany informacji oraz wymagania dotyczące integracji.

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

4.2.1 Wymagania ogólne

4.2.1.1 Wymagania ogólne dla czatu

KARTA WYMAGANIA OGÓLNEGO DLA CZATU			
Identyfikator:	WOCZAT-01	Priorytet:	S
Nazwa:	Wysyłanie wiadomości na czacie		
Opis:	System umożliwia użytkownikowi wysyłanie wiadomości w ramach wybranego czatu, tak aby uczestnicy mogli przekazywać sobie wiedzę na temat dronów lub umawiać się na wspólne spotkania w danym spocie .		
Udziałowiec:	U3		

KARTA WYMAGANIA OGÓLNEGO DLA CZATU (cd.)	
Wymagania powiązane:	WFCZAT-01, WFCZAT-02, WFCZAT-03, WFCZAT-04, WFCZAT-06, WFCZAT-10, WPCZAT-01, WPCZAT-02, WPCZAT-04, WPCZAT-06

Tabela 4.45: Karta wymagania ogólnego dla czatu: Wysyłanie wiadomości na czacie

KARTA WYMAGANIA OGÓLNEGO DLA CZATU			
Identyfikator:	WOCZAT-02	Priorytet:	S
Nazwa:	Edycja czatu		
Opis:	System umożliwia użytkownikowi wprowadzanie podstawowych zmian w konfiguracji czatu, takich jak nazwa czatu czy skład uczestników.		
Udziałowiec:	U3		
Wymagania powiązane:	WFCZAT-08, WFCZAT-09, WFCZAT-11, WPCZAT-01, WPCZAT-02		

Tabela 4.46: Karta wymagania ogólnego dla czatu: Edycja czatu

KARTA WYMAGANIA OGÓLNEGO DLA CZATU			
Identyfikator:	WOCZAT-03	Priorytet:	S
Nazwa:	Przeglądanie historii czatu		
Opis:	System udostępnia użytkownikowi możliwość przeglądania wcześniejszych wiadomości na czacie, tak aby mógł wracać do poprzednich rozmów.		
Udziałowiec:	U3		
Wymagania powiązane:	WFCZAT-07, WFCZAT-12, WPCZAT-01, WPCZAT-02, WPCZAT-03, WPCZAT-05		

Tabela 4.47: Karta wymagania ogólnego dla czatu: Przeglądanie historii czatu

KARTA WYMAGANIA OGÓLNEGO DLA CZATU			
Identyfikator:	WOCZAT-04	Priorytet:	S
Nazwa:	Tworzenie czatu		
Opis:	System umożliwia użytkownikowi inicjowanie nowych rozmów poprzez tworzenie czatów prywatnych (1:1) lub grupowych z wybranymi uczestnikami.		
Udziałowiec:	U3		
Wymagania powiązane:	WFCZAT-05 , WPCZAT-01 , WPCZAT-02		

Tabela 4.48: Karta wymagania ogólnego dla czatu: Tworzenie czatu

4.2.1.2 Wymagania ogólne dla mapy

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-01	Priorytet:	S
Nazwa:	Wyświetlanie spotów na mapie		
Opis:	System wyświetla mapę, na której zaznaczone są spotty w formie wielokątów lub markerów, tak aby były widoczne dla użytkownika.		
Udziałowiec:	U3		
Wymagania powiązane:	WFMAPA-01 , WFMAPA-02 , WFMAPA-18 , WFMAPA-20 , WFMAPA-22 , WFMAPA-24 , WFMAPA-25 , WFMAPA-26 , WPMAPA-01 , WPMAPA-02 , WPMAPA-03		

Tabela 4.49: Karta wymagania ogólnego dla mapy: Wyświetlanie [spotów](#) na mapie

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-02	Priorytet:	S
Nazwa:	Wyświetlanie szczegółów spota		

KARTA WYMAGANIA OGÓLNEGO DLA MAPY (cd.)	
Opis:	Po kliknięciu na spota , system wyświetla użytkownikowi informacje o danym spocie , takich jak nazwa, lokalizacja, opis, zdjęcia czy komentarze.
Udziałowiec:	U3
Wymagania powiązane:	WFMAPA-02 , WFMAPA-03 , WFMAPA-04 , WFMAPA-06 , WFMAPA-07 , WFMAPA-08 , WFMAPA-09 , WFMAPA-10 , WFMAPA-11

Tabela 4.50: Karta wymagania ogólnego dla mapy: Wyświetlanie szczegółów spota

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-03	Priorytet:	S
Nazwa:	Wyświetlanie informacji pogodowych spota		
Opis:	Po kliknięciu na spota , system wyświetla użytkownikowi informacje o zarówno bieżącej, jak i prognozowanej pogodzie tak, aby użytkownik mógł dokładnie zapoznać się z warunkami panującymi w danym spocie .		
Udziałowiec:	U3		
Wymagania powiązane:	WFMAPA-18 , WFMAPA-19		

Tabela 4.51: Karta wymagania ogólnego dla mapy: Wyświetlanie informacji pogodowych [spota](#)

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-04	Priorytet:	S
Nazwa:	Wyszukiwanie spotów		
Opis:	System umożliwia użytkownikowi wyszukiwanie spotów na mapie oraz filtrowanie ich listy.		

KARTA WYMAGANIA OGÓLNEGO DLA MAPY (cd.)	
Udziałowiec:	U3
Wymagania powiązane:	WFMAPA-20, WFMAPA-21, WFMAPA-22, WFMAPA-23, WFMAPA-24

Tabela 4.52: Karta wymagania ogólnego dla mapy: Wyszukiwanie [spotów](#)

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-05	Priorytet:	S
Nazwa:	Komentowanie spotów		
Opis:	Użytkownik może dodać komentarz do wybranego spota , aby podzielić się swoją opinią i wrażeniami z innymi użytkownikami.		
Udziałowiec:	U3		
Wymagania powiązane:	WFMAPA-04 , WFMAPA-05 , WFMAPA-06		

Tabela 4.53: Karta wymagania ogólnego dla mapy: Komentowanie [spotów](#)

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-06	Priorytet:	S
Nazwa:	Dodawanie media do spotów		
Opis:	System umożliwia użytkownikowi dodanie do spota różne media, takie jak zdjęcia i filmy.		
Udziałowiec:	U3		
Wymagania powiązane:	WFMAPA-03 , WFMAPA-07 , WFMAPA-11 , WFMAPA-12 , WFMAPA-13 , WFMAPA-14 , WFMAPA-15 , WFMAPA-16 , WFMAPA-17		

Tabela 4.54: Karta wymagania ogólnego dla mapy: Dodawanie media do [spotów](#)

KARTA WYMAGANIA OGÓLNEGO DLA MAPY			
Identyfikator:	WOMAPA-07	Priorytet:	S
Nazwa:	Wyświetlenie lokalizacji użytkownika na mapie		
Opis:	System wyświetla na mapie obecną lokalizację użytkownika, aby mógł sprawdzić jakie spoty znajdują się w pobliżu.		
Udziałowiec:	U3		
Wymagania powiązane:	WFMAPA-10 , WFMAPA-25		

Tabela 4.55: Karta wymagania ogólnego dla mapy: Wyświetlenie lokalizacji użytkownika na mapie

4.2.1.3 Wymagania ogólne dla wyszukiwarki spotów

KARTA WYMAGANIA OGÓLNEGO DLA WYSZUKIWARKI SPOTÓW			
Identyfikator:	WOWYSZ-01	Priorytet:	M
Nazwa:	Wyszukiwanie na prostej wyszukiwarce		
Opis:	System udostępnia użytkownikowi prostą wyszukiwarkę spotów na stronie głównej, umożliwiającą przeglądanie listy spotów oraz wyszukiwanie wyników według lokalizacji (kraj, region, miasto). Dodatkowo system prezentuje karuzelę najpopularniejszych spotów dla wybranej lokalizacji wraz z podstawowymi informacjami o miejscu.		
Udziałowiec:	U3		
Wymagania powiązane:	WFWYSZ-01 , WFWYSZ-02 , WFWYSZ-03 , WFWYSZ-04 , WFWYSZ-05 , WPWYSZ-01 , WPWYSZ-02 .		

Tabela 4.56: Karta wymagania ogólnego dla wyszukiwarki spotów: Wyszukiwanie na prostej wyszukiwarce

KARTA WYMAGANIA OGÓLNEGO DLA WYSZUKIWARKI SPOTÓW			
Identyfikator:	WOWYSZ-02	Priorytet:	M
Nazwa:	Wyszukiwanie na zaawansowanej wyszukiwarce		
Opis:	System udostępnia użytkownikowi zaawansowaną wyszukiwarkę spotów na stronie głównej, umożliwiającą filtrowanie listy spotów według miasta, tagów oraz minimalnej oceny, a także sortowanie wyników według oceny oraz popularności. Wyniki wyszukiwania są aktualizowane zgodnie z ustalonymi filtrami i sortowaniem, a użytkownik może przejść do szczegółów wybranego spota.		
Udziałowiec:	U3		
Wymagania powiązane:	WFWYSZ-03 , WFWYSZ-04 , WFWYSZ-05 , WFWYSZ-06 , WFWYSZ-07 , WFWYSZ-08 , WPWYSZ-01 , WPWYSZ-02 .		

Tabela 4.57: Karta wymagania ogólnego dla wyszukiwarki spotów: Wyszukiwanie na zaawansowanej wyszukiwarce

4.2.2 Wymagania funkcjonalne

Niniejszy rozdział zawiera wymagania funkcjonalne postawione systemowi. Został on podzielony tematycznie.

4.2.2.1 Wymagania funkcjonalne dla mapy

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-01	Priorytet:	S
Nazwa:	Wyświetlanie spotów na mapie		
Opis:	System wyświetla na mapie spoty w formie wielokątów, a gdy widok mapy zostanie oddalony zamienia je na pinezki.		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik widzi na mapie zaznaczone spoty jako wielokąty. • Po oddaleniu widoku mapy, wielokąty są zamieniane na pinezki znajdujące się w tych samych miejscach.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01

Tabela 4.58: Wymaganie funkcjonalne dla mapy: Wyświetlanie **spotów** na mapie

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-02	Priorytet:	S
Nazwa:	Wyświetlanie szczegółów spota		
Opis:	Po kliknięciu na wybranego spota , system wyświetla o nim informacje.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może kliknąć na mapie dowolny spot. • O wybranym spocie wyświetlane są następujące informacje: nazwa, lokalizacja, ocena w gwiazdkach, liczba wyświetleń, opis, galeria z mediami (zdjęcia i filmy), lista komentarzy. 		
Udziałowiec:	U3		
Realizator:	Stanisław Oziemczuk		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
Wymagania powiązane:	WOMAPA-01 , WOMAPA-02

Tabela 4.59: Wymaganie funkcjonalne dla mapy: Wyświetlanie szczegółów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-03	Priorytet: S
Nazwa:	Wyświetlanie interaktywnej galerii mediów spota	
Opis:	W panelu ze szczegółami spota wyświetlana jest galeria mediów (zdjęć i filmów), w której użytkownik może przeglądać wszystkie media dodane do wybranego spota .	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlana jest galeria zawierająca zdjęcia oraz filmy dodane do wybranego spota. • Galeria umożliwia przechodzenie w sposób zapętlony między mediami za pomocą strzałek. • Pod mediami wyświetlane są znaczniki informujące o ilości elementów w galerii. 	
Udziałowiec:	U3	
Realizator:	Stanisław Oziemczuk	
Wymagania powiązane:	WOMAPA-02 , WOMAPA-06	

Tabela 4.60: Wymaganie funkcjonalne dla mapy: Wyświetlanie interaktywnej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-04	Priorytet: S
Nazwa:	Wyświetlanie przewijalnej listy komentarzy spota	
Opis:	W panelu ze szczegółami spota wyświetlana jest lista wszystkich komentarzy. Komentarz zawiera następujące elementy: treść, autor, data dodania, ocena w gwiazdkach, media (opcjonalnie), liczbę polubień, liczbę niepolubień.	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlana jest lista wszystkich komentarzy dodanych do wybranego spota. • Lista komentarzy jest przewijana za pomocą nieskończonego przewijania. • Każdy komentarz zawiera treść, autora, ocenę w gwiazdkach, datę dodania, liczbę polubień, liczbę niepolubień. • Jeśli do komentarze są dodane media, są one wyświetlane. 	
Udziałowiec:	U3	
Realizator:	Stanisław Oziemczuk	
Wymagania powiązane:	WOMAPA-02 , WOMAPA-05	

Tabela 4.61: Wymaganie funkcjonalne dla mapy: Wyświetlanie przewijalnej listy komentarzy [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-05	Priorytet: S
Nazwa:	Ocena komentarza spota	
Opis:	System umożliwia użytkownikowi ocenienie komentarza spota .	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest komentarz dodany do wybranego spota. • Komentarz zawiera przyciski do wystawienia zarówno pozytywnej, jak i negatywnej oceny. • Po kliknięciu przycisku użytkownik natychmiast widzi zmianę w liczbie wystawionych opinii wybranego rodzaju oraz oznaczenie, którą opcję wybrał. • Ponownie kliknięcie tego samego przycisku powoduje cofnięcie wystawienia oceny. • Kliknięcie przycisku przeciwnej operacji powoduje wykonanie jej i cofnięcie poprzedniej. • Jeżeli użytkownik nie jest zalogowany, wyświetlany jest odpowiedni komunikat. 	
Udziałowiec:	U3	
Realizator:	Stanisław Oziemczuk	
Wymagania powiązane:	WOMAPA-05	

Tabela 4.62: Wymaganie funkcjonalne dla mapy: Ocena komentarza [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-06	Priorytet: S
Nazwa:	Dodanie komentarza do spota	
Opis:	System umożliwia użytkownikowi dodanie komentarza do spota .	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk otwierający formularz umożliwiający dodanie nowego komentarza do wybranego spota. • Formularz zawiera następujące pola: ocena spota w gwiazdkach od 0 do 5, treść komentarza, dodanie filmów i/lub zdjęć (od 0 do 20). • Po kliknięciu przycisku do publikacji komentarza, użytkownik widzi go na liście wszystkich komentarzy. • Zamknięcie formularza nie powoduje zapisu danych. • Brak zalogowania uniemożliwia otwarcie formularza, użytkownik jest o tym odpowiednio informowany. • Wpisanie niepoprawnych danych do formularza blokuje możliwość publikacji komentarza, a użytkownik jest informowany o błędach. 	
Udziałowiec:	U3	
Realizator:	Stanisław Oziemczuk	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Wymagania powiązane:	WOMAPA-02 , WOMAPA-05
-----------------------------	---

Tabela 4.63: Wymaganie funkcjonalne dla mapy: Dodanie komentarza do [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-07	Priorytet: S
Nazwa:	Dodanie media do spota	
Opis:	System umożliwia użytkownikowi dodanie media (zdjęć lub filmów) do spota .	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Użytkownikowi wyświetlany jest przycisk otwierający formularz umożliwiający dodanie nowego media do wybranego spota. ● Formularz zawiera przycisk do wyboru zdjęć i filmów z urządzenia. ● Wybrane pliki są wyświetlane w podglądzie wraz z możliwością ich usunięcia. ● Dodanie medii powoduje wyświetlenie ich w galerii mediów spota oraz w odpowiedniej zakładce profilu użytkownika. ● Zamknięcie formularza nie powoduje zapisu danych. ● Brak zalogowania uniemożliwia otwarcie formularza, użytkownik jest o tym odpowiednio informowany. ● Wybranie niepoprawnych formatów plików blokuje możliwość dodania medii, użytkownik jest informowany o błędach.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-02 , WOMAPA-06

Tabela 4.64: Wymaganie funkcjonalne dla mapy: Dodanie media do [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-08	Priorytet: S
Nazwa:	Dodanie spota do listy ulubionych	
Opis:	System umożliwia użytkownikowi dodanie spota do listy ulubionych.	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający dodanie wybranego spota do listy ulubionych. • Po kliknięciu przycisku użytkownik widzi spota w swojej liście ulubionych, przycisk zmienia wygląd oznaczający, że spot znajduje się na liście. • Gdy spot znajduje się na liście ulubionych, kliknięcie przycisku powoduje usunięcie go z tej listy. • Brak zalogowania uniemożliwia dodanie spota do listy ulubionych, użytkownik jest o tym odpowiednio informowany. 	
Udziałowiec:	U3	
Realizator:	Stanisław Oziemczuk	
Wymagania powiązane:	WOMAPA-02	

Tabela 4.65: Wymaganie funkcjonalne dla mapy: Dodanie **spota** do listy ulubionych

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-09	Priorytet: S

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
Nazwa:	Udostępnienie spota
Opis:	System umożliwia użytkownikowi udostępnienie spota .
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający udostępnienie wybranego spota. • Po kliknięciu przycisku do schowka kopowany jest link do danego spota. • Wklejenie linku do przeglądarki powoduje otworzenie panelu ze szczegółami spota oraz przybliżenie jego lokalizacji na mapie. • Użytkownik jest informowany o błędach, które wystąpiły w trakcie udostępniania spota.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-02

Tabela 4.66: Wymaganie funkcjonalne dla mapy: Udostępnienie [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-10	Priorytet:	S
Nazwa:	Nawigowanie do spota		
Opis:	System pokazuje użytkownikowi trasę od jego lokalizacji do wybranego spota .		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Użytkownikowi wyświetlany jest przycisk umożliwiający pokazanie trasy do wybranego spota. ● Po kliknięciu przycisku pobierana jest aktualna lokalizacja użytkownika, a w przeglądarce otwierana jest nowa karta z Google Maps z trasą od pozycji użytkownika do danego spota. ● Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-02 , WOMAPA-07

Tabela 4.67: Wymaganie funkcjonalne dla mapy: Nawigowanie do [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-11	Priorytet: S
Nazwa:	Duża galeria mediów spota	
Opis:	System umożliwia użytkownikowi włączenie dużej galerii mediów wybranego spota .	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Po kliknięciu na zdjęcie w galerii mediów spota, użytkownikowi wyświetlana jest duża galeria mediów. Taka sama akcja jest wykonywana po kliknięciu zdjęcia lub film znajdujący się w komentarzu spota. ● W dużej galerii mediów wyświetlana jest lista wszystkich plików wybranego typu, przewijana przy użyciu nieskończonego przewijania. ● Wyświetlalne jest powiększone obecnie wybrane media. ● Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-02 , WOMAPA-06

Tabela 4.68: Wymaganie funkcjonalne dla mapy: Duża galeria mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-12	Priorytet:	S
Nazwa:	Zarządzanie listą medii w dużej galerii mediów spota		
Opis:	System umożliwia użytkownikowi sortowanie oraz filtrowanie medii w dużej galerii wybranego spota .		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Użytkownikowi wyświetlane są przyciski do filtrowania listy po typie medii (zdjęcia lub filmy) oraz umożliwiające sortowanie po liczbie polubień czy dacie dodania. ● Po kliknięciu przycisku lista jest aktualizowana, a obecne wybrane media jest ustawiane na pierwszy element listy. ● Zaznaczone opcje filtrowania oraz sortowania są oznaczone jako wybrane. ● Domyślną opcją filtrowania po typie media jest ta, którą użytkownik kliknął włączając dużą galerię mediów. ● Domyślnie sortowanie jest po dacie dodania media, od najnowszego. ● Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-06

Tabela 4.69: Wymaganie funkcjonalne dla mapy: Zarządzanie listą medii w dużej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-13	Priorytet: S

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
Nazwa:	Wyświetlanie powiększonego obecnie wybranego media w dużej galerii mediów spota
Opis:	System wyświetla użytkownikowi powiększone obecnie wybrane media w dużej galerii wybranego spota .
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlane jest obecnie wybrane media, które jest powiększone. • Powiększone media zawiera: informacje o autorze, przyciski do udostępnienia, pobrania, polubienia, powiększenia media na cały ekran oraz liczbę polubień. • Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-06

Tabela 4.70: Wymaganie funkcjonalne dla mapy: Wyświetlanie powiększonego obecnie wybranego media w dużej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-14	Priorytet: S
Nazwa:	Udostępnienie obecnie wybranego media w dużej galerii mediów spota	
Opis:	System umożliwia użytkownikowi udostępnienie obecnie wybranego media w dużej galerii wybranego spota .	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający udostępnienie obecnie wybranego media. • Po kliknięciu przycisku do schowka kopowany jest link do wybranego media. • Wklejenie linku do przeglądarki powoduje pobranie media. • Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-06

Tabela 4.71: Wymaganie funkcjonalne dla mapy: Udostępnienie obecnie wybranego media w dużej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-15	Priorytet: S
Nazwa:	Polubienie obecnie wybranego media w dużej galerii mediów spota	
Opis:	System umożliwia użytkownikowi polubienie obecnie wybranego media w dużej galerii wybranego spota .	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający polubienie obecnie wybranego media. • Po kliknięciu przycisku użytkownik natychmiast widzi aktualizację liczby polubień, a przycisk zmienia wygląd oznaczający wykonanie operacji. • Gdy media jest polubione, kliknięcie przycisku powoduje odwrotną operację. • Brak zalogowania uniemożliwia wykonanie operacji. • Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-06

Tabela 4.72: Wymaganie funkcjonalne dla mapy: Polubienie obecnie wybranego media w dużej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-16	Priorytet: S
Nazwa:	Powiększenie na cały ekran obecnie wybranego media w dużej galerii mediów spota	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Opis:	System umożliwia użytkownikowi powiększenie na cały ekran obecnie wybranego media w dużej galerii wybranego spota .
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający powiększenie na cały ekran obecnie wybranego media. • Po kliknięciu przycisku wybrane media jest powiększone na całą szerokość i wysokość ekranu. • Powiększenie elementu nie powoduje zaburzenia jego proporcji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-06

Tabela 4.73: Wymaganie funkcjonalne dla mapy: Powiększenie na cały ekran obecnie wybranego media w dużej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-17	Priorytet:	S
Nazwa:	Pobranie obecnie wybranego media w dużej galerii mediów spota		
Opis:	System umożliwia użytkownikowi pobranie obecnie wybranego media w dużej galerii wybranego spota .		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający pobranie obecnie wybranego media. • Po kliknięciu przycisku plik zostaje pobrany na urządzenie użytkownika. • Użytkownik jest informowany o błędach, które wystąpiły w trakcie operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-06

Tabela 4.74: Wymaganie funkcjonalne dla mapy: Pobranie obecnie wybranego media w dużej galerii mediów [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-18	Priorytet:	S
Nazwa:	Wyświetlanie informacji pogodowych spota		
Opis:	System wyświetla użytkownikowi obecne dane pogodowe wybranego spota .		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Po kliknięciu spota użytkownikowi wyświetlane są obecne dane pogodowe zawierające: temperaturę, prędkość wiatru, ikonę symbolizującą stan pogody (np. zachmurzenie) oraz przycisk do otwarcia panelu ze szczegółowymi danymi. ● Użytkownik jest informowany o błędach, które wystąpiły w podczas pobierania danych.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01 , WOMAPA-03

Tabela 4.75: Wymaganie funkcjonalne dla mapy: Wyświetlanie informacji pogodowych [spota](#)

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-19	Priorytet:	S
Nazwa:	Wyświetlanie szczegółowych informacji pogodowych spota		
Opis:	System wyświetla użytkownikowi szczegółowe dane pogodowe wybranego spota .		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Użytkownikowi wyświetlane są szczegółowe informacje pogodowe o wybranym spocie. ● Dane zawierają: <ul style="list-style-type: none"> – obecną godzinę i temperaturę – ikonę symbolizującą stan pogody wraz z opisem – wskaźnik UV – punkt rosy – wilgotność – prawdopodobieństwo opadów – prędkości wiatrów na różnych wysokościach do wyboru z możliwością określenia jednostki km/h lub m/s – wykres zawierający prognozę pogody na 3 kolejne dni ● Użytkownik jest informowany o błędach, które wystąpiły podczas pobierania danych.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-03

Tabela 4.76: Wymaganie funkcjonalne dla mapy: Wyświetlanie szczegółowych informacji pogodowych **spota**

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-20	Priorytet: S
Nazwa:	Wyszukiwanie spotów po nazwie	
Opis:	System umożliwia użytkownikowi wyszukiwanie spotów po nazwie.	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Wyświetlane jest pole, w które użytkownik może wpisać frazę. ● Gdy dane są wpisywane, wyświetlana jest lista nazw spotów, które zawierają w sobie podany fragment. ● Użytkownik może wybrać pozycję z listy, powodującą to wstawienie jej w pole wyszukiwania i pokazanie wyników. ● Użytkownik może nie wybrać pozycji z listy i kliknąć przycisk do pokazania wyników, wyświetlana jest lista wszystkich spotów, których nazwy zawierają w sobie podaną frazę. ● Lista z wynikami wyszukiwania jest przewijana za pomocą nieskończonego przewijania. ● W przypadku braku pasujących nazw spotów wyświetlany jest odpowiedni komunikat. ● Użytkownik jest informowany o błędach, które wystąpiły w czasie pobierania danych. ● Użytkownik ma możliwość wyczyszczenia pola do wyszukiwania i zamknięcia listy z wynikami.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
---	--

Wymagania powiązane:	WOMAPA-01 , WOMAPA-04
-----------------------------	---

Tabela 4.77: Wymaganie funkcjonalne dla mapy: Wyszukiwanie [spotów](#) po nazwie

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY	
Identyfikator:	WFMAPA-21
Priorytet:	S
Nazwa:	Sortowanie wyników wyszukiwania spotów po nazwie
Opis:	System umożliwia użytkownikowi sortowanie wyników zawierającą wyszukiwania spotów po nazwie.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlane są przyciski umożliwiające ustawienie sortowania wyników po: <ul style="list-style-type: none"> – ocenach rosnąco – ocenach malejąco – liczbie ocen rosnąco – liczbie ocen malejąco . • Po wybraniu opcji sortowania, lista jest natychmiast aktualizowana. • Użytkownik jest informowany o błędach, które wystąpiły w podczas pobierania danych.
Udziałowiec:	U3

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-04

Tabela 4.78: Wymaganie funkcjonalne dla mapy: Sortowanie wyników wyszukiwania [spotów](#) po nazwie

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-22	Priorytet: S
Nazwa:	Sortowanie wyników wyszukiwania spotów po nazwie	
Opis:	System umożliwia użytkownikowi wyświetlenie listy spotów znajdujących się w widocznym obszarze mapy.	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Użytkownikowi wyświetlany jest przycisk powodujący wyświetlenie listy wszystkich spotów znajdujących się w widocznym obszarze mapy. ● Po kliknięciu przycisku wyświetlana lista jest przewijalna przy użyciu nieskończonego przewijania. ● Zmienienie obszaru widocznego bez ponownego kliknięcia przycisku nie powoduje pobrania nowych danych. ● Ponowne kliknięcie przycisku powoduje pobranie nowych danych i zastąpienie nimi poprzednich wyników. ● Gdy w widocznym obszarze mapy nie ma spotów, wyświetlany jest odpowiedni komunikat. ● Użytkownik jest informowany o błędach, które wystąpiły w podczas pobierania danych.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01 , WOMAPA-04

Tabela 4.79: Wymaganie funkcjonalne dla mapy: Sortowanie wyników wyszukiwania **spotów** po nazwie

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-23	Priorytet: S
Nazwa:	Zarządzanie listą wyników wyszukiwania spotów w widocznym obszarze mapy	
Opis:	System umożliwia użytkownikowi sortowanie oraz filtrowanie wyników zawierających spotty znajdujące się w widocznym obszarze mapy.	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none">● Użytkownikowi wyświetlane są przyciski umożliwiające ustawienie sortowania wyników po:<ul style="list-style-type: none">– ocenach rosnąco– ocenach malejąco– liczbie ocen rosnąco– liczbie ocen malejąco ● Po wybraniu opcji sortowania, lista jest natychmiast aktualizowana.● Użytkownik może filtrować spoty ustawiając minimalną ocenę.● Użytkownik może filtrować wyniki poprzez wpisanie frazy. Podczas wpisywania wyświetlane są podpowiedzi w postaci listy nazw spotów zawierających wpisywany tekst. Wynikiem filtrowania są spoty, których nazwy zawierają w sobie wpisaną frazę.● Po ustaleniu filtrowania lista jest aktualizowana, brane są pod uwagę wszystkie filtry i sortowanie.● W przypadku braku pasujących wyników wyświetlany jest odpowiedni komunikat.● Użytkownik jest informowany o błędach, które wystąpiły w podczas pobierania danych.
-----------------------------	---

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-04

Tabela 4.80: Wymaganie funkcjonalne dla mapy: Zarządzanie listą wyników wyszukiwania [spotów](#) w widocznym obszarze mapy

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-24	Priorytet: S
Nazwa:	Przybliżanie mapy do lokalizacji spotów z listy z wynikami	
Opis:	System umożliwia użytkownikowi przybliżenie widoku mapy do lokalizacji spota , będące na liście wyników wyszukiwania.	
Kryteria akceptacji:	<ul style="list-style-type: none"> Po kliknięciu spota na liście wyników wyszukiwania, widok mapy przenoszony jest do jego lokalizacji. 	
Udziałowiec:	U3	
Realizator:	Stanisław Oziemczuk	
Wymagania powiązane:	WOMAPA-01 , WOMAPA-04	

Tabela 4.81: Wymaganie funkcjonalne dla mapy: Przybliżanie mapy do lokalizacji [spotów](#) z listy z wynikami

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY		
Identyfikator:	WFMAPA-25	Priorytet: S
Nazwa:	Wyświetlanie na mapie lokalizacji użytkownika	

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)	
Opis:	System umożliwia użytkownikowi wyświetlenie na mapie jego obecnej lokalizacji.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlany jest przycisk umożliwiający zaznaczenie na mapie jego obecnej lokalizacji. • Po kliknięciu przycisku pobierane są dane o lokalizacji użytkownika i jego pozycja jest zaznaczona na mapie. • Widok mapy jest przybliżany na wyznaczoną lokalizację. • Użytkownik jest informowany o błędach, które wystąpiły w podczas operacji.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01 , WOMAPA-07

Tabela 4.82: Wymaganie funkcjonalne dla mapy: Wyświetlanie na mapie lokalizacji użytkownika

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY			
Identyfikator:	WFMAPA-26	Priorytet:	S
Nazwa:	Ustawianie przybliżenia mapy		
Opis:	System umożliwia użytkownikowi zmianę przybliżenia widoku mapy.		

KARTA WYMAGANIA FUNKCJONALNEGO DLA MAPY (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownikowi wyświetlane są przyciski do przybliżania i oddalania widoku mapy. • Po kliknięciu jednego z przycisków wykonywana jest odpowiednia akcja. • Przejście między przybliżeniami jest płynne i w razie potrzeby ładowane są kolejne kafelki mapy.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01

Tabela 4.83: Wymaganie funkcjonalne dla mapy: Ustawianie przybliżenia mapy

4.2.2.2 Wymagania funkcjonalne dla czatu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
Identyfikator:	WFCZAT-01	Priorytet:	S
Nazwa:	Wysyłanie GIF'ów		
Opis:	System umożliwia wysyłanie w wiadomościach animowanych obrazów GIF w ramach wybranego czatu (1:1 lub grupowego).		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może wybrać animowany obraz GIF z wbudowanego okna wyboru i dołączyć go do wiadomości. • Po wysłaniu GIF'a wyświetla się poprawnie w treści czatu u wszystkich uczestników.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01.

Tabela 4.84: Wymaganie funkcjonalne dla czatu: Wysyłanie GIF'ów

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU

Identyfikator:	WFCZAT-02	Priorytet:	S
Nazwa:	Wysyłanie plików		
Opis:	System umożliwia dołączanie i wysyłanie plików (dokumentów, zdjęć, archiwów) jako załączników do wiadomości na czacie.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może wybrać jeden lub wiele plików z dysku i dołączyć je do wiadomości. • Odbiorca widzi w czacie element reprezentujący plik. • Kliknięcie w załącznik umożliwia pobranie lub otwarcie pliku. 		
Udziałowiec:	U3		
Realizator:	Adam Langmesser		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)	
Wymagania powiązane:	WOCZAT-01.

Tabela 4.85: Wymaganie funkcjonalne dla czatu: Wysyłanie plików

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU	
Identyfikator:	WFCZAT-03
Nazwa:	Wysyłanie wiadomości prywatnych
Opis:	System umożliwia prowadzenie prywatnych rozmów 1:1 pomiędzy dwoma użytkownikami.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może rozpoczęć nowy czat 1:1 z innym użytkownikiem lub kontynuować istniejący. • Wiadomości z prywatnego czatu są widoczne wyłącznie dla tych dwóch użytkowników. • Nowe wiadomości pojawiają się w czasie zbliżonym do rzeczywistego bez konieczności przeładowania strony.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01.

Tabela 4.86: Wymaganie funkcjonalne dla czatu: Wysyłanie wiadomości prywatnych

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU	
Identyfikator:	WFCZAT-04
Priorytet:	S

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)

Nazwa:	Wysyłanie wiadomości do wielu osób jednocześnie
Opis:	System umożliwia wysyłanie jednej wiadomości do wielu użytkowników w ramach czatu grupowego.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Wiadomość wysłana w czacie grupowym jest dostarczana wszystkim jego członkom. • UI wyraźnie wskazuje, że rozmowa to czat grupowy (nazwa, avatar, liczba uczestników). • Nowi uczestnicy dołączeni do czatu widzą historię rozmowy.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01 .

Tabela 4.87: Wymaganie funkcjonalne dla czatu: Wysyłanie wiadomości do wielu osób jednocześnie

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU

Identyfikator:	WFCZAT-05	Priorytet:	S
Nazwa:	Rozpoczynanie nowego czatu		
Opis:	System umożliwia użytkownikowi utworzenie nowego czatu prywatnego lub grupowego oraz dodanie do niego wskazanych uczestników.		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może zainicjować nowy czat z widoku listy czatów lub profilu innego użytkownika. • Po wysłaniu pierwszej wiadomości przez twórcę, czat pojawia się na liście czatów wszystkich jego członków.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-04.

Tabela 4.88: Wymaganie funkcjonalne dla czatu: Rozpoczynanie nowego czatu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
Identyfikator:	WFCZAT-06	Priorytet:	S
Nazwa:	Wysyłanie emotikonów		
Opis:	System umożliwia wysyłanie emoji w treści wiadomości na czacie.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może wstawiać emotikony z wbudowanego panelu wyboru emoji. 		
Udziałowiec:	U3		
Realizator:	Adam Langmesser		
Wymagania powiązane:	WOCZAT-01.		

Tabela 4.89: Wymaganie funkcjonalne dla czatu: Wysyłanie emotikonów

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
Identyfikator:	WFCZAT-07	Priorytet:	S
Nazwa:	Dostępność czatu po utworzeniu		
Opis:	Czat po utworzeniu pozostaje dostępny dla jego członków; użytkownik może później odnaleźć czat i wrócić do wcześniejszej konwersacji.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • Utworzone czaty pojawiają się na liście czatów użytkownika. • Po ponownym zalogowaniu użytkownik może otworzyć istniejący czat i zobaczyć jego historię. 		
Udziałowiec:	U3		
Realizator:	Adam Langmesser		
Wymagania powiązane:	WOCZAT-03.		

Tabela 4.90: Wymaganie funkcjonalne dla czatu: Dostępność czatu po utworzeniu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
Identyfikator:	WFCZAT-08	Priorytet:	S
Nazwa:	Edytowanie nazwy czatu grupowego		
Opis:	System umożliwia zmianę nazwy istniejącego czatu grupowego przez członka czatu.		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może edytować nazwę z poziomu ustawień czatu. • Nowa nazwa jest natychmiast widoczna na liście czatów u wszystkich uczestników. • Historia wiadomości pozostaje niezmieniona po zmianie nazwy czatu.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-02.

Tabela 4.91: Wymaganie funkcjonalne dla czatu: Edytowanie nazwy czatu grupowego

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU			
Identyfikator:	WFCZAT-09	Priorytet:	S
Nazwa:	Edycja zdjęcia czatu grupowego		
Opis:	System umożliwia zmianę obrazu/avatara reprezentującego czat grupowy.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • Członek czatu może wgrać nowe zdjęcie czatu grupowego. • Zmienione zdjęcie jest widoczne na liście czatów i w nagłówku rozmowy. • Niepoprawne formaty lub zbyt duże pliki są odrzucane z informacją o błędzie. 		
Udziałowiec:	U3		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)	
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-02.

Tabela 4.92: Wymaganie funkcjonalne dla czatu: Edycja zdjęcia czatu grupowego

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU	
Identyfikator:	WFCZAT-10
Priorytet:	W
Nazwa:	Edycja wysłanej wiadomości
Opis:	System umożliwia użytkownikowi edycję treści wcześniej wysłanej wiadomości na czacie.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może edytować swoje wiadomości przez ograniczony czas od momentu wysłania (konfigurowalny limit). • Zmieniona wiadomość jest oznaczona etykietą „(edytowano)”. • Odbiorcy widzą zaktualizowaną treść bez duplikowania wiadomości.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01.

Tabela 4.93: Wymaganie funkcjonalne dla czatu: Edycja wysłanej wiadomości

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU	
Identyfikator:	WFCZAT-11
Priorytet:	S

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)

Nazwa:	Dodawanie użytkowników do istniejącego czatu
Opis:	System umożliwia dodawanie nowych użytkowników do już istniejącego czatu grupowego.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Właściciel lub uprawniony użytkownik może wyszukać i dodać nowe osoby do czatu. • Nowo dodani użytkownicy pojawiają się na liście uczestników i mogą od razu brać udział w rozmowie.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-02.

Tabela 4.94: Wymaganie funkcjonalne dla czatu: Dodawanie użytkowników do istniejącego czatu

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU

Identyfikator:	WFCZAT-12	Priorytet:	S
Nazwa:	Wyświetlanie starszych wiadomości		
Opis:	System powinien domyślnie wyświetlać co najmniej ostatnie 20 wiadomości w czacie, a starsze wiadomości pobierać na bieżąco podczas przewijania historii przez użytkownika.		

KARTA WYMAGANIA FUNKCJONALNEGO DLA CZATU (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • Po wejściu na czat użytkownik widzi minimum 20 ostatnich wiadomości. • Przewijanie historii w góre automatycznie pobiera starsze wiadomości (mechanizm Infinite scroll). • Dociąganie wiadomości nie powoduje zauważalnych opóźnień interfejsu.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-03.

Tabela 4.95: Wymaganie funkcjonalne dla czatu: Wyświetlanie starszych wiadomości

4.2.2.3 Funkcjonalności dla forum

4.2.2.4 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 ; drona- rze 2.3 .	
Wymagania powiązane:		

Tabela 4.96: Profil użytkownika

KARTA WYMAGANIA		
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet: M
Nazwa:	Lista dodanych spotów	
Opis:	Jako użytkownik chcę sprawdzić listę spotów, które <u>dodałem</u> .	
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 + axios</code> ; prezentacja listy z podstawowymi danymi.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.97: 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 axios (POST) z withCredentials.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; droniarze 2.3 .	
Wymagania powiązane:		

Tabela 4.98: 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 spodem 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 + axios</code> ; prezentacja z miniaturami.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.99: 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 spodem 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 + axios</code> ; prezentacja z miniaturami.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.100: 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 + axios</code> ; standardowa prezentacja listy.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; drona- rze 2.3 .	
Wymagania powiązane:		

Tabela 4.101: 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 + axios</code> ; standardowa prezentacja listy.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; drona- rze 2.3 .	
Wymagania powiązane:		

Tabela 4.102: 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 + axios</code> ; standardowa prezentacja listy.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.103: 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 + axios</code> ; prezentacja w zakładkach/kategoriach.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.104: 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 + axios</code> ; standardowa prezentacja listy.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.105: 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 — aktualizowane dane.	
Sytuacje wyjątkowe:	Nieprawidłowe dane formularza; błąd połączenia z API.	
Szczegóły implementacji:	Formularz w React; validacja 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 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.106: 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 <code>axios</code> ; obsługa komunikatów o powodzeniu/błędach.	
Udziałowiec:	Zespół projektowy 2.1 ; promotor 2.2 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.107: 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 ; dronarze 2.3 .	
Wymagania powiązane:		

Tabela 4.108: Dodawanie do znajomych

4.2.2.5 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 axios z withCredentials. SSO: integracja z Google i GitHub (OAuth 2.0) z przekierowaniem i ustawniem sesji po stronie backendu (httpOnly cookie). Obsługa statusu 401 zgodnie z mechanizmem wylogowania.		
Udziałowiec:	Zespół projektowy 2.1 , promotor 2.2 , dronaře 2.3 .		
Wymagania powiązane:			

Tabela 4.109: Logowanie i rejestracja

4.2.2.6 Wymagania funkcjonalne dla wyszukiwarki spotów

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW			
Identyfikator:	WFWYSZ-01	Priorytet:	M
Nazwa:	Wyświetlenie najpopularniejszych spotów w karuzeli		

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Opis:	System prezentuje na stronie głównej karuzelę z najpopularniejszymi spotami, aby użytkownik mógł szybko przeglądać rekomendowane miejsca dla wybranej lokalizacji.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Karuzela wyświetla nazwę spota oraz miasto. • Każda pozycja w karuzeli zawiera zdjęcie (miniaturę) spota. • Karuzela jest widoczna po wejściu na stronę główną bez konieczności wykonywania dodatkowych akcji.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WPWYSZ-01 .

Tabela 4.96: Wymaganie funkcjonalne dla wyszukiarki spotów: Wyświetlenie najpopularniejszych spotów w karuzeli

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW		
Identyfikator:	WFWYSZ-02	Priorytet: M
Nazwa:	Wyszukiwanie za pomocą państwa, regionu oraz miasta	

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Opis:	System umożliwia użytkownikowi filtrowanie listy spotów według lokalizacji, poprzez wybór państwa, regionu oraz miasta. Zastosowanie filtrów następuje po wybraniu akcji wyszukiwania.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może wskazać państwo, region oraz miasto jako filtry wyszukiwania. • Zmiana wartości filtrów nie powoduje automatycznego odświeżenia wyników. • Po kliknięciu przycisku wyszukiwania lista spotów jest aktualizowana zgodnie z wybraną lokalizacją.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WFWYSZ-04 , WFWYSZ-03 , WPWYSZ-01 .

Tabela 4.97: Wymaganie funkcjonalne dla wyszukiwarki spotów: Wyszukiwanie za pomocą państwa, regionu oraz miasta

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW		
Identyfikator:	WFWYSZ-03	Priorytet:
Nazwa:	Wyświetlenie wyszukanych spotów	
Opis:	System wyświetla listę spotów spełniających kryteria wyszukiwania, aby użytkownik mógł przeglądać dostępne wyniki.	

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Wyniki wyszukiwania są prezentowane w formie listy (kart spotów). • Każda karta spota zawiera: <ul style="list-style-type: none"> – nazwę spota, – zdjęcie, – tagi, – średnią ocenę oraz liczbę ocen, – informacje pogodowe dla lokalizacji spota, – lokalizację (miasto).
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WOWYSZ-02 , WFWYSZ-02 , WFWYSZ-06 , WFWYSZ-05 , WPWYSZ-01 .

Tabela 4.98: Wymaganie funkcjonalne dla wyszukiwarki spotów: Wyświetlenie wyszukanych spotów

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW		
Identyfikator:	WFWYSZ-04	Priorytet: M
Nazwa:	Podpowiedzi wartości w polach lokalizacji (autocomplete)	

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Opis:	System wspiera użytkownika podczas uzupełniania filtrów lokalizacji, wyświetlając listę podpowiedzi dla państwa, regionu oraz miasta na podstawie wpisywanej frazy.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Podczas wpisywania w pole państwa, regionu lub miasta system wyświetla listę podpowiedzi pasujących do wpisanej frazy. • Lista podpowiedzi zawęża się wraz z dopisywaniem kolejnych znaków. • Użytkownik może wybrać wartość z listy podpowiedzi (kliknięciem), co uzupełnia pole wyszukiwania.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WOWYSZ-02 , WFWYSZ-02 , WPWYSZ-02 .

Tabela 4.99: Wymaganie funkcjonalne dla wyszukiwarki spotów: Podpowiedź wartości w polach lokalizacji (autocomplete)

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW	
Identyfikator:	WFWYSZ-05
Priorytet:	M
Nazwa:	Przycisk do pokazania spota na mapie oraz zobaczenia jego szczegółów

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Opis:	System udostępnia w wynikach wyszukiwania akcje umożliwiające użytkownikowi przejście do szczegółów spota oraz wyświetlenie go na mapie.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Dla każdego spota na liście wyników dostępna jest akcja „Details”. • Dla każdego spota na liście wyników dostępna jest akcja „See on map”. • Kliknięcie akcji „See on map” przenosi użytkownika do widoku mapy z zaznaczonym spotem. • Kliknięcie akcji „Details” przenosi użytkownika do widoku mapy i otwiera szczegóły spota.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WOWYSZ-02 , WFWYSZ-03 .

Tabela 4.100: Wymaganie funkcjonalne dla wyszukiwarki spotów: Przycisk do pokazania spota na mapie oraz zobaczenia jego szczegółów

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW	
Identyfikator:	WFWYSZ-06
Nazwa:	Wyszukiwanie za pomocą miasta oraz tagów

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Opis:	System umożliwia użytkownikowi zawężenie wyników wyszukiwania poprzez wybór miasta oraz tagów opisujących spot. Zastosowanie filtrów następuje po ponownym uruchomieniu wyszukiwania.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może wskazać miasto jako kryterium wyszukiwania. • Użytkownik może wybrać jeden lub wiele tagów jako filtr. • Zmiana miasta nie powoduje automatycznego odświeżenia wyników. • Zmiana tagów powoduje automatyczne odświeżenie wyników. • Po ponownym uruchomieniu wyszukiwania (kliknięciu przycisku wyszukiwania) lista wyników jest aktualizowana zgodnie z wybranymi filtrami.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-02 , WFWYSZ-03 , WFWYSZ-04 , WFWYSZ-08 , WFWYSZ-07 , WPWYSZ-01 , WPWYSZ-02 .

Tabela 4.101: Wymaganie funkcjonalne dla wyszukiwarki spotów: Wyszukiwanie za pomocą miasta oraz tagów

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW		
Identyfikator:	WFWYSZ-07	Priorytet: M
Nazwa:	Sortowanie po popularności oraz ocenie	
Opis:	System umożliwia użytkownikowi sortowanie listy spotów według popularności oraz średniej z opinii.	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Użytkownik może ustawić kryterium, wybierając jedną z następujących opcji: <ul style="list-style-type: none"> – sortowanie według popularności rosnąco (od najmniej do najbardziej popularnych), – sortowanie według popularności malejąco (od najbardziej do najmniej popularnych), – sortowanie według średniej ocen rosnąco (od najmniej do najbardziej ocenionych), – sortowanie według średniej ocen malejąco (od najbardziej do najmniej ocenionych), • Po zastosowaniu sortowania lista wyników jest aktualizowana zgodnie z ustawieniami. 	
Udziałowiec:	U3	
Realizator:	Mateusz Redosz	
Wymagania powiązane:	WOWYSZ-02 , WFWYSZ-03 , WFWYSZ-08 , WPWYSZ-01 .	

Tabela 4.102: Wymaganie funkcjonalne dla wyszukiwarki spotów: Sortowanie

po popularności oraz ocenie

KARTA WYMAGANIA FUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW		
Identyfikator:	WFWYSZ-08	Priorytet: M
Nazwa:	Filtrowanie po ocenie	
Opis:	System umożliwia użytkownikowi filtrowanie wyników wyszukiwania według minimalnej oceny spota, aby szybciej odnajdować miejsca o wysokiej jakości.	
Kryteria akceptacji:	<ul style="list-style-type: none">Użytkownik może ustawić minimalną ocenę (wybór wartości).Lista wyników po ustawieniu filtra prezentuje wyłącznie spoty spełniające warunek minimalnej oceny.	
Udziałowiec:	U3	
Realizator:	Mateusz Redosz	
Wymagania powiązane:	WOWYSZ-02 , WFWYSZ-03 , WPWYSZ-01 .	

Tabela 4.103: Wymaganie funkcjonalne dla wyszukiwarki spotów: Filtrowanie po ocenie

4.2.2.7 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>jasny</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 darkMode: 'class'; motyw przełączany przez dodanie/usunięcie klasy dark na elemencie <html>;		
Udziałowiec:	Zespół projektowy 2.1 , promotor 2.2 , dronarze 2.3 .		
Wymagania powiązane:			

Tabela 4.118: 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 , drona-rze 2.3 .		
Wymagania powiązane:			

Tabela 4.119: 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 <i>dark</i> na <i>document.documentElement</i> oraz aktualizuje <i>localStorage (theme = 'dark' 'light')</i> ; brak przeładowania strony.		
Udziałowiec:	Zespół projektowy 2.1 , promotor 2.2 , droniarze 2.3 .		
Wymagania powiązane:			

Tabela 4.120: Szybki przełącznik motywu w interfejsie

4.2.3 Wymagania pozafunkcjonalne

Niniejszy rozdział zawiera wymagania pozafunkcjonalne postawione systemowi. Został on podzielony tematycznie.

4.2.3.1 Wymagania pozafunkcjonalne dla czatu

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU			
Identyfikator:	WPCZAT-01	Priorytet:	S
Nazwa:	Dostęp do czatów ograniczony do uczestników (autoryzacja)		
Typ:	Bezpieczeństwo		

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU (cd.)

Opis:	Wymaganie dotyczy autoryzacji na poziomie pojedynczych czatów . System zapewnia, że zalogowany użytkownik widzi wyłącznie listę czatów oraz wiadomości z czatów, których jest uczestnikiem. Informacje o innych czatach nie są prezentowane w interfejsie ani dostępne poprzez API , nawet jeśli użytkownik zna ich identyfikatory.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Dla zalogowanego użytkownika lista czatów zawiera wyłącznie czaty, w których jest on uczestnikiem. • Zapytania do API odwołujące się do czatu, którego użytkownik nie jest członkiem, są odrzucane (kodem 403) i nie zwracają żadnych danych o tym czacie ani jego wiadomościach.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01 , WOCZAT-03 .

Tabela 4.104: Wymaganie pozafunkcjonalne dla czatu: Dostęp do czatów ograniczony do uczestników (autoryzacja)

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU			
Identyfikator:	WPCZAT-02	Priorytet:	S
Nazwa:	Korzystanie z czatu wymaga zalogowania (uwierzytelnienie)		
Typ:	Bezpieczeństwo		

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU (cd.)

Opis:	Wymaganie dotyczy uwierzytelnienia . Jakakolwiek próba skorzystania z modułu czatu (wejście na widok czatu, pobieranie listy czatów, wysyłanie lub odbieranie wiadomości, tworzenie czatów) wymaga wcześniejszego zalogowania się do systemu. Użytkownik niezalogowany w ogóle nie może uzyskać dostępu do danych czatu.
Kryteria akceptacji:	<ul style="list-style-type: none"> • Wejście na widok czatu przez użytkownika niezalogowanego powoduje przekierowanie na ekran logowania lub wyświetlenie komunikatu o braku uprawnień. • Zapytania do API czatu wykonane bez ważnego JWT są odrzucone kodem 401 i nie zwracają żadnych danych.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01 , WOCZAT-02 , WOCZAT-03 , WOCZAT-04 .

Tabela 4.105: Wymaganie pozafunkcjonalne dla czatu: Korzystanie z czatu wymaga zalogowania (uwierzytelnienie)

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU		
Identyfikator:	WPCZAT-03	Priorytet: S
Nazwa:	Grupowanie wiadomości według daty wysłania	
Typ:	Użyteczność	
Opis:	Wiadomości na czacie są prezentowane w logicznych grupach odpowiadających datom ich wysłania, co ułatwia użytkownikom orientację w historii rozmowy.	

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU (cd.)

Kryteria akceptacji:	<ul style="list-style-type: none"> • W widoku czatu pojawiają się wizualne znaczniki dat. • Wiadomości są zawsze przypisane do poprawnej grupy daty wysłania niezależnie od strefy czasowej klienta. • Zmiana zakresu historii (scrollowanie, przeładowanie) zachowuje poprawne grupowanie dat.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01 , WOCZAT-03 .

Tabela 4.106: Wymaganie pozafunkcjonalne dla czatu: Grupowanie wiadomości według daty wysłania

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU		
Identyfikator:	WPCZAT-04	Priorytet: S
Nazwa:	Wyraźne oznaczenie nadawcy i czasu wysłania	
Typ:	Użyteczność	
Opis:	Każda wiadomość na czacie jest opatrzona wyraźną informacją, kto jest jej nadawcą oraz kiedy została wysłana. Informacje te są łatwo zauważalne i spójne wizualnie.	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Przy każdej wiadomości widoczna jest nazwa lub alias nadawcy. • Po najechaniu kursem na daną wiadomość widoczna jest godzina jej wysłania. 	
Udziałowiec:	U3	
Realizator:	Adam Langmesser	

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU (cd.)	
Wymagania powiązane:	WOCZAT-01 , WOCZAT-03 .

Tabela 4.107: Wymaganie pozafunkcjonalne dla czatu: Wyraźne oznaczenie nadawcy i czasu wysłania

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU			
Identyfikator:	WPCZAT-05	Priorytet:	S
Nazwa:	Czas załadowania starszych wiadomości poniżej 10 sekund		
Typ:	Wydajność		
Opis:	Podczas przewijania historii czatu załadowanie kolejnej partii starszych wiadomości powinno trwać krócej niż 10 sekundy w typowych warunkach sieciowych.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • W co najmniej 95% pomiarów w warunkach deweloperskich czas pobrania starszych wiadomości mieści się w przedziale 0–10 s. • Interfejs wyraźnie sygnalizuje trwające ładowanie. • Brak zauważalnych „zawieszeń” interfejsu podczas operacji pobierania danych. 		
Udziałowiec:	U3		
Realizator:	Adam Langmesser		
Wymagania powiązane:	WOCZAT-03 .		

Tabela 4.108: Wymaganie pozafunkcjonalne dla czatu: Czas załadowania starszych wiadomości poniżej 10 sekund

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU			
Identyfikator:	WPCZAT-06	Priorytet:	S
Nazwa:	Natychmiastowe wysyłanie wiadomości		
Typ:	Wydajność		
Opis:	Po wysłaniu wiadomości przez użytkownika powinna ona pojawić się w widoku czatu w czasie subiektywnie natychmiastowym (rzędu setek milisekund), a pozostali uczestnicy powinni ją zobaczyć w czasie zbliżonym do rzeczywistego.		
Kryteria akceptacji:	<ul style="list-style-type: none"> • W typowych warunkach sieciowych użytkownik widzi swoją nową wiadomość w czasie poniżej 1 s od wysłania. • Pozostali uczestnicy czatu otrzymują wiadomość bez konieczności ręcznego odświeżania. 		
Udziałowiec:	U3		
Realizator:	Adam Langmesser		
Wymagania powiązane:	WOCZAT-01 .		

Tabela 4.109: Wymaganie pozafunkcjonalne dla czatu: Natychmiastowe wysyłanie wiadomości

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU			
Identyfikator:	WPCZAT-07	Priorytet:	S
Nazwa:	Integracja z API Tenor		
Typ:	Użyteczność		
Opis:	GIF'y są pobierane z API Tenor, który jest ich popularnym dostawcą. Zastosowanie tego rozwiązania znacząco ułatwi użytkownikom korzystanie z czatu.		

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA CZATU (cd.)	
Kryteria akceptacji:	<ul style="list-style-type: none"> • GIF'Y dostępne w czacie są pobierane z API Tenor.
Udziałowiec:	U1
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01.

Tabela 4.110: Wymaganie pozafunkcjonalne dla czatu: Integracja z API Tenor

4.2.3.2 Wymagania pozafunkcjonalne dla mapy

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA MAPY		
Identyfikator:	WP MAPA-01	Priorytet: S
Nazwa:	Hostowanie mapy na niezależnym serwerze	
Typ:	Niezawodność	
Opis:	Mapa jest hostowana na serwerze zarządzanym przez zespół projektowy. Korzystanie z własnego hosta zamiast publicznej instancji dostarczanej przez producenta mapy pozwoli uniknąć jego nadmiernego obciążenia, powodującego zwiększenie czasu potrzebnego na pobranie kafelków mapy.	
Kryteria akceptacji:	<ul style="list-style-type: none"> • Mapa jest hostowana na niezależnym serwerze. • Serwer jest zarządzany przez zespół projektowy. • Czas pobierania kafelków mapy z serwera jest nie większy niż z serwera dostarczanego przez producenta mapy. 	
Udziałowiec:	U2	
Realizator:	Stanisław Oziemczuk	

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA MAPY (cd.)	
Wymagania powiązane:	WOMAPA-01

Tabela 4.111: Wymaganie pozafunkcjonalne dla mapy: Hostowanie mapy na niezależnym serwerze

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA MAPY	
Identyfikator:	WPMAPA-02
Priorytet:	S
Nazwa:	Czas ładowania mapy poniżej 10 sekund
Typ:	Wydajność
Opis:	Ładowanie kafelków mapy podczas pierwszego wejścia na stronę zajmuje maksymalnie 10 sekund przy standardowym połączeniu sieciowym.
Kryteria akceptacji:	<ul style="list-style-type: none"> • W przynajmniej 95% przypadków załadowanie kafelek mapy w widocznym jej obszarze trwa mniej niż 10 sekund.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01

Tabela 4.112: Wymaganie pozafunkcjonalne dla mapy: Czas ładowania mapy poniżej 10 sekund

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA MAPY	
Identyfikator:	WPMAPA-03
Priorytet:	S
Nazwa:	Czas pobierania spotów poniżej 10 sekund
Typ:	Wydajność

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA MAPY (cd.)	
Opis:	Pobieranie spotów z backendu i wyświetlenie ich na mapie trwa maksymalnie 10 sekund przy standardowym połączeniu sieciowym.
Kryteria akceptacji:	<ul style="list-style-type: none"> • W przynajmniej 95% przypadków pobranie i wyświetlenie spotów zajmuje mniej niż 10 sekund.
Udziałowiec:	U3
Realizator:	Stanisław Oziemczuk
Wymagania powiązane:	WOMAPA-01

Tabela 4.113: Wymaganie pozafunkcjonalne dla mapy: Czas pobierania [spotów](#) poniżej 10 sekund

4.2.3.3 Wymagania pozafunkcjonalne dla wyszukiwarki spotów

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW			
Identyfikator:	WPWYSZ-01	Priorytet:	M
Nazwa:	Czas ładowania wyszukanych spotów nie przekracza 10 sekund		
Typ:	Wydajność		
Opis:	System powinien zwracać wyniki wyszukiwania spotów w czasie nie dłuższym niż 10 sekund w typowych warunkach sieciowych, aby użytkownik mógł sprawnie przeglądać wyniki.		

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW (cd.)	
Kryteria akceptacji:	<ul style="list-style-type: none"> • W co najmniej 95% pomiarów w warunkach deweloperskich czas odpowiedzi dla wyszukiwania mieści się w przedziale 0–10 s. • W trakcie pobierania danych interfejs wyświetla informację o ładowaniu. • W przypadku błędu połączenia użytkownik otrzymuje czytelny komunikat.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WOWYSZ-02 , WFWYSZ-01 , WFWYSZ-02 , WFWYSZ-03 , WFWYSZ-06 , WFWYSZ-07 , WFWYSZ-08 .

Tabela 4.114: Wymaganie pozafunkcjonalne dla wyszukiwarki spotów: Czas ładowania wyszukanych spotów nie przekracza 10 sekund

KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA WYSZUKIWARKI SPOTÓW	
Identyfikator:	WPWYSZ-02
Priorytet:	M
Nazwa:	Podpowiedzi w polach lokalizacji (kraj/region/miasto)
Typ:	Użyteczność
Opis:	W polach wyboru lokalizacji system prezentuje podpowiedzi dopasowane do wprowadzanej frazy, co ułatwia szybkie i bezbłędne uzupełnianie filtrów.

**KARTA WYMAGANIA POZAFUNKCJONALNEGO DLA
WYSZUKIWARKI SPOTÓW (cd.)**

Kryteria akceptacji:	<ul style="list-style-type: none"> ● Podczas wpisywania frazy w polu (kraj/region/miasto) pojawia się lista pasujących propozycji. ● Lista propozycji zawiera tylko wartości dostępne w systemie. ● Użytkownik może wybrać odpowiedź myszą.
Udziałowiec:	U3
Realizator:	Mateusz Redosz
Wymagania powiązane:	WOWYSZ-01 , WOWYSZ-02 , WFWYSZ-02 , WFWYSZ-04 , WFWYSZ-06 .

Tabela 4.115: Wymaganie pozafunkcjonalne dla wyszukiwarki spotów: Podpowiedzi w polach lokalizacji (kraj/region/miasto)

Rozdział 5

Projekt

5.1 Architektura systemu

W niniejszym rozdziale przedstawiona zostanie architektura systemu, czyli sposób, w jaki poszczególne komponenty komunikują się między sobą, a także technologie, za pomocą których zostały stworzone.

Jednym z kluczowych etapów realizacji projektu był wybór odpowiedniej architektury systemowej. Ostatecznie przyjęto oddzielenie poszczególnych warstw aplikacji, co zapewnia większą elastyczność, skalowalność oraz ułatwia rozwój w przyszłości. Przyjęte komponenty prezentują się następująco:

- frontend – React z wykorzystaniem TypeScriptu,
- backend – Java Spring Boot,
- baza danych – PostgreSQL,
- redis – wykorzystywany jako baza danych klucz-wartość pełniąca rolę warstwy cache.

Jest to podejście, w którym zespół projektowy posiada największe doświadczenie, dlatego zostało ono zastosowane. Pozwala ono również na tworzenie aplikacji responsywnej, dostępnej zarówno na komputerach, jak i urządzeniach mobilnych. Warstwa wizualna została przygotowana przy użyciu React w wersji z TypeScriptem oraz biblioteki Tailwind CSS, zapewniającej szybkie i wygodne stylowanie

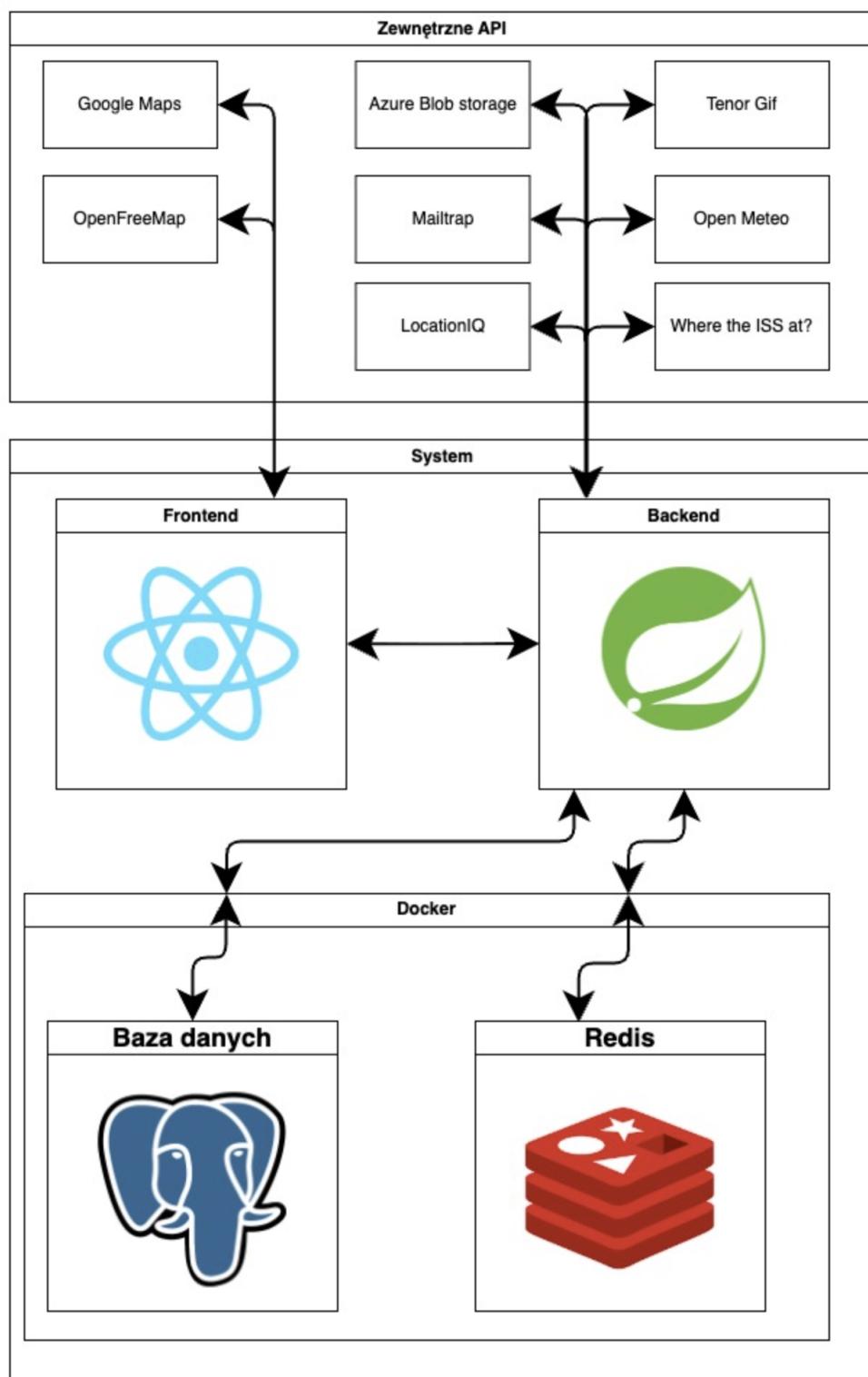
komponentów. Z kolei za komunikację oraz logikę biznesową odpowiada **backend** oparty na **frameworku** Spring Boot, realizujący założenia architektury **REST API**. Jako system zarządzania danymi wybrano relacyjną bazę danych PostgreSQL, z którą zespół posiada największe doświadczenie. Relacyjny model danych doskonale sprawdza się w tym projekcie, zapewniając integralność danych, możliwość tworzenia złożonych zapytań oraz wysoką stabilność.

Redis został wykorzystany jako warstwa **cache**, której zadaniem jest przyspieszenie działania aplikacji poprzez ograniczenie liczby odwołań do głównej **bazy danych**. Dzięki przechowywaniu często wykorzystywanych danych w pamięci operacyjnej znacznie skraca się czas odpowiedzi systemu, co pozytywnie wpływa na wydajność oraz skalowalność rozwiązania. Zastosowanie **Redisa** okazało się szczególnie korzystne w przypadku operacji powtarzalnych i odczytowych, które nie wymagają każdorazowego dostępu do relacyjnej **bazy danych**.

5.1.1 Diagram architektury

Projekt aplikacji oparto na architekturze klient–serwer z podziałem na **frontend** i **backend**. Takie podejście ułatwia rozwój i utrzymanie systemu oraz umożliwia skalowanie poszczególnych komponentów niezależnie od siebie. Komunikacja między **frontendem** a **backendem** odbywa się za pomocą **REST API**, przy czym dane przesyłane są w formacie JSON. Integracja między warstwami aplikacji jest dzięki temu lekka, czytelna i łatwa do rozszerzenia w przyszłości.

Architektura została opracowana dla środowiska deweloperskiego. W obecnym zakresie prac nie uwzględniono implementacji środowiska produkcyjnego.



Rysunek 5.1: Diagram architektury

5.1.2 Komponenty systemu

System składa się z kilku głównych komponentów, z których każdy pełni ściśle określona rolę.

- **Frontend** – odpowiada za warstwę prezentacji oraz interfejs użytkownika dostępny dla wszystkich użytkowników systemu,
- **Backend** – odpowiada za autoryzację użytkowników oraz obsługę komunikacji między **frontendem** a **bazą danych**,
- **Baza danych** – przechowuje wszystkie dane aplikacji, w tym dane użytkowników, dane operacyjne oraz informacje potrzebne do działania systemu.
- **Redis** – wykorzystywany jako warstwa cache, przechowująca często odczytywane dane w pamięci operacyjnej, co znacząco przyspiesza działanie systemu oraz zmniejsza obciążenie głównej bazy danych.

Szczegółowy wykaz wykorzystywanych zewnętrznych API zamieszczono w rozdziale 3.

- Azure Blob Storage – [3.4](#)
- Mailtrap – [3.5](#)
- LocationIQ – [3.6](#)
- Google Maps – [3.7](#)
- OpenFreeMap – [3.8](#)
- Open Meteo – [3.9](#)
- Tenor Gif – [3.10](#)
- Where the ISS at? – [3.11](#)

5.2 Projekt bazy danych

5.2.1 Model danych

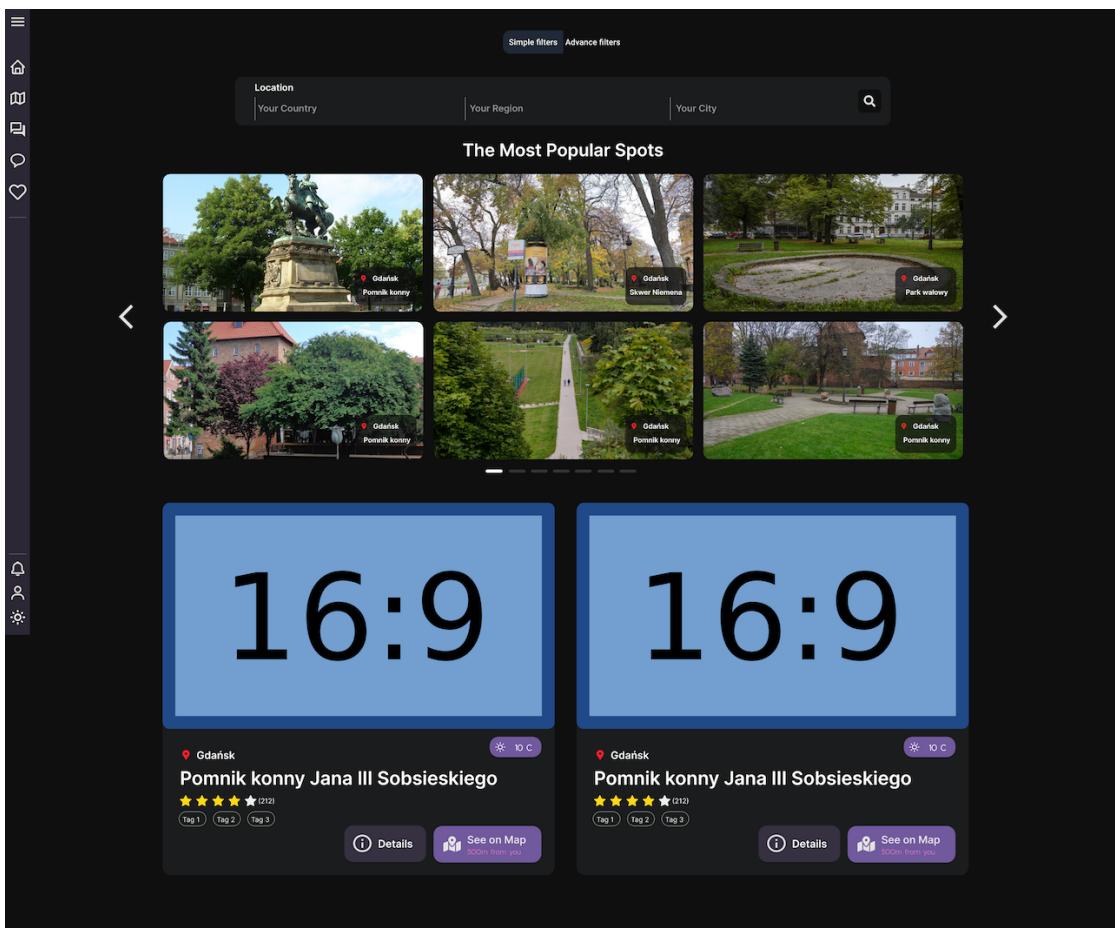
5.2.2 Diagram ERD

5.3 Architektura interfejsu użytkownika

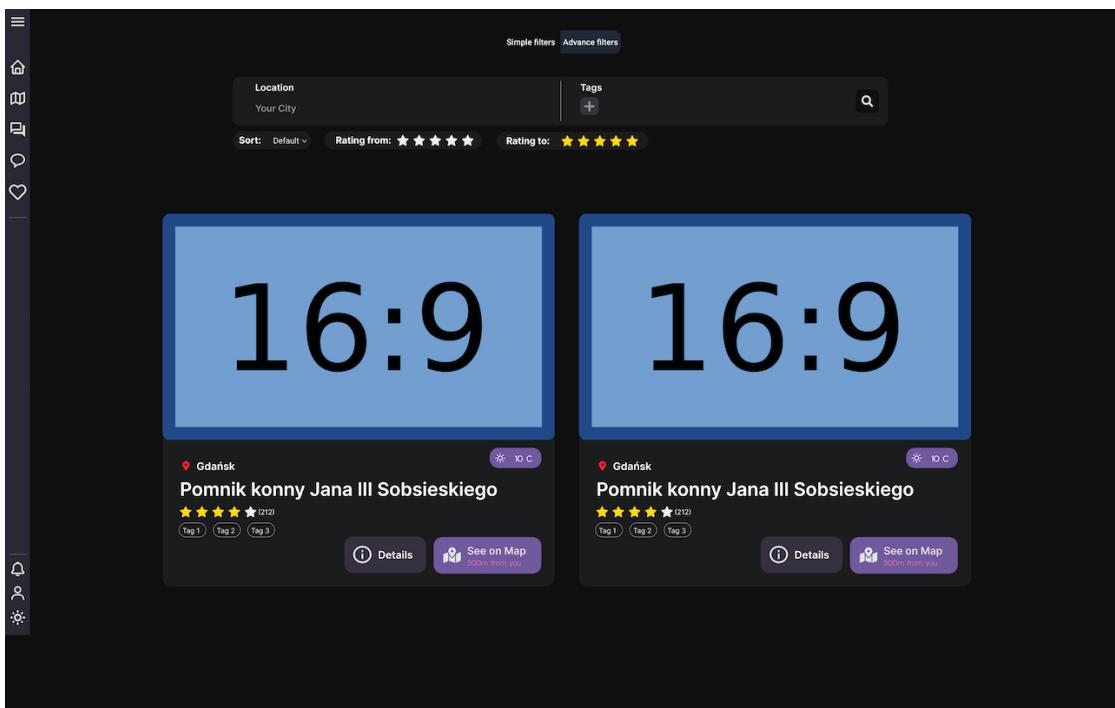
Niniejszy rozdział opisuje projekt interfejsu użytkownika stworzony w figmie w ramach przedmiotu PRZ1.

5.3.1 Projekt strony głównej

Na potrzeby strony głównej aplikacji zaprojektowano dwa warianty widoku: prosty (rys. 5.2) oraz zaawansowany (rys. 5.3), tak aby możliwe było dopasowanie sposobu wyszukiwania do preferencji użytkownika.

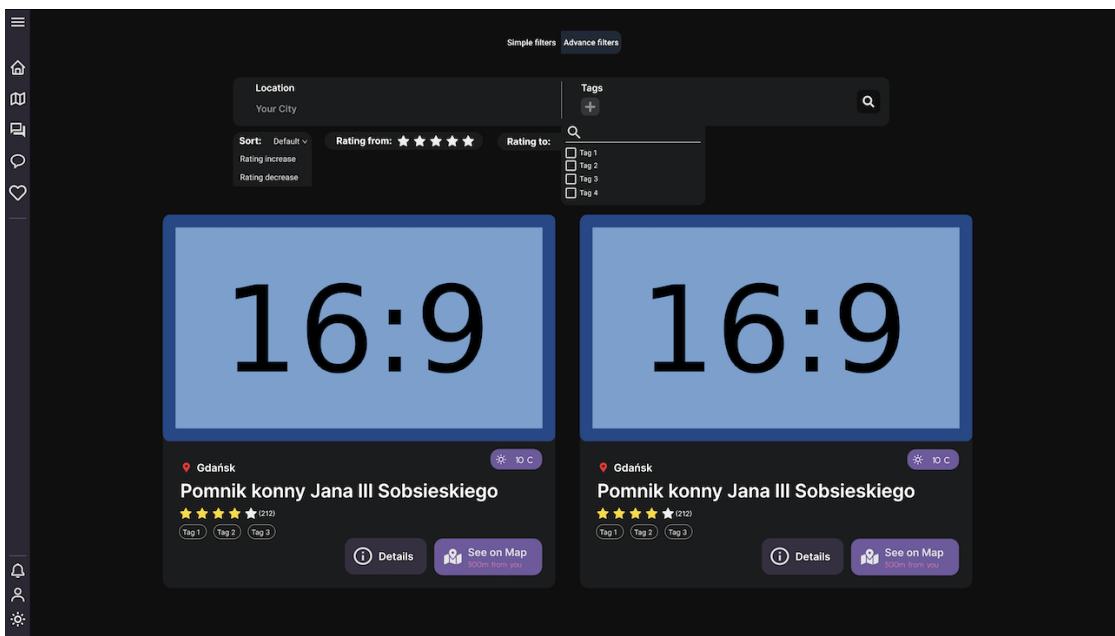


Rysunek 5.2: Projekt prostej strony głównej



Rysunek 5.3: Projekt zaawansowanej strony głównej (1)

W obu widokach, w górnej części strony, umieszczono przyciski służące do przełączania się między trybem prostym a zaawansowanym. Poniżej znajduje się wyszukiwarka [spotów](#), której układ zależy od wybranego trybu. Dla prostego widoku przewidziano trzy pola tekstowe do podania kolejno: kraju, regionu oraz miasta; każde z nich zawiera również listę z podpowiedziami odpowiadających im lokalizacji. Dla widoku zaawansowanego zaprojektowano pojedyncze pole do wpisania miasta oraz listę umożliwiającą wybór dostępnych tagów. Dodatkowo udostępniono możliwość sortowania wyników według oceny oraz filtrowania ich z uwzględnieniem minimalnej i maksymalnej oceny (rys. 5.4).



Rysunek 5.4: Projekt zaawansowanej strony głównej (2)

W trybie prostym przewidziano karuzelę z najpopularniejszymi [spotami](#), dzięki czemu możliwe jest szybkie odnalezienie interesujących miejsc bez konieczności przeglądania całej listy wyników.

Na dole strony umieszczono listę wyszukanych [spotów](#). Każdy kafelek reprezentujący [spot](#) zawiera następujące informacje:

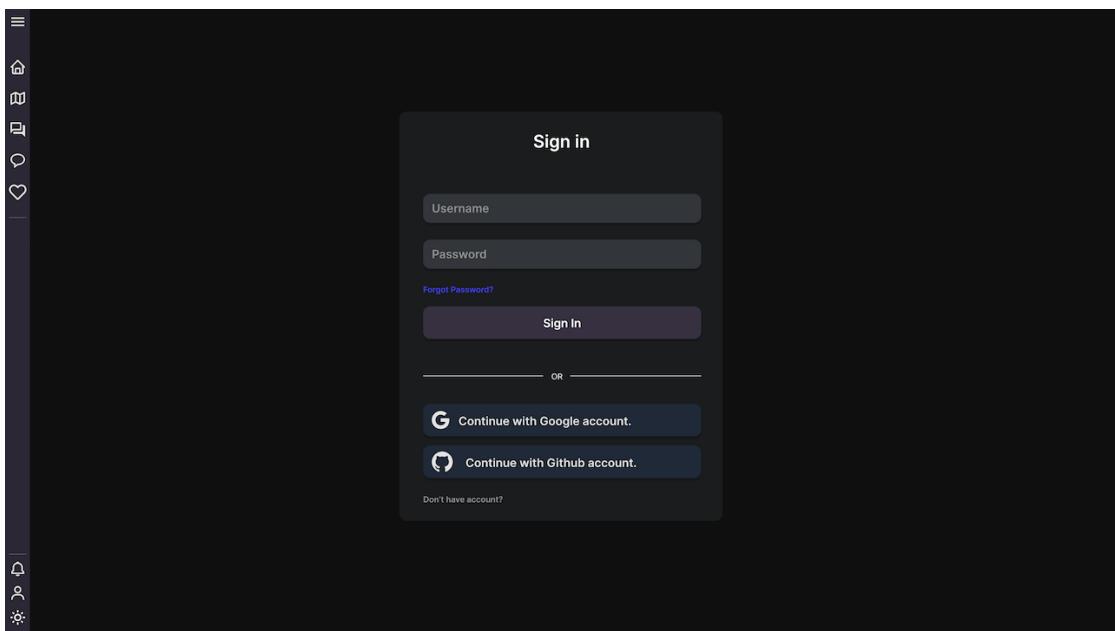
- nazwę miasta, w którym znajduje się [spot](#),
- nazwę [spota](#),
- informacje pogodowe (typ pogody oraz aktualna temperatura),
- średnią ocenę oraz liczbę wystawionych opinii,
- listę tagów opisujących [spota](#),
- przycisk przejścia do widoku szczegółów,
- przycisk wyświetlenia [spota](#) na mapie wraz z informacją o przybliżonej odległości od bieżącej lokalizacji użytkownika.

Oba warianty strony utrzymano w ciemnej kolorystyce, spójnej z pozostałymi elementami UI aplikacji. Układ komponentów zaplanowano tak, aby zachować czytelność i hierarchię informacji, a także umożliwić wygodne korzystanie z aplikacji na ekranach o różnej rozdzielczości.

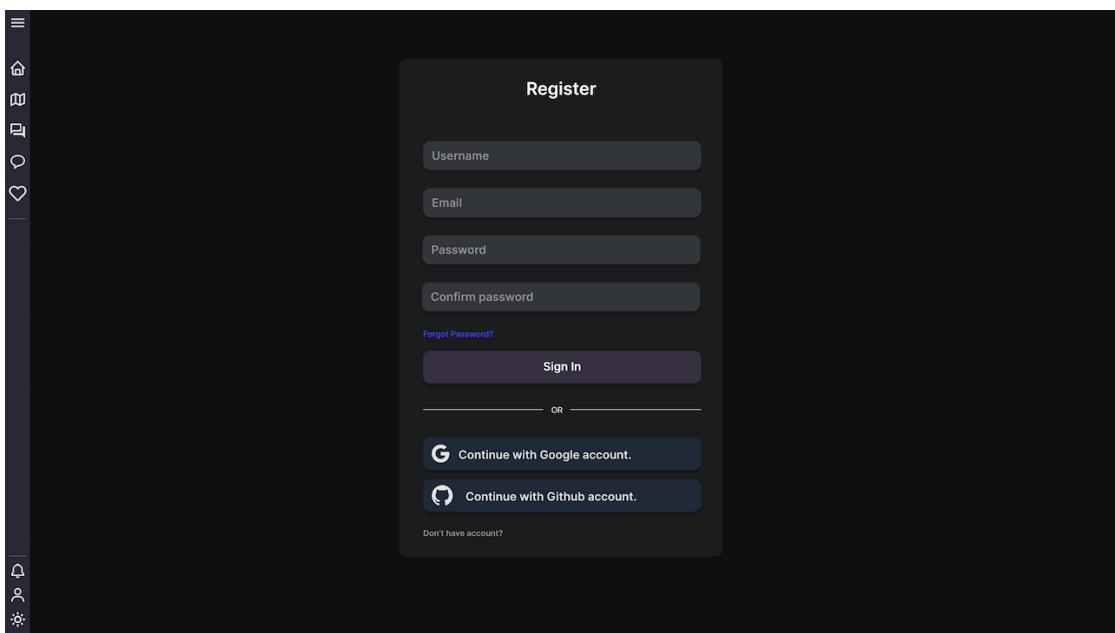
5.3.2 Projekt panelu logowania

W ramach przedmiotu PRZ1 zaprojektowano wygląd strony z formularzem logowania (rys. 5.5) oraz rejestracji (rys. 5.6). Dla zachowania prostoty i wygody użytkownika oba widoki mają niemal identyczny układ; w formularzu rejestracji dodano jedynie dwa dodatkowe pola: adres e-mail oraz pole służące do potwierdzania hasła. Na obu stronach umieszczono również przyciski umożliwiające logowanie oraz rejestrację z wykorzystaniem mechanizmu OAuth dla kont Google oraz GitHub.

Pod formularzem logowania przewidziano odnośnik do przypomnienia hasła oraz przycisk przejścia do formularza rejestracji, natomiast na stronie rejestracji umieszczono przycisk powrotu do formularza logowania. Formularze zaprojektowano w ciemnej kolorystyce, z centralnym ułożeniem panelu i wyraźnym rozdzielением elementów interaktywnych.



Rysunek 5.5: Projekt strony logowania



Rysunek 5.6: Projekt strony rejestracji

5.3.3 Projekt mapy

5.3.4 Projekt chatu

5.3.5 Projekt forum

5.3.6 Projekt panelu użytkownika

Zaprojektowano również panel użytkownika, który składa się z ośmiu podstron:

- **Profile** – profil użytkownika,
- **Spots list** – lista [spotów](#) dodanych do różnych list (np. ulubionych, do powrotnego odwiedzenia),
- **Photos list** – lista zdjęć, które użytkownik dodał do [spotów](#), komentarzy pod spotami oraz wpisów na forum,
- **Movies list** – lista filmów, które użytkownik dodał do [spotów](#), komentarzy pod spotami oraz wpisów na forum,
- **Friends** – lista znajomych,
- **Add spot** – lista [spotów](#) dodanych przez użytkownika oraz formularz pozwalający na dodanie nowego [spota](#),
- **Comments** – lista komentarzy dodanych przez użytkownika pod [spotami](#),
- **Settings** – ustawienia konta (zmiana nazwy użytkownika, adresu e-mail oraz hasła).

5.3.6.1 Profile

Profil użytkownika składa się z dwóch wariantów widoku: prywatnego (rys. 5.7), przeznaczonego dla właściciela konta, oraz publicznego (rys. 5.8), widocznego dla innych użytkowników. Pod względem wizualnym oba widoki są niemal identyczne.

Na obu widokach umieszczone duże, okrągłe zdjęcie profilowe użytkownika, jego nazwę oraz statystyki obejmujące liczbę obserwujących, obserwowanych, znajomych oraz dodanych zdjęć. Poniżej, w widoku publicznym, znajdują się przyciski umożliwiające:

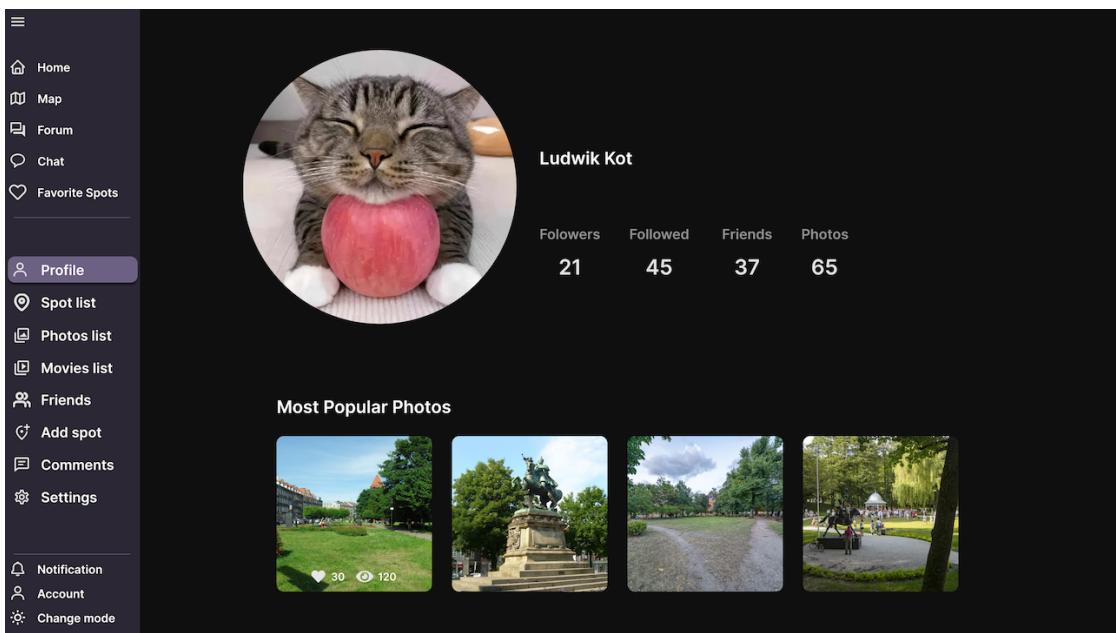
- dodanie lub usunięcie danej osoby z listy znajomych (rys. 5.8, 5.14),
- rozpoczęcie lub zakończenie obserwowania profilu (rys. 5.8, 5.15).

Po wysłaniu zaproszenia do znajomych treść odpowiedniego przycisku zmienia się na „Request send” (rys. 5.13). Na dole widoku wyświetlna jest lista czterech najpopularniejszych zdjęć danego użytkownika.

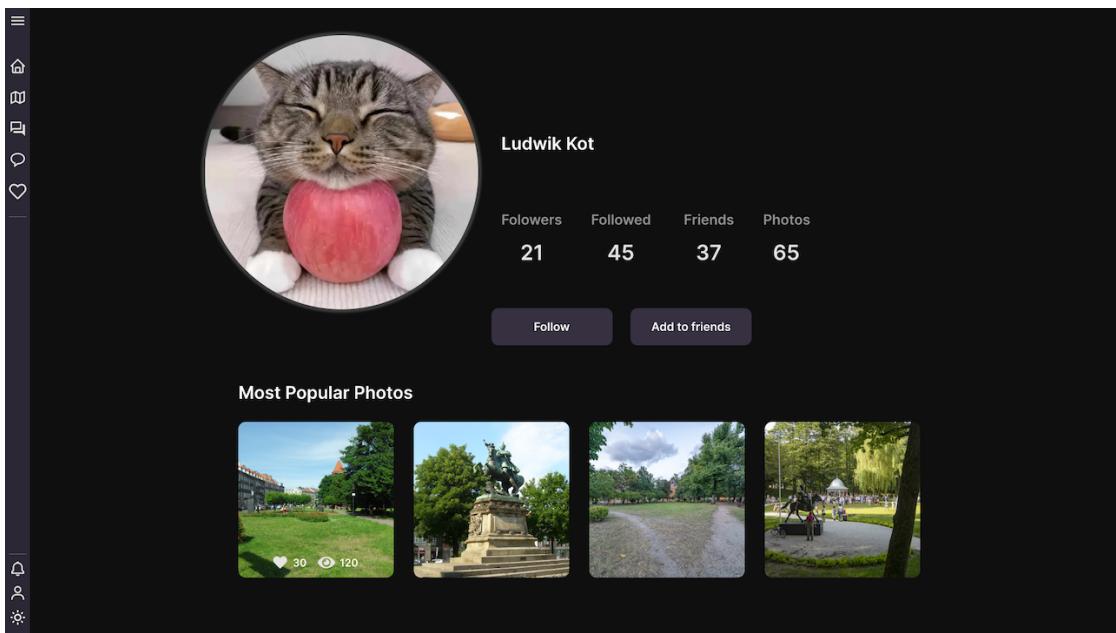
Kliknięcie którejkolwiek z wartości statystyk w widoku publicznym powoduje przejście do odpowiednich list:

- lista obserwujących – rys. 5.9,
- lista obserwowanych – rys. 5.10,
- lista znajomych – rys. 5.11,
- lista zdjęć – rys. 5.12.

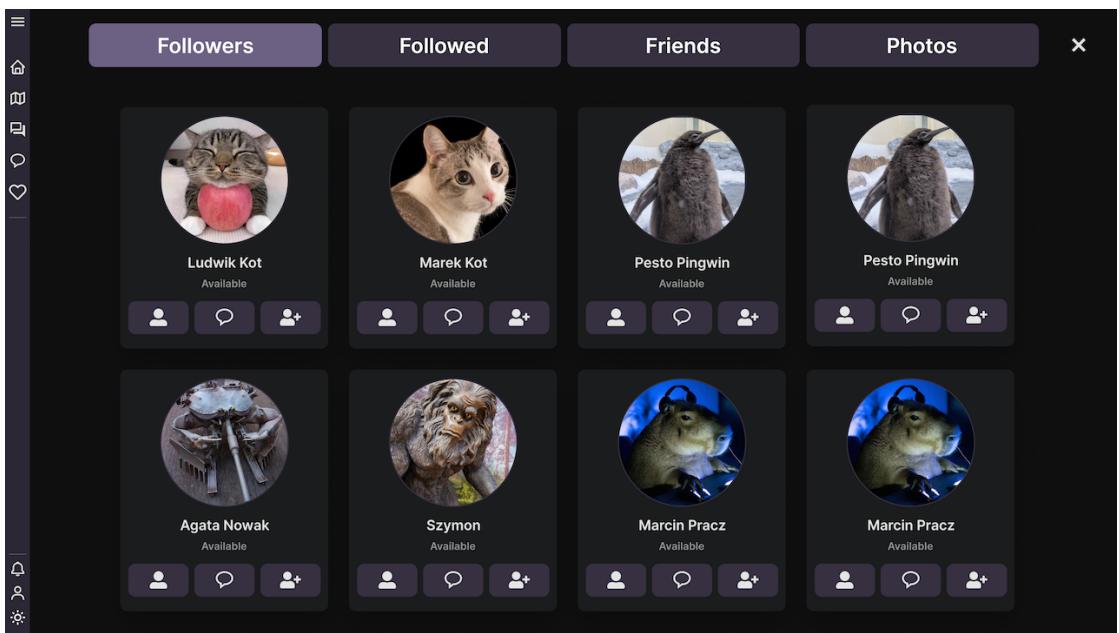
Analogiczne działanie przewidziano dla profilu własnego użytkownika. Po wybraniu statystyk związanych ze znajomymi lub zdjęciami następuje przejście odpowiednio do list *Friends* (rys. 5.20) oraz *Photos* (rys. 7.102).



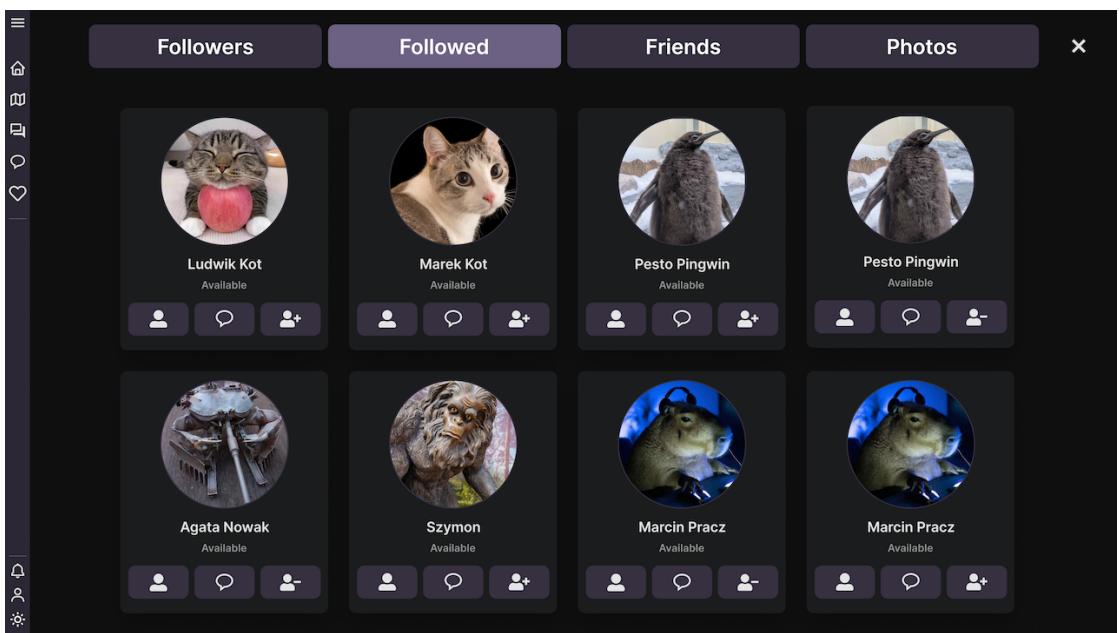
Rysunek 5.7: Widok prywatny profilu



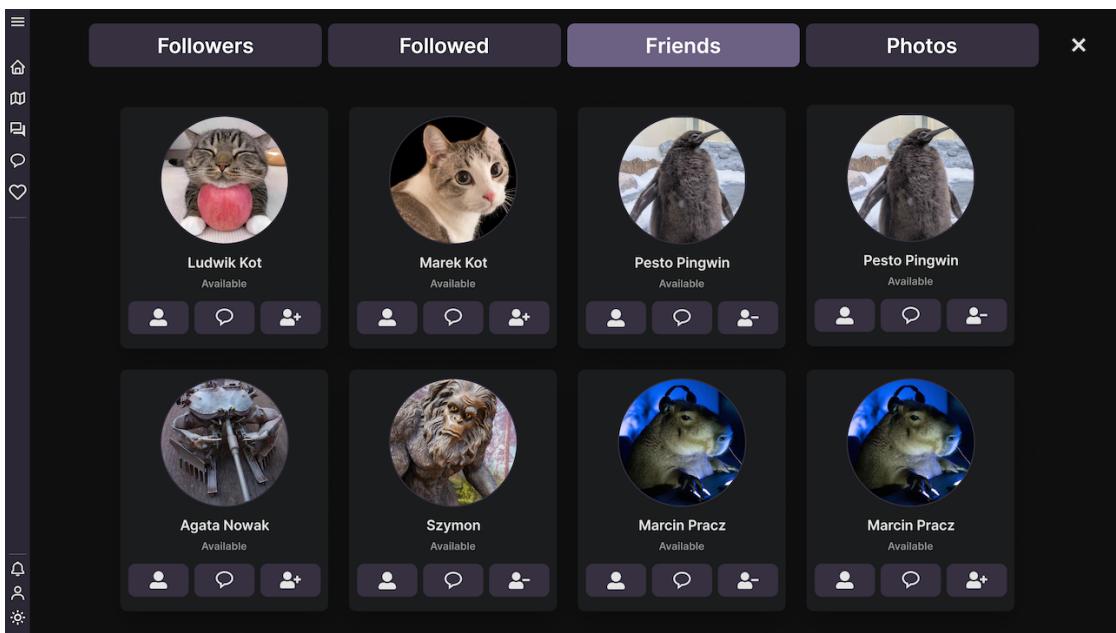
Rysunek 5.8: Widok publiczny profilu



Rysunek 5.9: Lista obserwujących



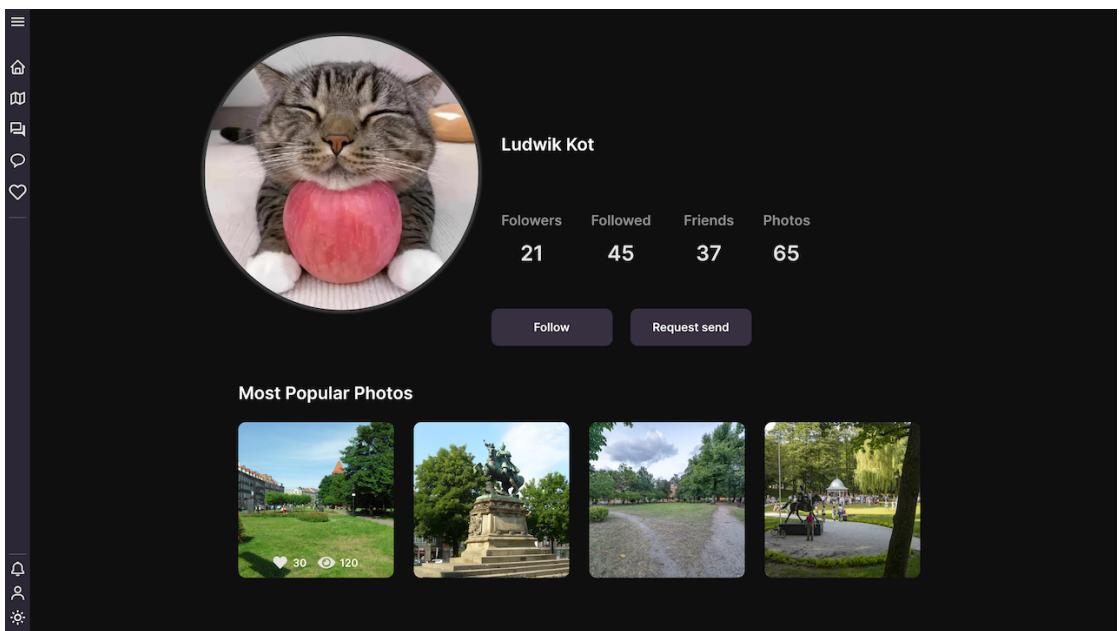
Rysunek 5.10: Lista obserwowanych



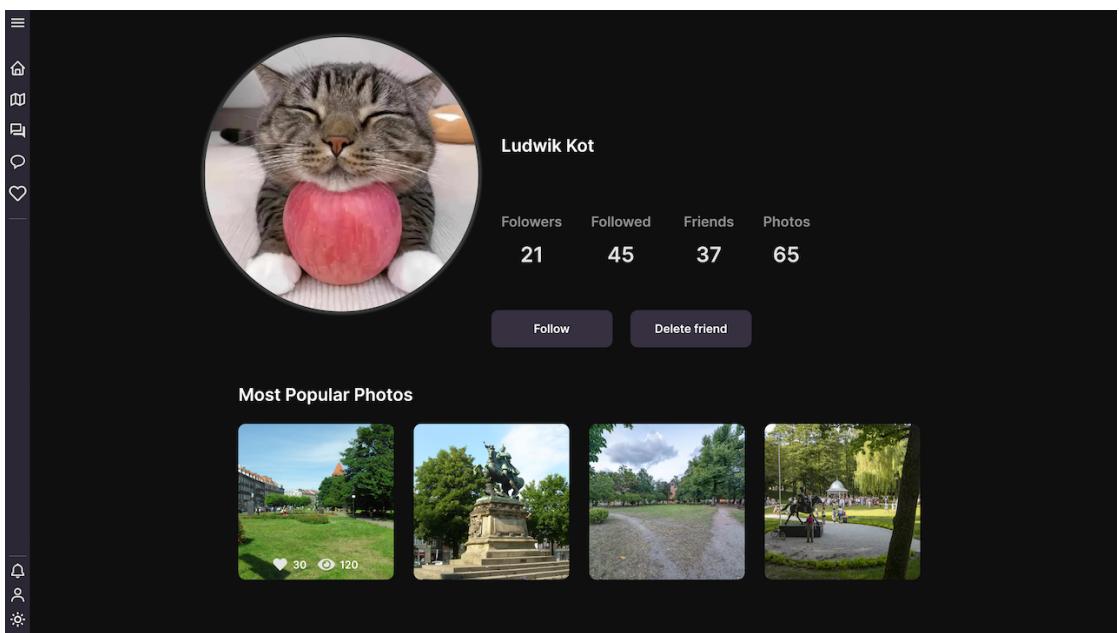
Rysunek 5.11: Lista znajomych



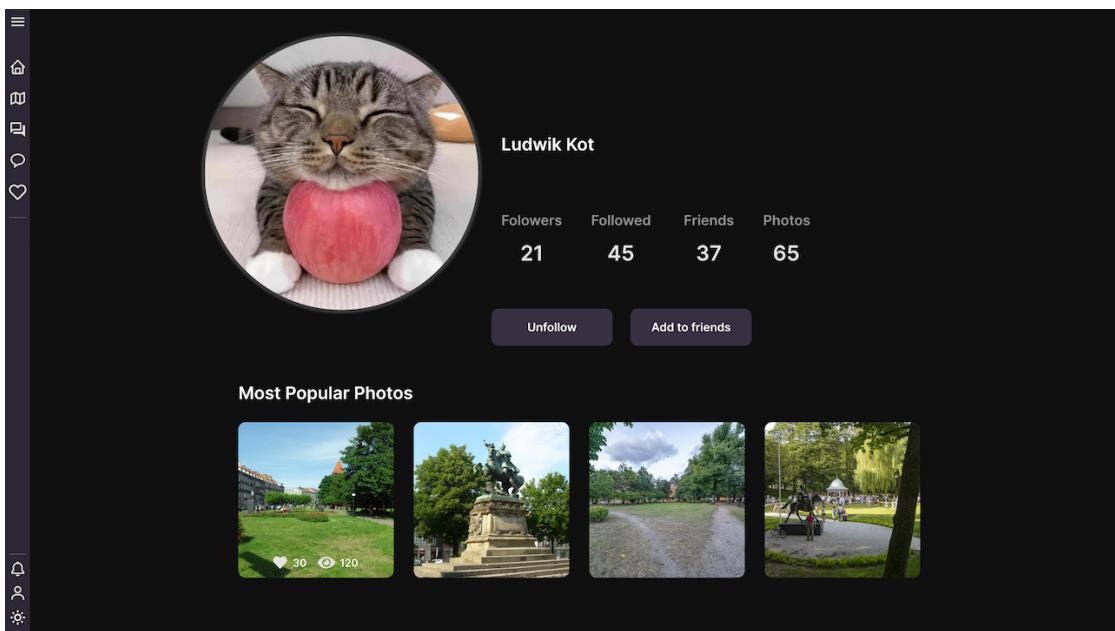
Rysunek 5.12: Lista zdjęć użytkownika



Rysunek 5.13: Zaproszenie wysłane



Rysunek 5.14: Możliwość usunięcia z listy znajomych



Rysunek 5.15: Możliwość zakończenia obserwowania

5.3.6.2 Spots list

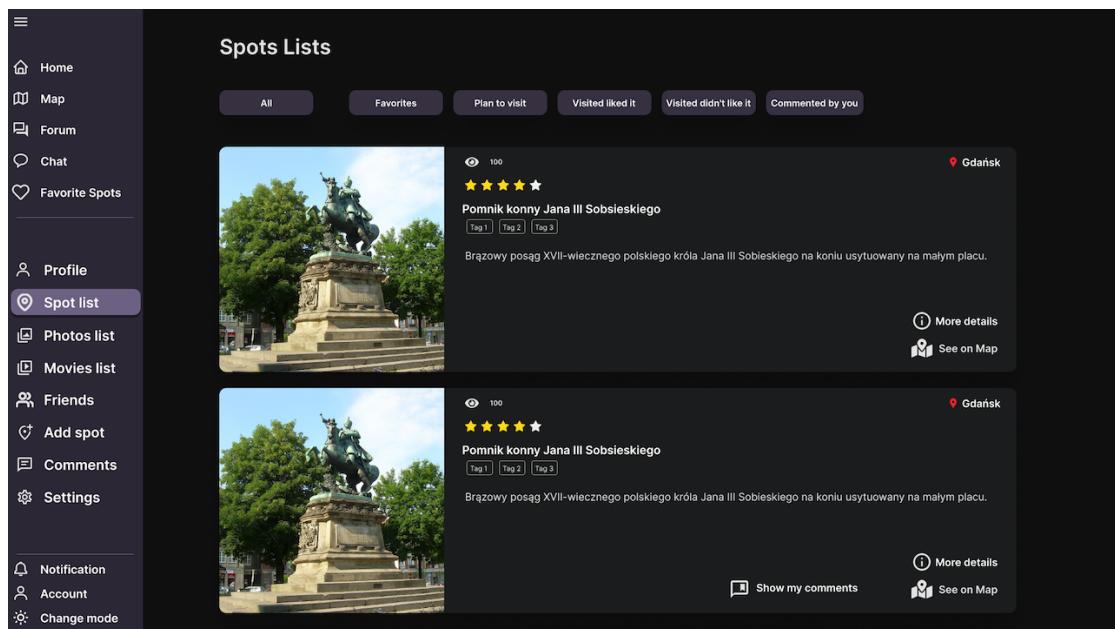
Lista [spotów](#) stanowi część panelu użytkownika, w której gromadzone są miejsca oznaczone przez użytkownika jako: ulubione, planowane do odwiedzenia, odwiedzone i ocenione pozytywnie, odwiedzone i ocenione negatywnie oraz skomentowane (rys. 5.16).

W górnej części widoku umieszczono nagłówek informujący, w jakiej sekcji panelu znajduje się użytkownik. Poniżej znajduje się zestaw sześciu przycisków służących do przełączania się między wymienionymi kategoriami list. Dzięki temu możliwe jest szybkie filtrowanie zawartości bez konieczności zmiany podstrony.

Niżej prezentowana jest lista [spotów](#) należących do aktualnie wybranej kategorii. Każdy kafelek reprezentujący miejsce zawiera następujące informacje:

- liczbę wyświetleń,
- średnią ocenę,
- nazwę [spota](#),

- przypisane tagi,
- krótki opis,
- nazwę miasta, w którym znajduje się **spot**,
- przycisk wyświetlenia szczegółów **spota**,
- przycisk wyświetlenia **spota** na mapie.



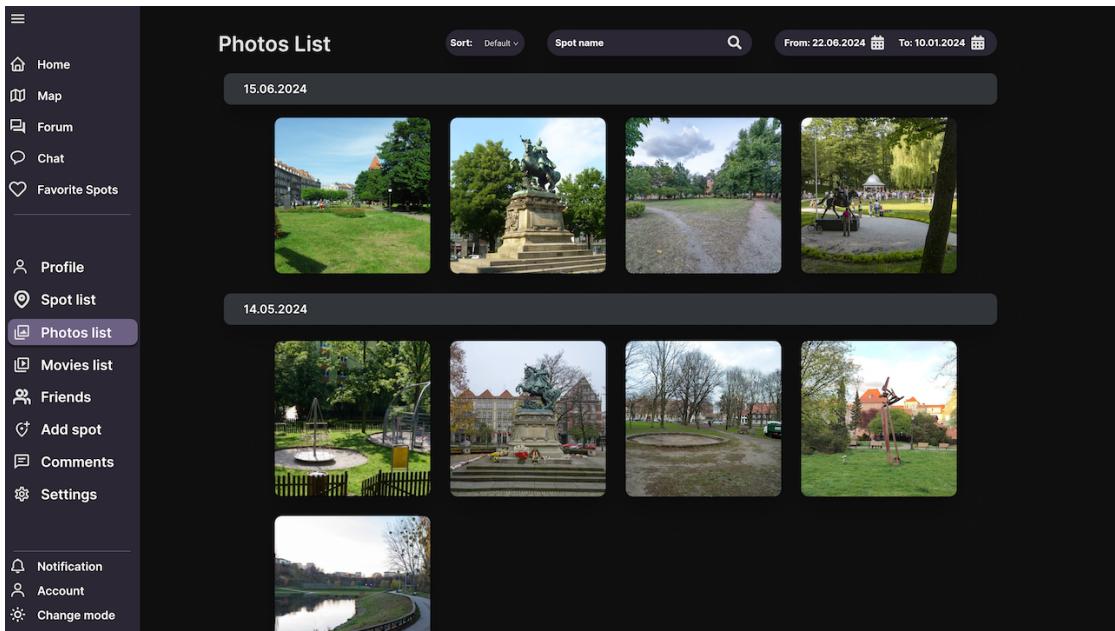
Rysunek 5.16: Projekt strony z listą ulubionych **spotów**

5.3.6.3 Photos list

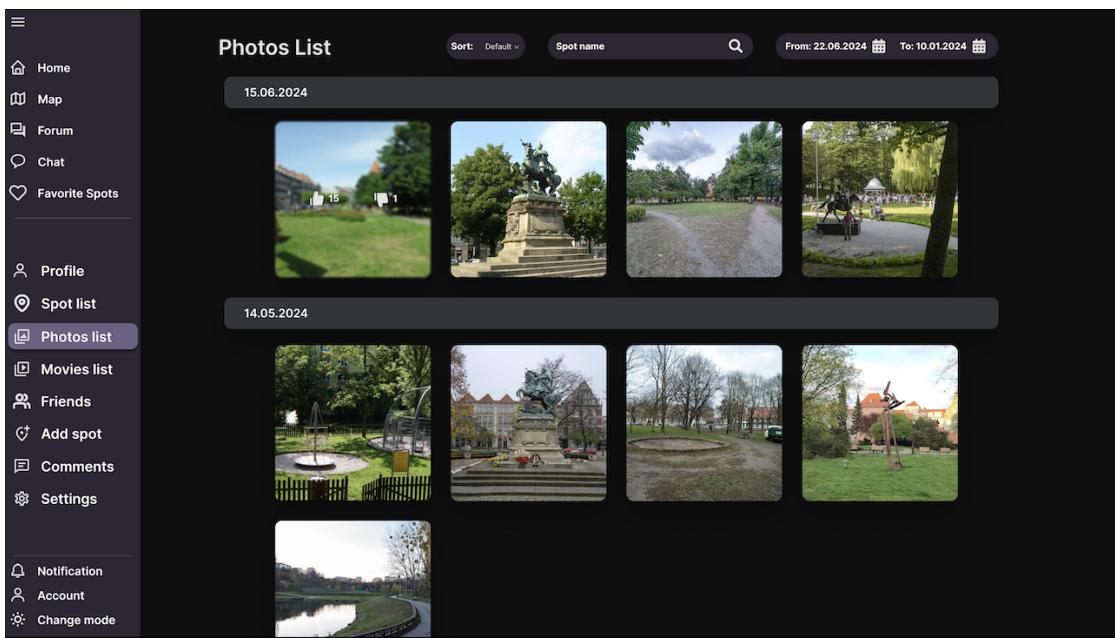
Kolejnym widokiem jest lista zdjęć (rys. 7.102), które zostały dodane do systemu. Podobnie jak w liście **spotów**, na górze strony znajduje się nagłówek informujący, w jakiej sekcji aktualnie znajduje się użytkownik. Tuż obok umieszczono panel wyszukiwania interesujących go zdjęć, na który składają się: panel sortowania, wyszukiwarka po nazwie **spota** oraz panel filtrowania po dacie dodania.

Poniżej wyświetlone są listy zdjęć zgrupowane według daty dodania (każda grupa poprzedzona jest paskiem z informacją o dacie dodania). W ramach grupy

prezentowane są miniatury zdjęć; po najechaniu kursorem na wybrane zdjęcie (rys. 5.18) pojawiają się informacje o liczbie polubień i niepolubień danego zdjęcia.



Rysunek 5.17: Projekt strony z listą dodanych zdjęć

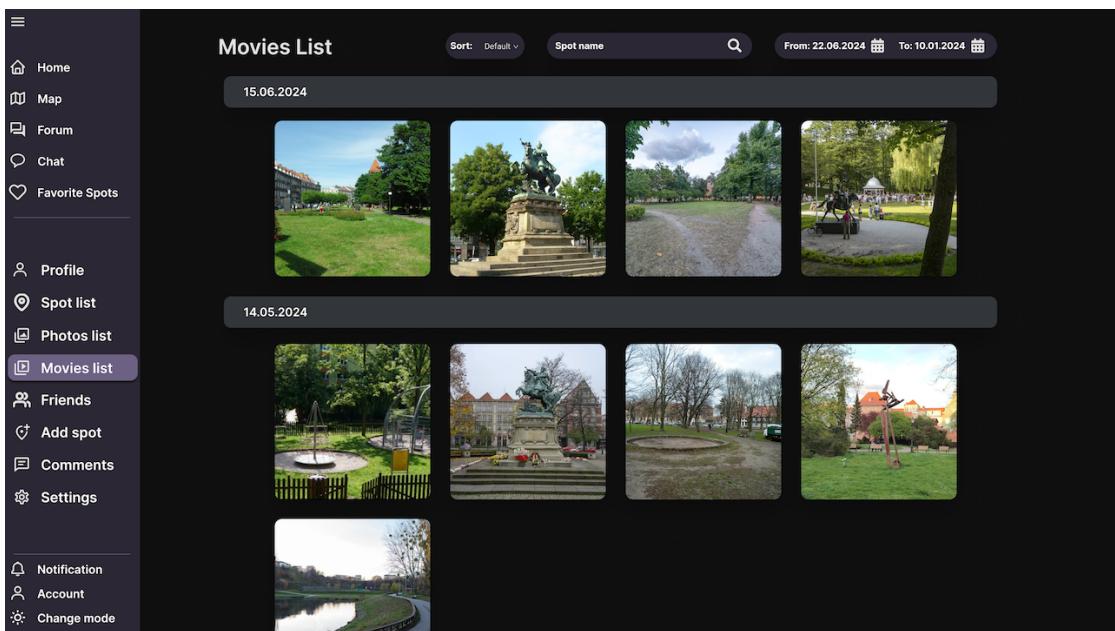


Rysunek 5.18: Projekt strony z listą dodanych zdjęć po najechaniu kursem

5.3.6.4 Movies list

Kolejnym widokiem jest lista filmów (rys. 7.104), które zostały dodane do systemu przez użytkownika. Podobnie jak w pozostałych widokach, w górnej części strony znajduje się nagłówek informujący, w jakiej sekcji panelu aktualnie znajduje się użytkownik. Zaraz obok umieszczono panel wyszukiwania filmów, obejmujący: panel sortowania, wyszukiwarkę po nazwie **spota** oraz filtry zakresu dat dodania materiału.

Niżej prezentowane są listy filmów zgrupowane według daty dodania (każda grupa poprzedzona jest paskiem z informacją o dacie). W obrębie grupy wyświetlane są miniatury filmów; po najechaniu kursem na wybraną miniaturę pojawiają się dodatkowe informacje, między innymi liczba polubień oraz liczba oznaczeń typu „nie lubię” danego filmu.



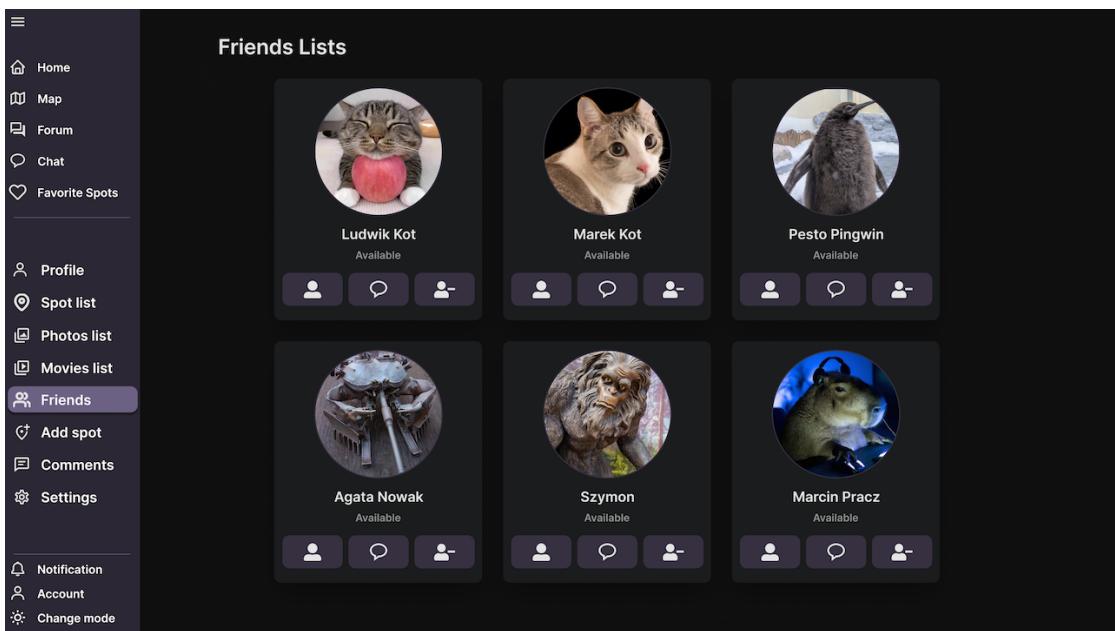
Rysunek 5.19: Projekt strony z listą dodanych filmów

5.3.6.5 Friends

Kolejnym widokiem jest lista znajomych użytkownika (rys. 5.20). W górnej części strony umieszczono nagłówek informujący, że wyświetlana jest sekcja listy znajomych. Poniżej prezentowane są kafelki z profilami osób dodanych do znajomych.

Każdy kafelek zawiera: zdjęcie profilowe użytkownika, jego nazwę oraz zestaw trzech przycisków:

- pierwszy służy do przejścia do profilu danego znajomego,
- drugi umożliwia otwarcie konwersacji z wybraną osobą,
- trzeci pozwala na usunięcie użytkownika z listy znajomych.



Rysunek 5.20: Projekt strony z listą znajomych

5.3.6.6 Add spot

Kolejnym widokiem jest strona służąca do dodawania [spotów](#) przez użytkownika (rys. 5.21). Podobnie jak w poprzednich sekcjach, w górnej części umieszczono nagłówek z nazwą sekcji. Dodatkowo w prawym górnym rogu znajduje się przycisk otwierający okno modalne z formularzem dodawania nowego [spota](#) (rys. 5.22). Poniżej wyświetlana jest lista [spotów](#), które zostały dodane przez użytkownika w aplikacji. Każdy kafelek zawiera:

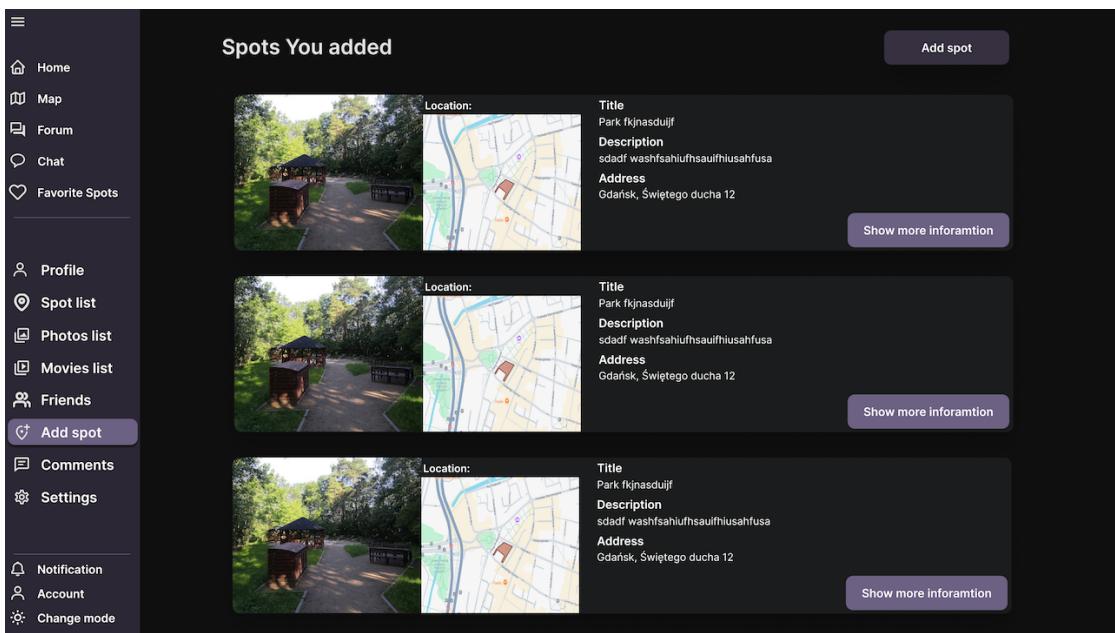
- pierwsze zdjęcie z danego miejsca,
- małą mapę z zaznaczoną lokalizacją [spota](#),
- nazwę [spota](#),
- krótki opis,
- adres (miasto i ulice),

- przycisk wyświetlenia szczegółowych informacji.

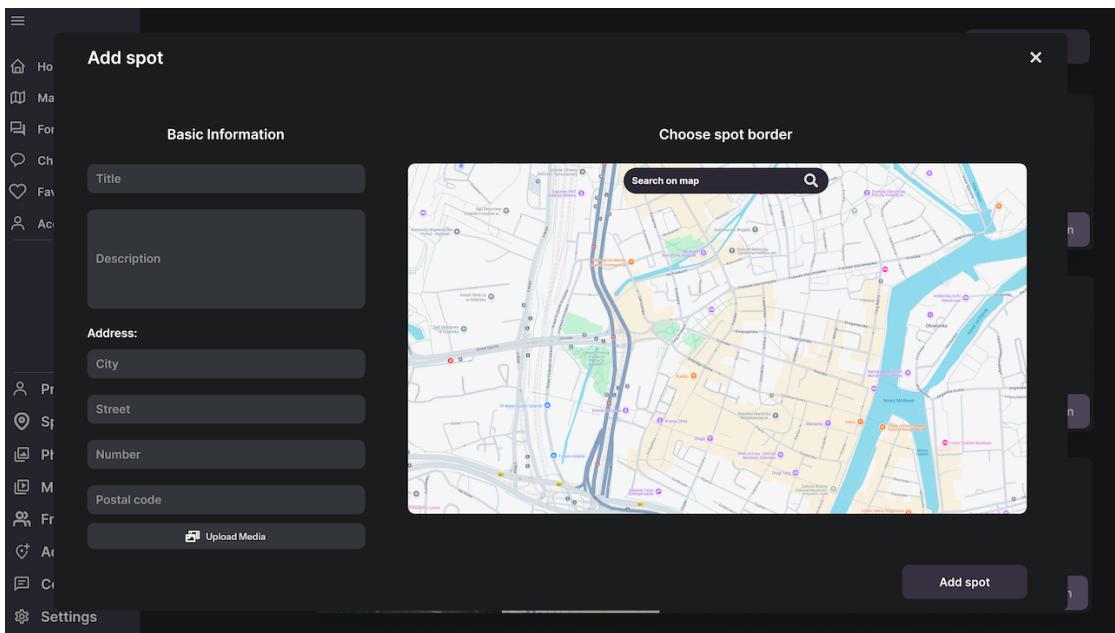
Formularz dodawania nowego **spota** umieszczono w oknie modalnym. Na jego górze znajduje się nagłówek określający wykonywaną akcję. Cały modal podzielono na dwie części:

- **Panel informacji tekstowych** – zawiera pola formularza do wprowadzania danych o **spocie**:
 - nazwa **spota**,
 - opis,
 - adres: miasto, ulica, numer budynku oraz kod pocztowy,
 - przycisk umożliwiający dodanie materiałów multimedialnych (zdjęć oraz filmów).
- **Panel mapy** – służy do określenia lokalizacji **spota** na mapie, z możliwością:
 - wyszukania miejsca po nazwie,
 - zaznaczenia obszaru będącego granicą **spota**.

Na dole okna/modalnego znajduje się przycisk potwierdzający dodanie nowego **spota** do systemu.



Rysunek 5.21: Projekt strony z listą dodanych spotów

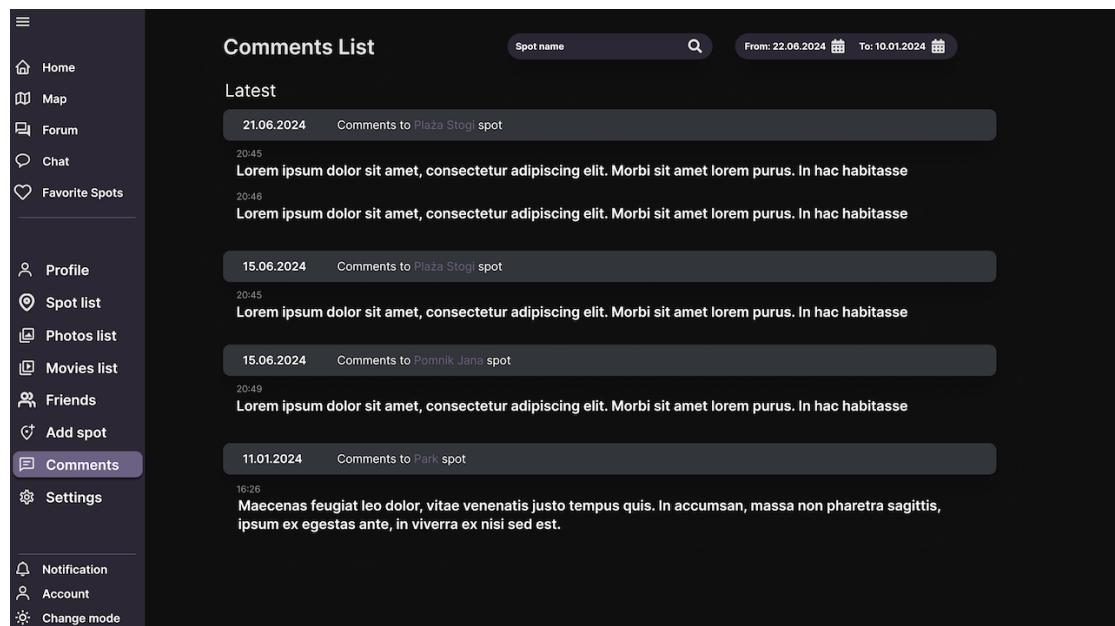


Rysunek 5.22: Projekt formularza do dodawania nowego spota

5.3.6.7 Comments

Podobnie jak w pozostałych widokach, w górnej części strony umieszczono nagłówek z nazwą sekcji (rys. 7.109). Po jego prawej stronie znajduje się panel wyszukiwania komentarzy, obejmujący pole tekstowe umożliwiające wyszukiwanie po nazwie **spota** oraz pola filtrowania zakresu dat (*od* oraz *do*).

Poniżej prezentowana jest lista komentarzy dodanych przez użytkownika. Elementy listy są grupowane według daty dodania oraz **spota**, którego dotyczy komentarz. Każda grupa poprzedzona jest paskiem informacyjnym zawierającym datę oraz nazwę **spota**, co ułatwia orientację w historii aktywności. W obrębie danej grupy wyświetlane są poszczególne komentarze razem z godziną dodania oraz treścią wpisu.



Rysunek 5.23: Projekt strony z listą dodanych komentarzy

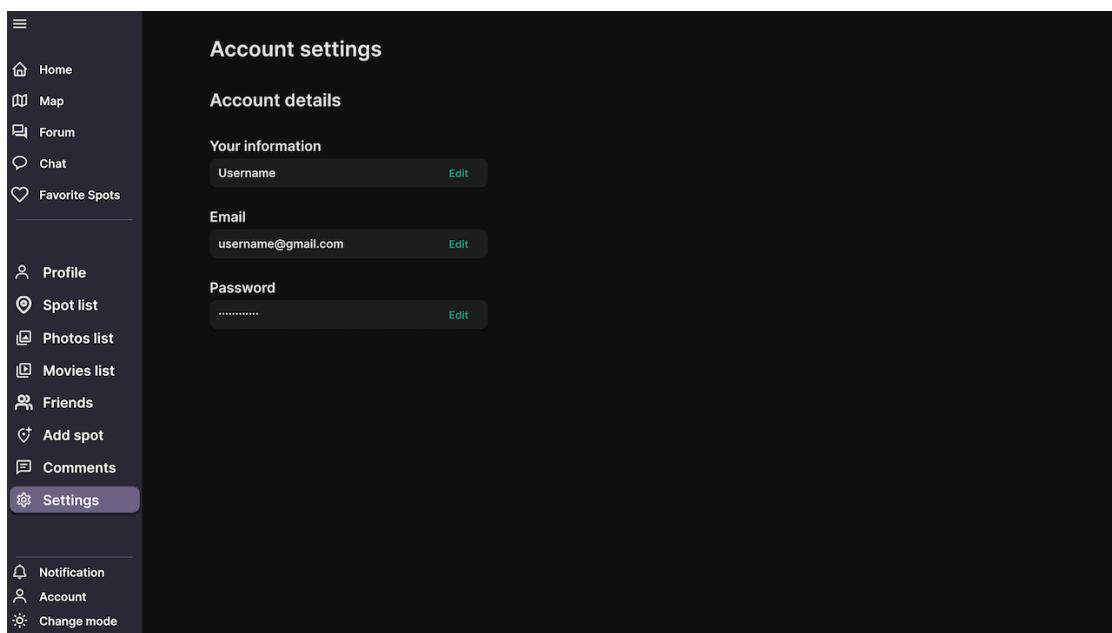
5.3.6.8 Settings

Ostatnią stroną panelu użytkownika są ustawienia (rys. 7.110). W górnej części widoku umieszczono nagłówek informujący, w jakiej sekcji aplikacji aktualnie

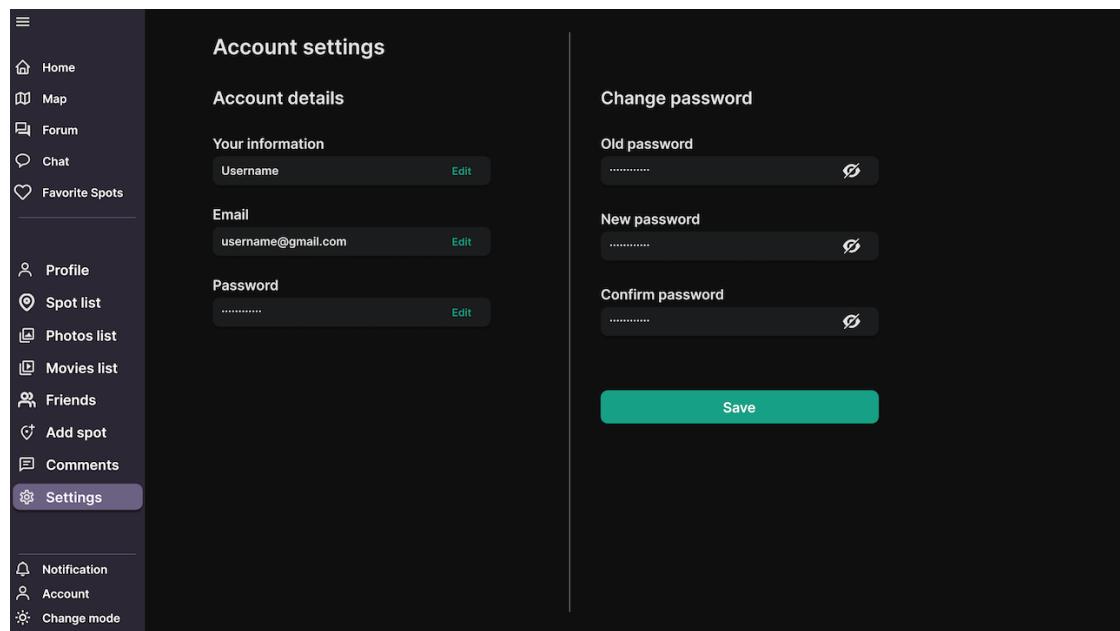
znajduje się użytkownik. Poniżej wyświetlane są trzy segmenty:

- informacje o użytkowniku (nazwa użytkownika),
- adres e-mail,
- hasło.

Każdy z segmentów wyposażono w przycisk *Edit*. Po jego kliknięciu po prawej stronie ekranu pojawia się druga część widoku (rys. 5.25), zawierająca formularz odpowiedni dla edytowanej sekcji oraz informację, który element konta jest aktualnie modyfikowany. W przypadku zmiany hasła formularz obejmuje pola na hasło bieżące, nowe hasło oraz jego potwierdzenie, przy czym wartości są domyślnie maskowane. Na dole prawej części strony umieszczono przycisk służący do zatwierdzenia wprowadzonych zmian.



Rysunek 5.24: Projekt strony ustawień (1)



Rysunek 5.25: Projekt strony ustawień (2)

Rozdział 6

Przebieg realizacji projektu

W niniejszym rozdziale przedstawiono przebieg realizacji projektu w ujęciu funkcjonalnym, z podziałem na główne moduły aplikacji. Opis koncentruje się na tym, co zostało zaimplementowane w ramach kolejnych obszarów systemu. Taki sposób ujęcia odpowiada charakterowi prac prowadzonych zgodnie z metodyką [Disciplined Agile Delivery - Lean Life Cycle](#), gdzie rozwój funkcjonalności odbywał się równolegle, a zakres modułów był stopniowo doprecyzowywany.

Rozwój był wspierany przez stałe [review kodu](#), automatyzację testów oraz rozwój infrastruktury deweloperskiej.

6.1 Elementy niezależne od modułu

W całym okresie realizacji rozwijano także elementy wspólne.

- **Stos technologiczny i struktura projektu** – doprecyzowanie architektury rozwiązania oraz organizacji repozytorium dla [backendu](#) i [frontenu](#).
- **Podstawy warstwy klienckiej** – stworzenie fundamentów aplikacji [fronthowej](#) ([routingu](#), konfiguracja stylów, struktury projektu), a także dodanie do projektu potrzebnych [bibliotek](#) m.in.: [TanStack Query](#), [Redux](#) oraz [Tailwind CSS](#).
- **Uwierzytelnianie i bezpieczeństwo** – implementacja logowania i rejestracji, doprecyzowanie ról i uprawnień w [Spring Security](#), a także logowanie za

pomocą [OAuth](#) poprzez Google oraz [GitHuba](#).

- **Odzyskiwanie dostępu do konta** – implementacja resetowania hasła.
- **Komunikacja e-mail** – integracja z zewnętrzną usługą wysyłania wiadomości mailowych.
- **Środowisko deweloperskie** – konteneryzacja serwisów, na które składa się aplikacja, z wykorzystaniem [Dockera](#). Przygotowanie skryptów ułatwiających uruchamianie środowiska deweloperskiego.
- **Jakość i automatyzacja** – rozwój testów ([integracyjnych](#), [jednostkowych](#) i [E2E](#)), uruchamianie testów w [CI/CD](#), ujednolicenie formatowania za pomocą [Prettier'a](#) i kontrola jakości kodu w repozytorium za pomocą [ESLint](#).
- **Wydajność** – wprowadzenie mechanizmu optymalizacji w postaci [cache'a](#).
- **Dane deweloperskie** – przygotowanie danych testowych wykorzystywanych do demonstracji i weryfikacji działania produktu.
- **Komunikaty systemowe** – implementacja mechanizmu prezentacji informacji o błędach i sukcesach.
- **Dokumentacja** – opracowanie dokumentacji projektu w [LaTeX'ie](#).

Przed rozpoczęciem implementacji modułów przygotowano projekt [interfejsu użytkownika \(UI\)](#), obejmujący kluczowe widoki. Stanowił on punkt odniesienia dla dalszych prac i ułatwił iteracyjne dopasowywanie funkcjonalności do sposobu użycia systemu.

6.2 Moduły aplikacji

6.2.1 Rozwój funkcjonalności w modułach

Poniżej przedstawiono rozwój funkcjonalności w kluczowych modułach aplikacji. Każdy moduł rozwijano iteracyjnie, stopniowo rozszerzając jego zakres.

6.2.1.1 Mapa

- **Prototyp mapy** – przygotowanie wstępnej wersji mapy (proof-of-concept) w celu weryfikacji możliwości interaktywnej prezentacji danych.
- **Migracja do rozwiązania docelowego** – przejście na React-MapLibre oraz ponowna implementacja podstaw wyświetlania **spotów**.
- **Model spota** – rozszerzanie modelu o dane lokalizacyjne oraz ujednolicenie sposobu wyznaczania punktu reprezentującego **spota**.
- **Widok szczegółów spota** – rozwój prezentacji informacji o **spocie**.
- **Komentarze i oceny** – dodanie możliwości dodawania komentarzy przez użytkownika.
- **Multimedia dla spotów** – ujednolicenie obsługi zdjęć i filmów oraz implementacja prezentacji multimediiów w widoku **spota**.
- **Galeria multimediiów** – wprowadzenie rozszerzonej galerii multimediiów dla danego **spota**.
- **Ulubione spoty** – możliwość dodawania **spotów** do ulubionych oraz integracja tej funkcji z panelem użytkownika.
- **Responsywność** – dostosowanie modułu mapy i widoku szczegółów **spotów** do różnych rozmiarów ekranu.
- **Podstawowa pogoda** – pobieranie danych pogodowych dla wybranego **spota**.
- **Rozszerzona pogoda** – wprowadzenie pogody szczegółowej.
- **Wyszukiwanie w obszarze mapy** – dodanie możliwości wyszukiwania **spotów** w aktualnie widocznym obszarze mapy.

6.2.1.2 Czat

- **Podstawy czatu** – implementacja listy czatów wraz z wyświetlaniem aktualnie otwartego.
- **Komunikacja w czasie rzeczywistym** – wdrożenie [WebSocket](#) do pobierania i wysyłania wiadomości.
- **Usprawnienia UX** – grupowanie wiadomości po dacie, wielowierszowe pole tekstowe, dodanie mechanizmu nieskończonego przewijania wiadomości w danym czacie.
- **Emoji i GIF** – dodanie obsługi [emoji](#) oraz wysyłania animowanych obrazów [GIF](#).
- **Wydajność i niezawodność** – optymistyczne wysyłanie wiadomości oraz implementacja stronicowania dla pobierania starszych wiadomości.
- **Rozmowy prywatne** – możliwość rozpoczęcia i kontynuowania rozmów bezpośrednio z list relacji społecznościowych w panelu użytkownika.
- **Czaty grupowe** – tworzenie czatów grupowych, edycja nazwy/obrazu oraz dodawanie kolejnych uczestników.
- **Obsługa załączników** – wysyłanie i wyświetlanie plików oraz obrazów.

6.2.1.3 Forum

- **Zarządzanie postami przez użytkownika** – edycja, usuwanie oraz dodawanie wraz z przypisywaniem kategorii i tagów.
- **Edytor treści** – integracja [edytora rich-text](#) na potrzeby tworzenia i edycji postów.
- **Multimedia w postach** – integracja z [Azure Blob Storage](#) do przesyłania i przechowywania mediów.

- **Przeglądanie, sortowanie oraz wyszukiwanie** – możliwość sortowania listy postów oraz wyszukiwania postów z wykorzystaniem filtrów.
- **Zarządzanie komentarzami przez użytkownika** – dodawanie, edycja oraz usuwanie postów przez użytkownika.
- **Głosowanie** – możliwość głosowania na posty oraz komentarze.
- **Zgłaszanie treści** – możliwość zgłaszania postów i komentarzy jako nieodpowiednich.
- **Obserwowanie postów** – możliwość obserwowania wybranych postów.
- **Udostępnianie postów** – możliwość udostępniania postów poprzez link.
- **Top 3 posty miesiąca** – prezentacja trzech najpopularniejszych postów z danego miesiąca.
- **Regulamin forum** – udostępnienie regulaminu forum.

6.2.1.4 Wyszukiwarka spotów

- **Wyszukiwanie spotów** – wdrożenie wyszukiwania po nazwie z panelem wyników.
- **Wyszukiwanie po lokalizacji** – dodanie wyszukiwania po lokalizacji wraz z listą wyników i dystansem od użytkownika.
- **Najpopularniejsze spotty** – dodanie karuzeli z najpopularniejszymi [spotty](#).
- **Funkcje zaawansowane** – [Paginacja](#) wyników, sortowanie (po popularność i ocenie) oraz filtrowanie po minimalnej ocenie.

6.2.1.5 Panel użytkownika

- **Profil użytkownika** – wdrożenie profilu oraz rozdzielenie widoku własnego profilu i profilu innego użytkownika.

- **Relacje społecznościowe** – obsługa znajomych, obserwowanych i obserwujących, wraz z nawigacją oraz dedykowanymi widokami list.
- **Zaproszenia do znajomych** – statusy relacji (wysłane/otrzymane/zakończone), widok listy zaproszeń oraz akcje akceptowania i odrzucania.
- **Wyszukiwanie użytkowników** – funkcja dodawania znajomych z paginacją i wyszukiwaniem po nazwie użytkownika.
- **Aktywność** – sekcje zdjęć i filmów, widok dodanych komentarzy do spota.
- **Ustawienia konta** – edycja danych konta (nazwa użytkownika, e-mail, hasło) oraz obsługa zmiany zdjęcia profilowego.
- **Moje spoty i ulubione** – lista dodanych [spotów](#) oraz zarządzanie ulubionymi [spotami](#) z integracją z mapą.

6.3 Podsumowanie etapu finalizacji

Końcowy okres realizacji koncentrował się na finalizacji implementacji funkcjonalności oraz uzupełnieniu dokumentacji technicznej i tekstuowej. Przyjętą datą zakończenia prac projektowych jest **10 stycznia 2026 roku**.

Rozdział 7

Realizacja Projektu

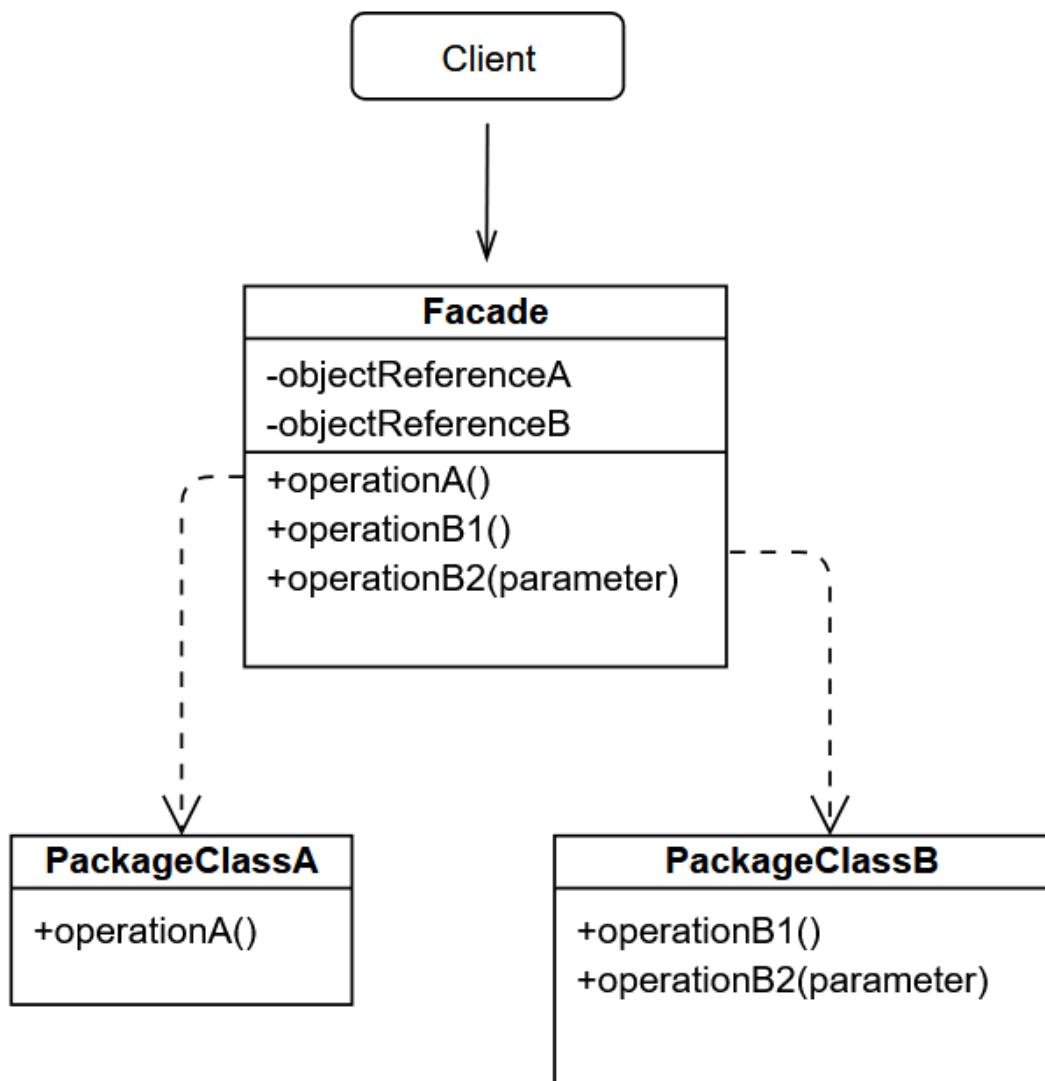
7.1 Wzorce projektowe

Podczas prac deweloperskich nad projektem skorzystano z różnych wzorców projektowych. Zarówno na frontendzie, jak i backendzie istnieje wiele propozycji, z których członkowie zespołu starali się korzystać w celu poszerzenia wiedzy, umiejętności oraz otrzymania wysokiej jakości pisanego kodu. Poniżej przedstawiono wybrane rozwiązania.

7.1.1 Backend

7.1.1.1 Fasada

Jest to strukturalny wzorzec projektowy, który nakłada na bibliotekę lub zestaw klas interfejs ułatwiający korzystanie z zawartych w nich operacji.



Rysunek 7.1: Diagram klas wzorca projektowego Fasada

W projekcie Fasada została zastosowana zaimplementowana jako *PolygonAreaCalculator*. To klasa odpowiedzialna za obliczanie pola powierzchni [spota](#) na podstawie ograniczających go punktów. Do wykonywania konkretnych obliczeń wykorzystano [bibliotekę](#) *geographiclib*, której komponenty są używane podczas wywołania metody *calculateArea*. Dzięki zastosowaniu Fasady, gdy zajdzie potrzeba

zmiany biblioteki, ponownej implementacji będzie wymagać tylko metoda *calculateArea* – sposób jej wywoływania pozostanie bez zmian.

Poniżej przedstawiono implementację klasy *PolygonAreaCalculator* oraz jej przykładowe użycie podczas operacji dodawania nowego *spota*.

```
import com.merkury.vulcanus.model.embeddable.BorderPoint;
import net.sf.geographiclib.Geodesic;
import net.sf.geographiclib.PolygonArea;
import net.sf.geographiclib.PolygonResult;

public class PolygonAreaCalculator { 4 usages  ↳ stanoz

    Params: points – must be added in clockwise or counterclockwise orders
    Returns: area in square meters

    public static double calculateArea(BorderPoint[] points) { 2 usages  ↳ stanoz
        int n = points.length;
        PolygonArea pa = new PolygonArea(Geodesic.WGS84, polyline:false);
        for (int i = 0; i < n - 1; i++) {
            BorderPoint point = points[i];
            pa.AddPoint(point.getX(), point.getY());
        }
        PolygonResult res = pa.Compute( reverse false, sign:true );
        return Math.abs(res.area);
    }
}
```

Rysunek 7.2: Implementacja wzorca Fasada

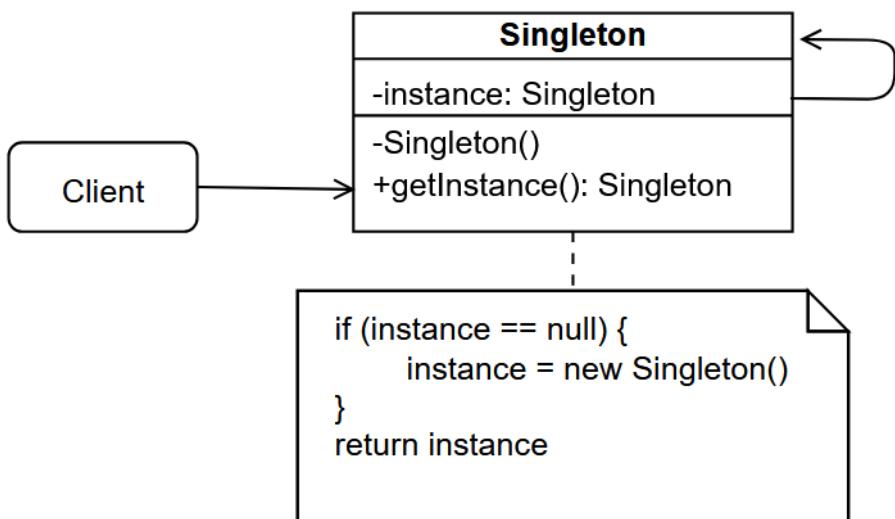
```
var area = PolygonAreaCalculator.calculateArea(spot.borderPoints().toArray(new BorderPoint[0]));
```

Rysunek 7.3: Przykładowe użycie klasy *PolygonAreaCalculator*

7.1.1.2 Singleton

Kreacyjny wzorzec projektowy zapewniający istnienie dokładnie jednej instancji danego obiektu, która jest dostępna globalnie. Konstruktor takiego obiektu jest

prywatny, a dostęp do niego odbywa się poprzez statyczną metodę zwracającą istniejącą instancję lub jeśli jej nie ma, tworzącą nową. Używany jest między innymi do zarządzania konfiguracjami czy połączeniami do bazy danych. Wzorzec Singleton łamie zasadę [Single Responsibility](#), ponieważ taki obiekt oprócz wykonywania swojej logiki, dba o swoją unikatowość.



Rysunek 7.4: Diagram klas wzorca projektowego Singleton

W projekcie skorzystano z [frameworka](#) Spring Boot, w którym wszystkie klasy oznaczone jako [Beany](#) są Singletonami. Przykładem jest nadanie klasie [adnotacji](#) `@Service`. Raz stworzony obiekt jest wykorzystywany przez wszystkie inne potrzebujące go obiekty.

Poniżej przedstawiono wykorzystanie [adnotacji](#) `@Service` oraz użycie tej klasy jako zależność w innej klasie.

```
@RequiredArgsConstructor 3 usages  ⚙ stanoz +2
@Service
public class UserDataService {
    private final OAuth2AuthorizedClientService oAuth2AuthorizedClientService;
    private final WebClient webClient;
    private finalUrlsProperties urlsProperties;
    private final UserRepository userRepository;
    private final JwtManager jwtManager;
```

Rysunek 7.5: Adnotacja frameworka Spring Boot tworząca Singleton

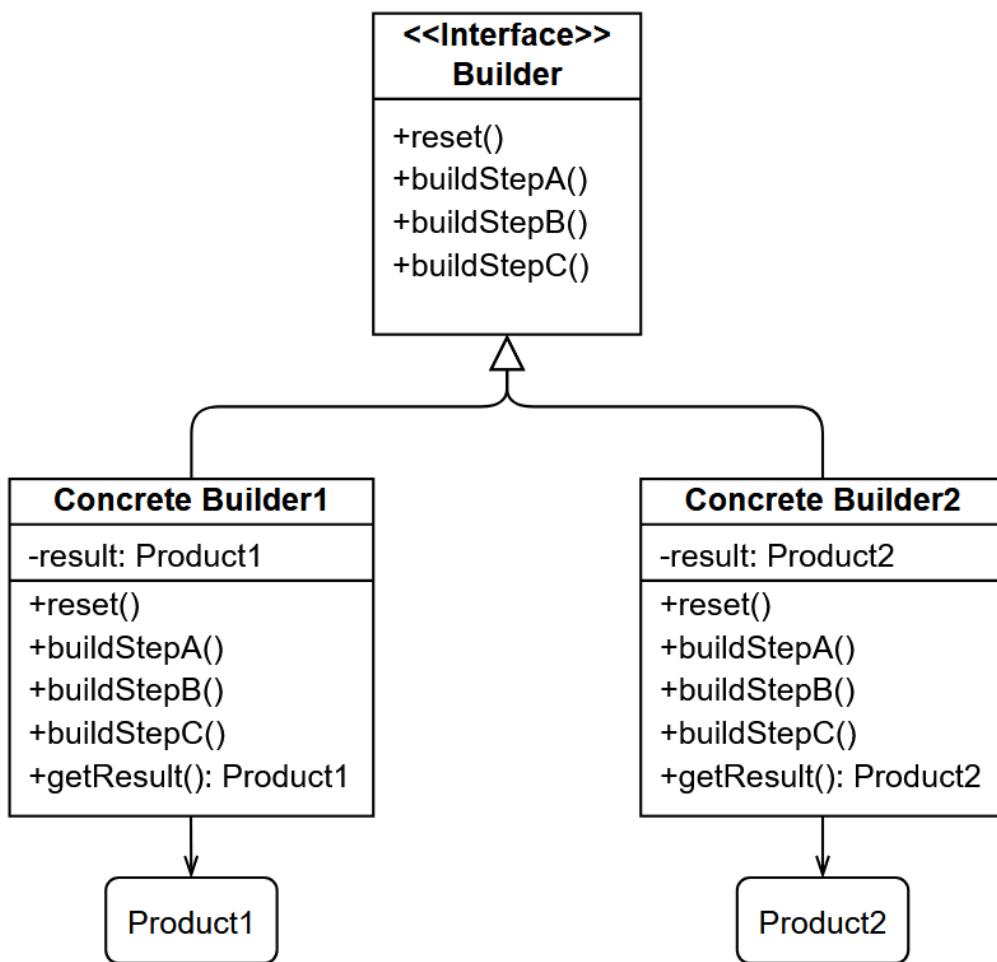
```
@Service 2 usages  ⚙ stanoz +2
@RequiredArgsConstructor
public class SpotCommentService {

    private final SpotCommentRepository spotCommentRepository;
    private final SpotRepository spotRepository;
    private final UserDataService userDataService;
```

Rysunek 7.6: Użycie Singletona jako zależności

7.1.1.3 Builder

Kreacyjny wzorzec projektowy, który ułatwia tworzenie skomplikowanych obiektów poprzez rozbicie tego procesu na mniejsze, konfigurowalne etapy. Eliminuje potrzebę korzystania z konstruktorów zawierających wiele parametrów. Stworzony zostaje obiekt budujący (Budowniczy), który implementuje poszczególne kroki konstrukcji obiektu, a na końcu wywoływana jest metoda inicjalizująca go. Nie jest wymagane wywołanie wszystkich kroków, ponadto można stworzyć wielu Budowniczych, kreujących różne warianty obiektu.



Rysunek 7.7: Diagram klas wzorca projektowego Builder

Do zaimplementowania tego wzorca projektowego wykorzystano adnotację `@Builder` z biblioteki *Lombok*, która powoduje utworzenie Budowniczego dla danej klasy. Builder został zastosowany między innymi w klasach reprezentujących encje, co poprawiło czytelność kodu przy ich tworzeniu.

Poniżej przedstawiono zastosowanie adnotacji `@Builder` oraz przykładowe użycie tego wzorca podczas konstruowania encji.

```
@Entity(name = "spots_tags")  ↳ stanoz
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public class SpotTag {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @Builder.Default
    @ManyToMany(mappedBy = "tags")
    private Set<Spot> spots = new HashSet<>();
}
```

Rysunek 7.8: Implementacja wzorca projektowego Builder

```
public class SpotTagMapper { 5 usages  ↳ stanoz
    private SpotTagMapper() {}  no usages  ↳ stanoz

    public static SpotTagDto toDto(@NotNull SpotTag spotTag) {  ↳ stanoz
        return SpotTagDto.builder()
            .id(spotTag.getId())
            .name(spotTag.getName())
            .build();
    }
}
```

Rysunek 7.9: Przykładowe użycie wzorca projektowego Builder

7.1.1.4 Chain of Responsibility

Opisy wzorców projektowych użytych na backendzie zostały wykonane na podstawie treści zawartych w książce [14].

7.1.2 Frontend

7.1.2.1 Hooks Pattern

Wzorzec projektowy, który umożliwia korzystanie z mechanizmów Reacta takich jak stan czy cykl życia komponentu w komponentach funkcyjnych. Przed wprowadzeniem hook'ów do zarządzania tymi właściwościami trzeba było tworzyć komponenty klasowe, co powodowało większe skomplikowanie kodu. Hooks Pattern pozwala również na tworzenie własnych hook'ów zawierających logikę używaną w wielu komponentach.

W projekcie wykorzystano ten wzorzec do zarządzania stanem oraz efektami ubocznymi w komponentach za pomocą wbudowanych hook'ów `useState` i `useEffect`.

```
const [searchedSpots, setSearchedSpots] = useState<HomePageSpotDto[]>([]);
```

Rysunek 7.10: Przykładowe użycie hook'a useState

```
useEffect(() : void => {
  if (isSuccess) {
    dispatch(accountAction.setIsLogged());
    dispatch(accountAction.setUsername(enteredValue.username));
  }
}, [isSuccess, dispatch, enteredValue.username]);
```

Rysunek 7.11: Przykładowe użycie hook'a useEffect

Zaimplementowano też własne hooki. Jednym z nich jest `useBoolean` obslugujący logikę do zarządzania zmiennymi przyjmującymi wartości `true` lub `false`. Udostęp-

niane są funkcje do ustawienia wartości, a także do zmiany jej na przeciwną do poprzedniej. Korzystanie z tego **hook'a** pozwoliło uniknąć powtarzalności kodu oraz poprawiło jego czytelność. Poniżej przedstawiono implementację oraz przykładowe użycie **hooka** *useBoolean*.

```
import { useCallback, useState } from "react";

export function useBoolean(initialValue = false) :readonly [boolean, () => void, () => void... { Show usages & Mredosz
  const [value, setValue] = useState(initialValue);

  const setTrue :() => void = useCallback(() :void => setValue( value: true), []);
  const setFalse :() => void = useCallback(() :void => setValue( value: false), []);
  const toggle :() => void = useCallback(() :void => setValue((prev :boolean) :boolean => !prev), []);

  return [value, setTrue, setFalse, toggle] as const;
}
```

Rysunek 7.12: Implementacja hook'a useBoolean

```
const [areHintsShown, showHints, hideHints] = useBoolean();
```

Rysunek 7.13: Przykładowe użycie hook'a useBoolean

7.1.2.2 Optimistic UI

Wzorzec, w którym **UI** jest aktualizowane natychmiast po tym gdy użytkownik wykona czynność, przy założeniu że ta akcja po przetworzeniu przez **backend** zostanie zakończona sukcesem. Jeżeli operacja nie powiedzie się, użytkownik zostanie o tym poinformowany, a stan **UI** wróci do poprzedniego. Dzięki zastosowaniu tego **wzorca** zmniejszone jest odczucie opóźnień w działaniu aplikacji oraz poprawa w płynnym jej użytkowaniu. W projekcie Optimistic **UI** zostało wykorzystane w module Chatu. Gdy użytkownik wyśle nową wiadomość, natychmiast jest ona wyświetlana w oknie konwersacji. Poniżej przedstawiono implementację tego **wzorca** w **komponentach** *ChatBottomBar* oraz *ChatMessagingWindow*.

```
if (text) {
    const chatId : number = selectedChatId;

    const optimisticUUID : string =
        typeof crypto !== "undefined" && "randomUUID" in crypto
            ? crypto.randomUUID()
            : `${Date.now()}-${Math.random().toString(16).slice(2)}`;

    const optimistic: LocalMsg = {
        id: -Date.now(),
        chatId,
        content: text,
        sentAt: new Date().toISOString(),
        sender: { id: -1, name: username || "You", imgUrl: "" },
        optimistic: true,
        optimisticUUID,
    };

    dispatch(
        chatActions.setLastMessage({ chatId, message: optimistic }),
    );

    queryClient.setQueryData<InfiniteData<ChatMessagesPageDto>>(
        ["messages", chatId],
        (old : InfiniteData<ChatMessagesPageDto, unknown...>) : { pages: ChatMessagesPageDto[]; pageParam... } => {
            if (!old) return old;
            const first : ChatMessagesPageDto = old.pages[0] ?? {
                messages: [],
                hasNextSlice: true,
                numberOfMessages: 0,
                sliceNumber: 0,
            };
        });
}
```

Rysunek 7.14: Implementacja komponentu ChatBottomBar (1)

```

    const newFirst = {
      ...first,
      messages: [
        optimistic as ChatMessageDto,
        ...(first.messages ?? []),
      ],
    };
    return { ...old, pages: [newFirst, ...old.pages.slice( start:1)] };
  },
);

queryClient.setQueriesData<InfiniteData<ChatPage>>(
  { queryKey: ["user-chat-list"] },
  (old : InfiniteData<ChatPage, unknown> | undefined ) :{ pages: { items: ChatDto[]}; nextPage?: number } => {
    if (!old) return old;
    const pages:{ items: ChatDto[]; nextPage?: number | undefined } = old.pages.map((p : ChatPage ) :{ items: ChatDto[]; nextPage?: number | undefined } => ({
      ...p,
      items: p.items.map((c : ChatDto ) : ChatDto =>
        c.id === chatId
          ? { ...c, lastMessage: optimistic }
          : c,
      ),
    }));
    return { ...old, pages };
  },
);

```

Rysunek 7.15: Implementacja komponentu ChatBottomBar (2)

```
const payload: ChatMessageToSendDto & {  
    optimisticMessageUUID: string;  
} = {  
    chatId,  
    content, text,  
    sentAt: optimistic.sentAt,  
    optimisticMessageUUID: optimisticUUID,  
};  
  
setIsSending( value: true);  
try {  
    publish( destination: `/app/send/${chatId}/message` , payload);  
    setMessageToSend( value: "");  
} finally {  
    setIsSending( value: false);  
}  
}
```

Rysunek 7.16: Implementacja komponentu ChatBottomBar (3)

Komponent ten odpowiedzialny jest za przygotowanie wiadomości, przesłanie jej do wyświetlenia komponentowi *ChatMesseginingWindow* oraz równoczesne wysłanie danych na odpowiedni endpoint do backendu.

```

useEffect(() :void => {
  if (!chatDto?.id || !lastIncoming) return;

  if (String(lastIncoming.chatId) !== String(chatDto.id)) return;

  if (lastIncoming.id === lastAppliedIdRef.current) return;

  queryClient.setQueryData<InfiniteData<MessagesSlice>>(
    ["messages", lastIncoming.chatId],
    (old : InfiniteData<MessagesSlice, unknown> | un... ) : InfiniteData<MessagesSlice, unknown> | un... => {
      if (!old) return old;

      const hasSameId :boolean = old.pages.some((p : MessagesSlice ) : boolean =>
        (p.messages ?? []).some((m : ChatMessageDto ) : boolean => m.id === lastIncoming.id),
      );
      if (hasSameId) {
        lastAppliedIdRef.current = lastIncoming.id;
        return old;
      }

      let replaced :boolean = false;
      const pagesReplaced :MessagesSlice[] = old.pages.map((p : MessagesSlice ) : MessagesSlice => {
        if (replaced) return p;
        const msgs :ChatMessageDto[] = (p.messages ?? []).map((m : ChatMessageDto ) : ChatMessageDto => {
          const isOptimistic :boolean =
            (m as any).optimistic === true ||
            (typeof m.id === "number" && m.id < 0);

          const sameAuthorName :boolean =
            (m as any).sender?.name ===
            (lastIncoming as any).sender?.name;
          const sameContent :boolean =
            (m as any).content ===
            (lastIncoming as any).content;
        });
      });
    });
  );
}

```

Rysunek 7.17: Implementacja komponentu ChatMessagingWindow (1)

```

    const timeClose : boolean =  

      Math.abs(  

        new Date(m.sentAt).getTime() -  

        new Date(lastIncoming.sentAt).getTime(),  

      ) < 5000;  
  

    if (  

      !replaced &&  

      isOptimistic &&  

      sameAuthorName &&  

      sameContent &&  

      timeClose  

    ) {  

      replaced.= true;  

      return lastIncoming;  

    }  

    return m;  

});  

return { ...p, messages: msgs };  

});  
  

if (replaced) {  

  lastAppliedIdRef.current = lastIncoming.id;  

  return { ...old, pages: pagesReplaced };  

}  
  

const first :MessagesSlice = old.pages[0] ?? {  

  messages: [],  

  hasNextSlice: true,  

  sliceNumber: 0,  

};
```

Rysunek 7.18: Implementacja komponentu ChatMessagingWindow (2)

```
        const first : MessagesSlice = old.pages[0] ?? {
          messages: [],
          hasNextSlice: true,
          sliceNumber: 0,
        };
        const newFirst = {
          ...first,
          messages: [lastIncoming, ...(first.messages ?? [])],
        };
        lastAppliedIdRef.current = lastIncoming.id;
        return { ...old, pages: [newFirst, ...old.pages.slice(start: 1)] };
      },
    );
}, [chatDto?.id, lastIncoming, queryClient]);
```

Rysunek 7.19: Implementacja komponentu ChatMessagingWindow (3)

```

return (
  <div
    ref={containerRef}
    className="dark:scrollbar-track-violetDark scrollbar-thumb-violetLight scrollbar-thumb-rounded-full
    scrollbar-thin dark:bg-violetDark/20 flex h-full flex-col-reverse overflow-y-scroll bg-gray-50 py-1"
  >
    {messages.map((message: ChatMessageDto, idx: number) : Element => {
      const thisDate: string = new Date(message.sentAt).toDateString();
      const prevMessage: ChatMessageDto = messages[idx + 1];
      const prevDate: string =
        prevMessage && new Date(prevMessage.sentAt).toDateString();

      const shouldGroupMessagesByTime: boolean =
        checkIfShouldGroupMessagesByTime(message, prevMessage);

      return (
        <div
          key={`${chatDto.id}:${message.id}`}
          className="hover:bg-violetLight/10 pl-2"
        >
          {thisDate !== prevDate && (
            <div className="my-2 flex w-full items-center">
              <hr className="flex-grow border-gray-500" />
              <span className="px-2 text-xs dark:text-gray-300">
                {format(new Date(message.sentAt), formatStr: "PPP")}
              </span>
              <hr className="flex-grow border-gray-500" />
            </div>
          )}
          <ChatMessage
            message={message}
            shouldGroupMessagesByTime={
              shouldGroupMessagesByTime
            }
          />
        </div>
      );
    });
  );
}

```

Rysunek 7.20: Implementacja komponentu ChatMessagingWindow (4)

W tym **komponencie** sprawdzane jest czy wiadomość pochodzi z optymistycznego przebiegu realizacji oraz jest wyświetlana w oknie konwersacji.

7.1.2.3 Protected route

Wzorzec polegający na uniemożliwieniu dostępu do stron lub podstron aplikacji nieuwierzytelnionym użytkownikom. W projekcie do realizacji tego wzorca wykorzystano **komponent** *ProtectedRoute* oraz bibliotekę *react-router-dom*. Gdy użytkownik będzie próbował przejść do sekcji wymagającej wcześniejszego zalogowania

bez uczynienia tego, zostanie automatycznie przekierowany na stronę główną aplikacji.

```
export default function ProtectedRoute({ children }) { Show usages ⌘Mredosz
  const isLoggedIn = useSelector(selector: (state: StateType) => state.account.isLoggedIn);

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

Rysunek 7.21: Implementacja komponentu ProtectedRoute

```
path: "account",
children: [
  {
    index: true,
    element: <Navigate to="profile" replace />,
  },
  {
    path: "profile",
    element: (
      <ProtectedRoute>
        <UserOwnProfile />
      </ProtectedRoute>
    ),
  },
  {
    path: "profile/:username",
    element: <ProfileForViewer />,
  },
]
```

Rysunek 7.22: Przykładowe użycie komponentu ProtectedRoute

7.1.2.4 Portal

To renderowanie komponentów w innym miejscu drzewa DOM niż wynika z ich hierarchii ułożenia. Mimo takiego ustawienia, propagacja zdarzeń przebiega w sposób niezmieniony, tzn. jest obsługiwana przez komponent nadzędny. Stosowane gdy komponent musi być wyświetlony nad innymi elementami UI, aby uniknąć ograniczeń stylów rodzica. W projekcie ten wzorzec zastosowano poprzez komponent *Modal*, którego implementację oraz przykładowe użycie przedstawiono poniżej.

```
interface ModalProps { Show usages  ↳ Mredosz
  onClose: () => void;
  children: ReactNode;
  onClick: () => void;
  isOpen: boolean;
}

export default function Modal({ Show usages  ↳ Mredosz
  onClose,
  children,
  onClick,
  isOpen,
}: ModalProps): ReactPortal | null {
  const modalRoot: HTMLElement | null = document.getElementById(elementId: "modal");
  if (!modalRoot) {
    return null;
  }

  return createPortal(
    <AnimatePresence initial={false}>
      {isOpen} && (
        <>
          <motion.div
            className="fixed inset-0 z-62 bg-black/70"
            onClick={onClose}
            initial={{ opacity: 0 }}
            animate={{ opacity: 1 }}
            exit={{ opacity: 0 }}
          ></motion.div>
      )
    </AnimatePresence>
  );
}
```

Rysunek 7.23: Implementacja komponentu Modal (1)

```
<motion.div
  className="■dark:bg-darkBgSoft ■dark:text-darkText □bg-lightBgSoft ■text-lightText
  fixed top-1/2 left-1/2 z-63 w-11/12 max-w-md -translate-x-1/2 -translate-y-1/2 rounded-md
  p-8 shadow-md"
  onClick={({event : MouseEvent<HTMLDivElement, MouseEvent>} : void => event.stopPropagation())
  initial={{ opacity: 0, y: 50 }}
  animate={{ opacity: 1, y: 0 }}
  exit={{ opacity: 0, y: -50 }}
  transition={{ duration: 0.3 }}>
  {children}
  <form method="dialog" className="mt-3 flex space-x-3">
    <Button
      variant={ButtonVariantType.MODAL}
      onClick={onClose}
      className="■bg-red-600 ■hover:bg-red-700">
      no
    </Button>
    <Button
      variant={ButtonVariantType.MODAL}
      onClick={onClick}
      className="■bg-green-600 ■hover:bg-green-700">
      yes
    </Button>
  </form>
</motion.div>
</>
)
)
}
</AnimatePresence>,
modalRoot,
);
}
```

Rysunek 7.24: Implementacja komponentu Modal (2)

```

        <Button
          variant={ButtonVariantType.PROFILE}
          onClick={getFriendActionHandler(data.friendStatus)}
        >
          {statusToMessage[data.friendStatus] ?? "unknown status"}
        </Button>
      </div>
    </Profile>
    <Modal
      onClose={closeModal}
      onClick={handleConfirm}
      isOpen={isModalOpen}
    >
      <h2 className="text-xl text-shadow-md">
        {modalAction === SocialListType.FRIENDS
          ? `Are you sure you want to remove ${username} as a friend?`
          : `Are you sure you want to unfollow ${username}?`}
      </h2>
    </Modal>
  </>
);
}

```

Rysunek 7.25: Przykładowe użycie komponentu Modal

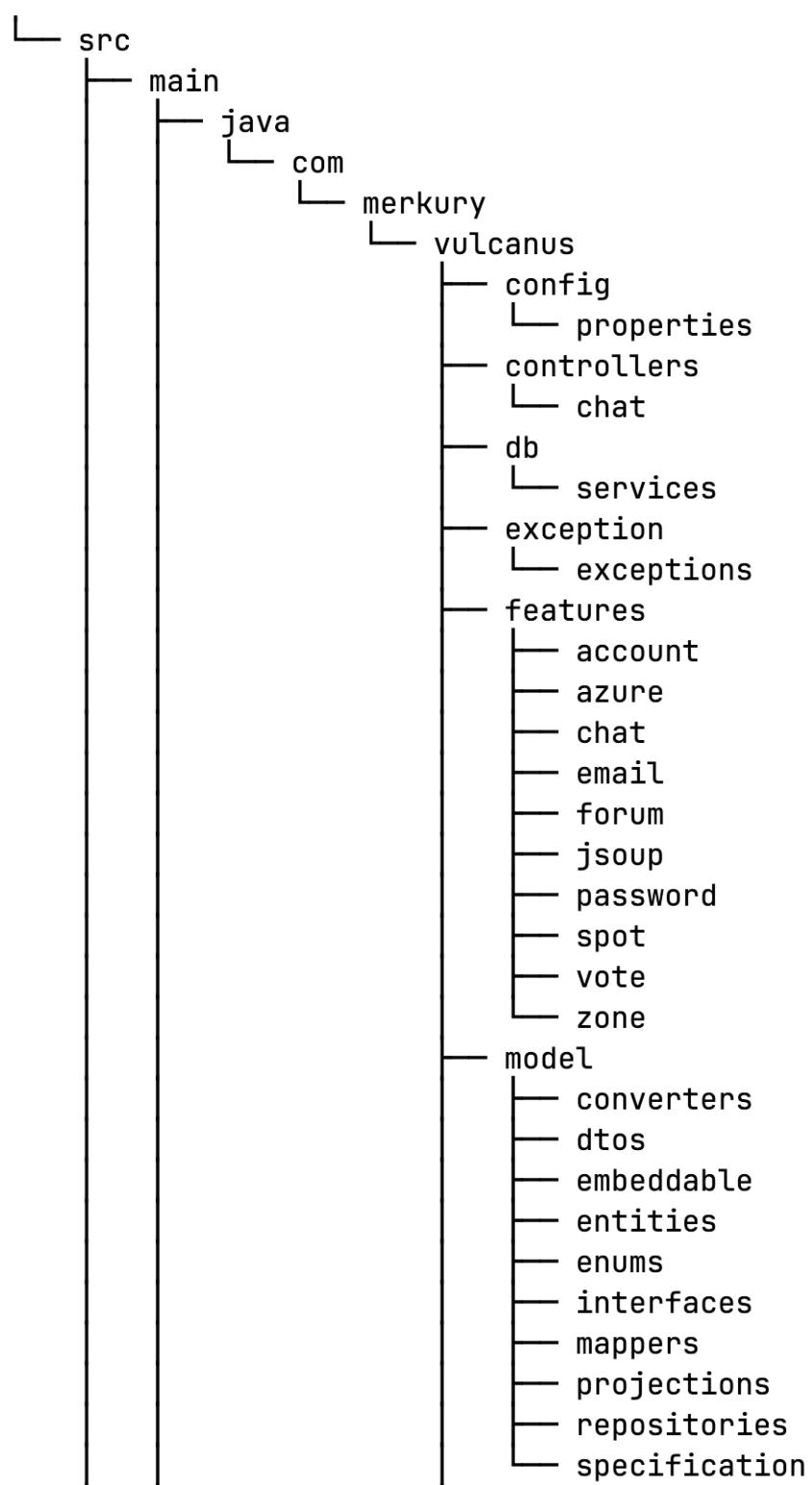
7.2 Implementacja backendu

W niniejszym rozdziale przedstawiono strukturę **backendu** aplikacji, opis implementowanych **endpointów**, integrację z bazą danych, mechanizmy uwierzytelniania, proces konteneryzacji systemu oraz sposób zaimplementowania **cache**.

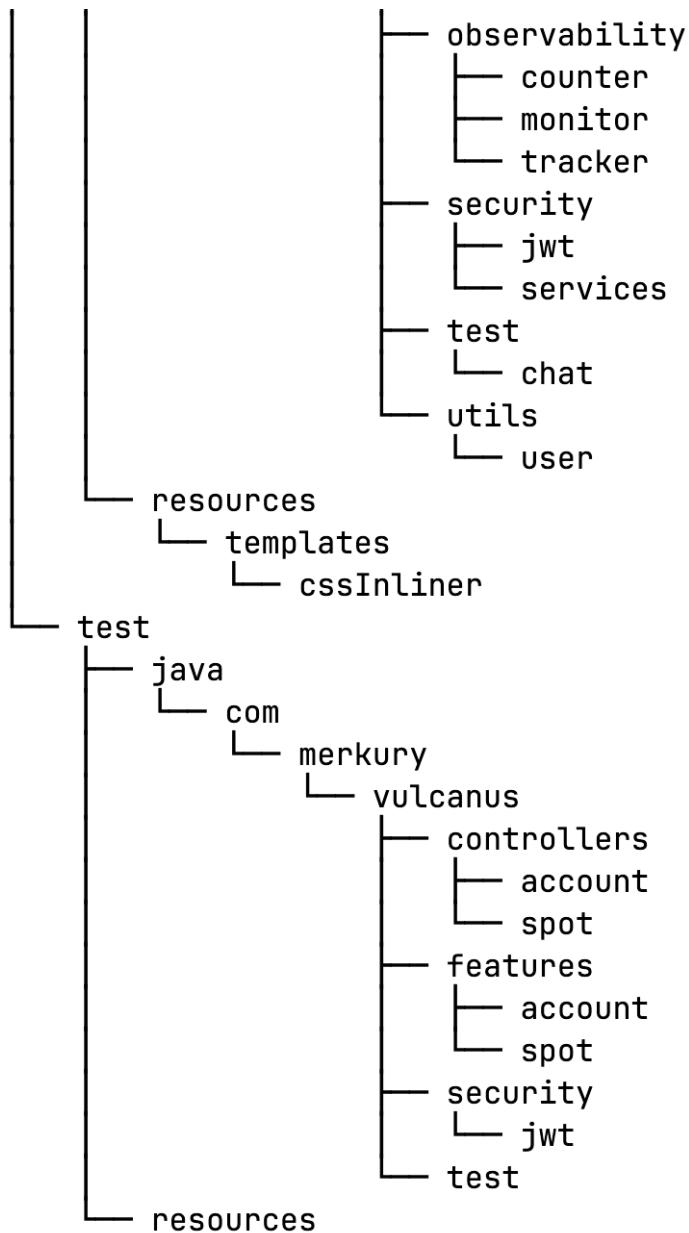
7.2.1 Struktura projektu

Backend aplikacji został zaimplementowany przy użyciu **frameworka** Spring Boot, co umożliwiło stworzenie spójnej i skalowej architektury w prosty sposób. W projekcie zastosowano rozwiązanie typu **REST API**, gdyż zespół projektowy dysponuje największym doświadczeniem w jego wykorzystaniu. Struktura projektu została zorganizowana zgodnie z podejściem **folder by type**, dzięki czemu każdy

plik znajduje się w odpowiadającym mu katalogu. Takie podejście ułatwia zarówno lokalizację istniejących plików, jak i określenie miejsca tworzenia nowych komponentów. Poniżej przedstawiono przykładową strukturę katalogów [backendu](#):



Rysunek 7.26: Struktura katalogów (1)



Rysunek 7.27: Struktura katalogów (2)

Dzięki takiej organizacji kod jest bardziej czytelny i łatwiejszy w utrzymaniu. Umożliwia również szybkie odnalezienie odpowiednich modułów oraz ułatwia rozbudowę projektu w przyszłości.

7.2.2 Endpointy systemu

Projektowany system udostępnia REST-owe API HTTP, za pomocą którego klienci komunikują się z serwerem.

Na potrzeby niniejszej pracy przez *endpoint REST API* rozumiany będzie konkretny punkt dostępu do systemu, zdefiniowany jako para:

metoda HTTP + ścieżka URL

pod którym aplikacja udostępnia określoną funkcjonalność lub zasób. Przykładowo, endpoint `GET /public/spot/{spotId}` służy do pobierania informacji o wybranym specie.

W dalszej części rozdziału przedstawiono zestawienie wszystkich endpointów HTTP systemu, a następnie szczegółowe karty wybranych endpointów, opisujące m.in. parametry wejściowe, `Query params` oraz strukturę odpowiedzi.

Zestawienie wszystkich endpointów HTTP panelu użytkownika	
Metoda HTTP	Ścieżka
GET	/user-dashboard/profile
GET	/public/user-dashboard/profile/{targetUsername}
PATCH	/user-dashboard/profile
GET	/user-dashboard/friends
GET	/public/user-dashboard/friends/{targetUsername}
PATCH	/user-dashboard/friends
PATCH	/user-dashboard/friends/change-status
GET	/user-dashboard/followers
GET	/public/user-dashboard/followers/{targetUsername}
GET	/user-dashboard/followed
GET	/public/user-dashboard/followed/{targetUsername}

GET	/user-dashboard/friends/find
GET	/user-dashboard/friends/invites
PATCH	/user-dashboard/followed
GET	/user-dashboard/favorite-spots
PATCH	/user-dashboard/favorite-spots
POST	/user-dashboard/add-spot-media
GET	/user-dashboard/is-spot-favourite
GET	/user-dashboard/photos
GET	/user-dashboard/comments
PATCH	/user-dashboard/settings
GET	/user-dashboard/settings
GET	/user-dashboard/movies
GET	/user-dashboard/photos/{targetUsername}
GET	/user-dashboard/add-spot
POST	/user-dashboard/add-spot
GET	/user-dashboard/add-spot/coordinates

Tabela 7.1: Zestawienie endpointów: panelu użytkownika

Zestawienie wszystkich endpointów HTTP modułu spotów	
Metoda HTTP	Ścieżka
GET	/public/spot/gallery
GET	/public/spot/gallery-media-position
GET	/public/spot/gallery-fullscreen-media
GET	/public/spot/current-view

GET	/public/spot/current-view/spot-names
GET	/public/spot/{spotId}
PATCH	/public/spot/increase-view-count
GET	/public/spot/search/map
GET	/public/spot/search/list
GET	/public/spot/names
GET	/public/spot/most-popular
GET	/public/spot/search/home-page
GET	/public/spot/search/home-page/locations
GET	/public/spot/search/home-page/advance
GET	/public/spot/get-spot-basic-weather
GET	/public/spot/get-spot-detailed-weather
GET	/public/spot/get-spot-wind-speeds
GET	/public/spot/get-spot-weather-timeline-plot-data
PATCH	/public/spot/increase-spot-media-views-count
PATCH	/public/spot/edit-spot-media-likes
GET	/spot/check-is-spot-media-liked
GET	/public/spot/get-spot-time-zone

Tabela 7.2: Zestawienie endpointów: modułu spotów

Zestawienie wszystkich endpointów HTTP komentarzy do spotów	
Metoda HTTP	Ścieżka
GET	/public/spot/{spotId}/comments
GET	/public/spot/{spotId}/comments/{commentId}

POST	/spot/{spotId}/comments
DELETE	/spot/comments/{commentId}
PATCH	/spot/comments/{commentId}
PATCH	/spot/comments/{commentId}/vote
GET	/spot/comments/vote-type

Tabela 7.3: Zestawienie endpointów: komentarzy do spotów

Zestawienie wszystkich endpointów HTTP postów forum	
Metoda HTTP	Ścieżka
GET	/public/post/{postId}
GET	/public/post
POST	/post
DELETE	/post/{postId}
PATCH	/post/{postId}
PATCH	/post/{postId}/vote
PATCH	/post/{postId}/follow
PATCH	/post/{postId}/report
GET	/public/categories-tags

Tabela 7.4: Zestawienie endpointów: postów forum

Zestawienie wszystkich endpointów HTTP komentarzy forum	
Metoda HTTP	Ścieżka

GET	/public/post/{postId}/comments
GET	/public/comments/{commentId}/replies
POST	/post/{postId}/comments
DELETE	/post/comments/{commentId}
PATCH	/post/comments/{commentId}
PATCH	/post/comments/{commentId}/vote
PATCH	/post/comments/{commentId}/report
POST	/comments/{commentId}/replies

Tabela 7.5: Zestawienie endpointów: komentarzy forum

Zestawienie wszystkich endpointów HTTP konta użytkownika i autoryzacji	
Metoda HTTP	Ścieżka
POST	/public/account/register
POST	/public/account/login
GET	/account/login-success
POST	/public/account/forgot-password
POST	/public/account/set-new-password
GET	/account/check

Tabela 7.6: Zestawienie endpointów: konta użytkownika i autoryzacji

Zestawienie wszystkich endpointów HTTP integracji GIF-ów (Tenor)

Metoda HTTP	Ścieżka
GET	/gifs/trending
GET	/gifs/search

Tabela 7.7: Zestawienie endpointów: integracji GIF-ów

Zestawienie wszystkich endpointów HTTP modułu czatu	
Metoda HTTP	Ścieżka
GET	/chats/{chatId}/messages
GET	/chats/user-chats
POST	/chats/get-or-create-private-chat
POST	/chats/{chatId}/send-files
POST	/chats/create/group
PATCH	/chats/{chatId}
GET	/chats/group-chat/add/search/{chatId}
PUT	/chats/add/users/{chatId}

Tabela 7.8: Zestawienie endpointów: modułu czatu

7.2.2.1 Panel użytkownika

KARTA ENDPOINTU API	
Identyfikator:	EP01
Ścieżka:	/public/user-dashboard/profile/{targetUsername}

Nazwa:	Pobierz profil innego użytkownika (widok publiczny)
Parametry wejściowe:	<ul style="list-style-type: none"> • targetUsername: String (nazwa użytkownika w ścieżce URL)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found
Dane zwracane:	<ul style="list-style-type: none"> • profile: UserProfileDto, zawiera: <ul style="list-style-type: none"> – username: String – profilePhoto: String (URL) – followersCount: Integer – followedCount: Integer – friendsCount: Integer – photosCount: Integer – mostPopularPhotos: List<ImageDto> • friendStatus: UserFriendStatus • isFollowing: Boolean • isOwnProfile: Boolean

Tabela 7.9: Karta endpointu: /public/user-dashboard/profile/{targetUsername}

KARTA ENDPOINTU API	
Identyfikator:	EP02
Ścieżka:	/user-dashboard/favorite-spots
Nazwa:	Pobierz listę ulubionych spotów użytkownika

Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • type: FavoriteSpotsListType (typ listy: ulubione, odwiedzone oraz do odwiedzenia) • page: Integer (opcjonalnie, domyślnie 0) • size: Integer (opcjonalnie, domyślnie 10)
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized
Dane zwracane:	<ul style="list-style-type: none"> • items: List<FavoriteSpotDto>, gdzie każdy element zawiera: <ul style="list-style-type: none"> – id: Long – name: String – country: String – city: String – description: String – rating: Double – viewsCount: Integer – imageUrl: String – type: FavoriteSpotsListType – coords: SpotCoordinatesDto – tags: Set<SpotTagDto> • hasNext: boolean

Tabela 7.10: Karta endpointu: /user-dashboard/favorite-spots

KARTA ENDPOINTU API	
Identyfikator:	EP03
Ścieżka:	/user-dashboard/photos
Nazwa:	Pobierz posortowane zdjęcia użytkownika
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • type: DateSortType (typ sortowania po dacie) • from: LocalDate (opcjonalnie, data od) • to: LocalDate (opcjonalnie, data do) • page: Integer (opcjonalnie, domyślnie 0) • size: Integer (opcjonalnie, domyślnie 20)
Kod(y) statusu odpowiedzi:	200 OK, 400 Bad Request, 401 Unauthorized
Dane zwracane:	<ul style="list-style-type: none"> • items: List<DatedMediaGroupDto>, gdzie: <ul style="list-style-type: none"> – date: LocalDate (data grupy) – media: List<MediaDto>, każdy element: <ul style="list-style-type: none"> * src: String (URL) * heartsCount: Integer * viewsCount: Integer * id: Long • hasNext: boolean

Tabela 7.11: Karta endpointu: /user-dashboard/photos

KARTA ENDPOINTU API	
Identyfikator:	EP04

Ścieżka:	/user-dashboard/add-spot
Nazwa:	Pobierz listę spotów dodanych przez użytkownika
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • page: Integer (opcjonalnie, domyślnie 0) • size: Integer (opcjonalnie, domyślnie 20)
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized
Dane zwracane:	<ul style="list-style-type: none"> • items: List<AddSpotDto>, każdy element: <ul style="list-style-type: none"> – id: Long – name: String – description: String – country: String – region: String – city: String – street: String – borderPoints: List<BorderPoint> (x, y) – firstPhotoUrl: String • hasNext: boolean

Tabela 7.12: Karta endpointu: /user-dashboard/add-spot

KARTA ENDPOINTU API	
Identyfikator:	EP05

Ścieżka:	/user-dashboard/add-spot
Nazwa:	Dodaj nowy spot użytkownika
Parametry wejściowe:	<ul style="list-style-type: none"> spot: String (część multipart, JSON z danymi nowego spotu) media: List<MultipartFile> (część multipart, pliki multimedialne spota)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found, 401 Unauthorized
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.13: Karta endpointu: /user-dashboard/add-spot

7.2.2.2 Spotty i pogoda

KARTA ENDPOINTU API	
Identyfikator:	EP06
Ścieżka:	/public/spot/gallery
Nazwa:	Pobierz stronę galerii mediów dla spota
Parametry wejściowe:	Brak

Query params:	<ul style="list-style-type: none"> • spotId: Long (identyfikator spota) • mediaType: String (typ plików, wartość enum GenericMediaType, PHOTO, VIDEO) • sorting: String (kryterium sortowania, po dacie / popularności) • page: Integer (numer strony, domyślnie 0) • size: Integer (rozmiar strony, domyślnie 6)
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found
Dane zwracane:	<ul style="list-style-type: none"> • Page<SpotSidebarMediaGalleryDto>: stronicowana lista elementów galerii, gdzie każdy element zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator media) – url: String (URL pliku) – mediaType: GenericMediaType (typ pliku)

Tabela 7.14: Karta endpointu: /public/spot/gallery

KARTA ENDPOINTU API	
Identyfikator:	EP07
Ścieżka:	/public/spot/current-view
Nazwa:	Pobierz listę spotów w aktualnym widoku mapy
Parametry wejściowe:	Brak

Query params:	<ul style="list-style-type: none"> • swLng: double (długość geograficzna lewego dolnego rogu) • swLat: double (szerokość geograficzna lewego dolnego rogu) • neLng: double (długość geograficzna prawego górnego rogu) • neLat: double (szerokość geograficzna prawego górnego rogu) • name: String (fragment nazwy spotu, domyślnie pusty) • sorting: String (tryb sortowania, domyślnie none) • ratingFrom: double (minimalna ocena, domyślnie 0.0) • page: Integer (numer strony, domyślnie 0)
Kod(y) statusu odpowiedzi:	200 OK
Dane zwracane:	<ul style="list-style-type: none"> • Page<SearchSpotDto>: stronicowana lista spotów, gdzie każdy element zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator spota) – name: String (nazwa spota) – rating: Double (0–5) – ratingCount: Integer (liczba ocen) – firstPhoto: String (URL pierwszego zdjęcia) – tags: Set<SpotTagDto> (tagi spota) – centerPoint: BorderPoint (środek obszaru spota)

Tabela 7.15: Karta endpointu: /public/spot/current-view

KARTA ENDPOINTU API

Identyfikator:	EP08
Ścieżka:	/public/spot/get-spot-basic-weather
Nazwa:	Pobierz podstawowe informacje pogodowe dla wskazanej lokalizacji
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • latitude: double (szerokość geograficzna) • longitude: double (długość geograficzna)
Kod(y) statusu odpowiedzi:	200 OK
Dane zwracane:	<ul style="list-style-type: none"> • temperature: Double (temperatura) • weatherCode: int (kod warunków pogodowych) • windSpeed: Double (prędkość wiatru) • isDay: boolean (czy jest dzień)

Tabela 7.16: Karta endpointu: /public/spot/get-spot-basic-weather

KARTA ENDPOINTU API	
Identyfikator:	EP09
Ścieżka:	/public/spot/get-spot-detailed-weather
Nazwa:	Pobierz szczegółowe informacje pogodowe dla wskazanej lokalizacji
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • latitude: double (szerokość geograficzna) • longitude: double (długość geograficzna)

Kod(y) statusu odpowiedzi:	200 OK
Dane zwracane:	<ul style="list-style-type: none"> • temperature: Double (temperatura) • weatherCode: int (kod warunków pogodowych) • precipitationProbability: Double (prawdopodobieństwo opadów) • dewPoint: Double (punkt rosy) • relativeHumidity: Double (wilgotność względna) • isDay: boolean (czy jest dzień) • uvIndexMax: Double (maksymalny indeks UV)

Tabela 7.17: Karta endpointu: /public/spot/get-spot-detailed-weather

KARTA ENDPOINTU API	
Identyfikator:	EP10
Ścieżka:	/public/spot/get-spot-wind-speeds
Nazwa:	Pobierz prędkości wiatru dla spotu na różnych wysokościach
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • latitude: double (szerokość geograficzna) • longitude: double (długość geograficzna) • spotId: long (identyfikator spota)
Kod(y) statusu odpowiedzi:	200 OK

Dane zwracane:	<ul style="list-style-type: none"> • windSpeeds100m: Double (prędkość wiatru na wysokości 100 metrów) • windSpeeds200m: Double (prędkość wiatru na wysokości 200 metrów) • windSpeeds300m: Double (prędkość wiatru na wysokości 300 metrów) • windSpeeds500m: Double (prędkość wiatru na wysokości 500 metrów) • windSpeeds750m: Double (prędkość wiatru na wysokości 750 metrów) • windSpeeds1000m: Double (prędkość wiatru na wysokości 1000 metrów)
-----------------------	--

Tabela 7.18: Karta endpointu: /public/spot/get-spot-wind-speeds

7.2.2.3 Wyszukiwarka spotów

KARTA ENDPOINTU API	
Identyfikator:	EP11
Ścieżka:	/public/spot/most-popular
Nazwa:	Pobierz 18 najpopularniejszych spotów
Parametry wejściowe:	Brak
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK

Dane zwracane:	<ul style="list-style-type: none"> • items: List<TopRatedSpotDto> każdy element zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator spota) – name: String (nazwa spota) – city: String (miasto, w którym znajduje się spot) – imageUrl: String (URL zdjęcia spota)
-----------------------	---

Tabela 7.19: Karta endpointu: /public/spot/most-popular

KARTA ENDPOINTU API	
Identyfikator:	EP12
Ścieżka:	/public/spot/search/home-page
Nazwa:	Wyszukaj spedy na stronie głównej na podstawie lokalizacji
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • country: String (opcjonalnie, kraj) • region: String (opcjonalnie, region) • city: String (opcjonalnie, miasto) • userLongitude: Double (opcjonalnie, długość geograficzna użytkownika) • userLatitude: Double (opcjonalnie, szerokość geograficzna użytkownika) • page: Integer (numer strony, domyślnie 0) • size: Integer (rozmiar strony, domyślnie 20)
Kod(y) statusu odpowiedzi:	200 OK

Dane zwracane:	<ul style="list-style-type: none"> • items: List<HomePageSpotDto> (lista znalezionych spotów), każdy element: <ul style="list-style-type: none"> – id: Long (identyfikator spota) – name: String (nazwa spota) – rating: Double (średnia ocena spota) – ratingCount: Integer (liczba ocen spota) – firstPhoto: String (URL pierwszego zdjęcia spota) – tags: Set<SpotTagDto> (zestaw tagów przypisanych do spota) – centerPoint: BorderPoint (punkt centralny obszaru spota) – city: String (miasto, w którym znajduje się spot) – distanceToUser: Double (odległość od lokalizacji użytkownika, jeśli dostępna) • hasNext: boolean (czy istnieje kolejna strona wyników)
-----------------------	--

Tabela 7.20: Karta endpointu: /public/spot/search/home-page

KARTA ENDPOINTU API	
Identyfikator:	EP13
Ścieżka:	/public/spot/search/home-page/locations
Nazwa:	Pobierz listę podpowiedzi lokalizacji dla wyszukiwarki na stronie głównej
Parametry wejściowe:	Brak

Query params:	<ul style="list-style-type: none"> • query: String (frazą wyszukiwania) • type: String (typ lokalizacji, kraj/region/miasto)
Kod(y) statusu odpowiedzi:	200 OK
Dane zwracane:	<ul style="list-style-type: none"> • List<String>: lista podpowiedzi (nazwy lokalizacji)

Tabela 7.21: Karta endpointu: /public/spot/search/home-page/locations

KARTA ENDPOINTU API	
Identyfikator:	EP14
Ścieżka:	/public/spot/search/home-page/advance
Nazwa:	Wyszukaj spoty na stronie głównej (zaawansowane filtrowanie)
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • city: String (opcjonalnie, miasto wyszukiwania) • tags: List<String> (opcjonalnie, lista tagów spota) • userLongitude: Double (opcjonalnie, długość geograficzna użytkownika) • userLatitude: Double (opcjonalnie, szerokość geograficzna użytkownika) • sort: SpotSortType (opcjonalnie, typ sortowania wyników) • filter: SpotRatingFilterType (opcjonalnie, filtr po ocenie) • page: Integer (numer strony, domyślnie 0) • size: Integer (rozmiar strony, domyślnie 20)

Kod(y) statusu odpowiedzi:	200 OK
Dane zwracane:	<ul style="list-style-type: none"> • items: List<HomePageSpotDto> (lista znalezionych spotów), każdy element: <ul style="list-style-type: none"> – id: Long (identyfikator spota) – name: String (nazwa spota) – rating: Double (średnia ocena spota) – ratingCount: Integer (liczba ocen spota) – firstPhoto: String (URL pierwszego zdjęcia spota) – tags: Set<SpotTagDto> (zestaw tagów przypisanych do spota) – centerPoint: BorderPoint (punkt centralny obszaru spota) – city: String (miasto, w którym znajduje się spot) – distanceToUser: Double (odległość od lokalizacji użytkownika, jeśli dostępna) • hasNext: boolean (czy istnieje kolejna strona wyników)

Tabela 7.22: Karta endpointu: /public/spot/search/home-page/advance

7.2.2.4 Komentarze do spotów

KARTA ENDPOINTU API	
Identyfikator:	EP15
Ścieżka:	/public/spot/{spotId}/comments

Nazwa:	Pobierz stronicowaną listę komentarzy dla wskazanego spota
Parametry wejściowe:	<ul style="list-style-type: none"> • spotId: Long (identyfikator spota w ścieżce URL)
Query params:	<ul style="list-style-type: none"> • page: Integer (numer strony, domyślnie 0; rozmiar strony = 2)
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found, 401 Unauthorized

Dane zwracane:	<ul style="list-style-type: none"> • Page<SpotCommentDto>: stronicowana lista komentarzy dla danego spota, każdy element zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator komentarza) – author: SpotCommentAuthorDto (dane autora komentarza) – text: String (treść komentarza) – rating: Double (ocena spota wystawiona w komentarzu, 0–5) – upvotes: Integer (liczba głosów pozytywnych na komentarz) – downvotes: Integer (liczba głosów negatywnych na komentarz) – publishDate: LocalDateTime (data i godzina publikacji komentarza) – isUpVoted: Boolean (czy bieżący użytkownik oddał głos w górę na ten komentarz) – isDownVoted: Boolean (czy bieżący użytkownik oddał głos w dół na ten komentarz) – numberOfMedia: Integer (łączna liczba dołączonych plików multimedialnych) – mediaList: List<SpotCommentMediaDto> (lista pierwszych plików komentarza)
-----------------------	--

Tabela 7.23: Karta endpointu: /public/spot/{spotId}/comments

KARTA ENDPOINTU API

Identyfikator:	EP16
Ścieżka:	/public/spot/{spotId}/comments/{commentId}
Nazwa:	Pobierz pełną listę mediów powiązanych z komentarzem
Parametry wejściowe:	<ul style="list-style-type: none"> spotId: Long (identyfikator spota w ścieżce URL) commentId: Long (identyfikator komentarza w ścieżce URL)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found
Dane zwracane:	<ul style="list-style-type: none"> List<SpotCommentMediaDto>: pełna lista mediów powiązanych z komentarzem, każdy element: <ul style="list-style-type: none"> id: Long (identyfikator pliku multimedialnego) url: String (URL pliku, używany do pobrania/wyświetlenia) genericMediaType: GenericMediaType (typ pliku, PHOTO lub VIDEO)

Tabela 7.24: Karta endpointu: /public/spot/{spotId}/comments/{commentId}

KARTA ENDPOINTU API	
Identyfikator:	EP17
Ścieżka:	/spot/{spotId}/comments
Nazwa:	Dodaj nowy komentarz do wskazanego spota

Parametry wejściowe:	<ul style="list-style-type: none"> • spotId: Long (identyfikator spota w ścieżce URL) • body: SpotCommentAddDto (dane nowego komentarza), zawiera: <ul style="list-style-type: none"> – text: String (treść komentarza) – rating: Double (ocena spota w komentarzu, zakres 0–5) – mediaFiles: List<MultipartFile> (lista załączonych plików, zdjęcia/filmy)
Query params:	Brak
Kod(y) statusu odpowiedzi:	201 Created, 404 Not Found, 401 Unauthorized, 422 Unprocessable Entity
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.25: Karta endpointu: /spot/{spotId}/comments

KARTA ENDPOINTU API	
Identyfikator:	EP18
Ścieżka:	/spot/comments/{commentId}/vote
Nazwa:	Oddaj głos na komentarz (góra/dół)
Parametry wejściowe:	<ul style="list-style-type: none"> • commentId: Long (identyfikator komentarza w ścieżce URL)
Query params:	<ul style="list-style-type: none"> • isUpvote: boolean (true = głos w góre, false = głos w dół)

Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 404 Not Found, 409 Conflict, 403 Forbidden
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.26: Karta endpointu: /spot/comments/{commentId}/vote

KARTA ENDPOINTU API	
Identyfikator:	EP19
Ścieżka:	/spot/comments/vote-type
Nazwa:	Pobierz informację, jak bieżący użytkownik zagłosował na komentarz
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • commentId: Long (identyfikator komentarza)
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found, 401 Unauthorized
Dane zwracane:	<ul style="list-style-type: none"> • voteInfo: SpotCommentVoteType (typ oddanego głosu, UPVOTE, DOWNVOTE, NONE)

Tabela 7.27: Karta endpointu: /spot/comments/vote-type

7.2.2.5 Forum – posty

KARTA ENDPOINTU API	
Identyfikator:	EP20
Ścieżka:	/public/post/{postId}
Nazwa:	Pobierz szczegółowe informacje o poście
Parametry wejściowe:	<ul style="list-style-type: none"> • postId: Long (identyfikator posta w ścieżce URL)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found

<p>Dane zwracane:</p>	<ul style="list-style-type: none"> • PostDetailsDto, zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator posta) – title: String (tytuł posta) – content: String (pełna treść posta) – category: ForumCategoryDto (kategoria forum, do której należy post) – tags: List<ForumTagDto> (lista tagów przypisanych do posta) – author: AuthorDto (dane autora posta) – isAuthor: Boolean (czy bieżący użytkownik jest autorem posta) – isFollowed: Boolean (czy bieżący użytkownik obserwuje ten post) – publishDate: LocalDateTime (data i godzina publikacji posta) – views: Integer (liczba wyświetleń posta) – upVotes: Integer (liczba głosów w górę na post) – downVotes: Integer (liczba głosów w dół na post) – isUpVoted: Boolean (czy bieżący użytkownik oddał głos w górę na post) – isDownVoted: Boolean (czy bieżący użytkownik oddał głos w dół na post) – commentsCount: Integer (łączna liczba komentarzy pod postem)
------------------------------	--

Tabela 7.28: Karta endpointu: /public/post/{postId}

KARTA ENDPOINTU API	
Identyfikator:	EP21
Ścieżka:	/post
Nazwa:	Dodaj nowy post na forum
Parametry wejściowe:	<ul style="list-style-type: none"> • body: PostDto (dane nowego posta), zawiera: <ul style="list-style-type: none"> – title: String (tytuł posta) – content: String (pełna treść posta) – category: String (nazwa kategorii forum, do której ma trafić post) – tags: List<String> (lista nazw tagów przypisanych do posta)
Query params:	Brak
Kod(y) statusu odpowiedzi:	201 Created, 400 Bad Request, 404 Not Found, 409 Conflict, 422 Unprocessable Entity
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.29: Karta endpointu: /post

KARTA ENDPOINTU API	
Identyfikator:	EP22
Ścieżka:	/post/{postId}
Nazwa:	Usuń wybrany post
Parametry wejściowe:	<ul style="list-style-type: none"> • postId: Long (identyfikator posta w ścieżce URL)

Query params:	Brak
Kod(y) statusu odpowiedzi:	204 No Content, 401 Unauthorized, 404 Not Found
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.30: Karta endpointu: /post/{postId}

KARTA ENDPOINTU API	
Identyfikator:	EP23
Ścieżka:	/post/{postId}/vote
Nazwa:	Oddaj głos na post (góra/dół)
Parametry wejściowe:	<ul style="list-style-type: none"> • postId: Long (identyfikator posta w ścieżce URL)
Query params:	<ul style="list-style-type: none"> • isUpvote: boolean (true = głos w góre, false = głos w dół)
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 404 Not Found
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.31: Karta endpointu: /post/{postId}/vote

KARTA ENDPOINTU API	
Identyfikator:	EP24
Ścieżka:	/public/categories-tags
Nazwa:	Pobierz listę wszystkich kategorii i tagów forum
Parametry wejściowe:	Brak
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK
Dane zwracane:	<ul style="list-style-type: none"> • ForumCategoriesAndTagsDto (zestaw kategorii i tagów forum), zawiera: <ul style="list-style-type: none"> – categories: List<ForumCategoryDto> (lista dostępnych kategorii), gdzie każdy element zawiera: <ul style="list-style-type: none"> * id: Long (identyfikator kategorii) * name: String (nazwa kategorii) * description: String (opis kategorii) * colour: String (kolor kategorii) – tags: List<ForumTagDto> (lista dostępnych tagów), gdzie każdy element zawiera: <ul style="list-style-type: none"> * id: Long (identyfikator tagu) * name: String (nazwa tagu)

Tabela 7.32: Karta endpointu: /public/categories-tags

7.2.2.6 Forum – komentarze do postów

KARTA ENDPOINTU API	
Identyfikator:	EP25
Ścieżka:	/public/post/{postId}/comments
Nazwa:	Pobierz stronicowaną listę komentarzy posta
Parametry wejściowe:	<ul style="list-style-type: none"> • postId: Long (identyfikator posta w ścieżce URL)
Query params:	<ul style="list-style-type: none"> • page: Integer (numer strony, domyślnie 0) • size: Integer (liczba komentarzy na stronie, domyślnie 10) • sortBy: PostCommentSortField (pole sortowania, domyślnie PUBLISH_DATE) • sortDirection: SortDirection (kierunek sortowania, domyślnie DESC)
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found

Dane zwracane:	<ul style="list-style-type: none"> • Page<PostCommentGeneralDto>, każdy element zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator komentarza) – content: String (treść komentarza) – upVotes: Integer (liczba głosów w góre) – downVotes: Integer (liczba głosów w dół) – repliesCount: Integer (liczba odpowiedzi) – publishDate: LocalDateTime (data publikacji) – author: AuthorDto (dane autora) – isAuthor: Boolean (czy bieżący użytkownik jest autorem) – isUpVoted: Boolean (czy użytkownik zagłosował w góre) – isDownVoted: Boolean (czy użytkownik zagłosował w dół) – isReply: Boolean (czy komentarz jest odpowiedzią) – isDeleted: Boolean (czy komentarz został usunięty logicznie)
-----------------------	---

Tabela 7.33: Karta endpointu: /public/post/{postId}/comments

KARTA ENDPOINTU API	
Identyfikator:	EP26
Ścieżka:	/post/{postId}/comments
Nazwa:	Dodaj nowy komentarz do posta

Parametry wejściowe:	<ul style="list-style-type: none"> • postId: Long (identyfikator posta w ścieżce URL) • body: PostCommentDto, zawiera: <ul style="list-style-type: none"> – content: String (treść komentarza)
Query params:	Brak
Kod(y) statusu odpowiedzi:	201 Created, 400 Bad Request, 404 Not Found, 422 Unprocessable Entity
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.34: Karta endpointu: /post/{postId}/comments

KARTA ENDPOINTU API	
Identyfikator:	EP27
Ścieżka:	/post/comments/{commentId}
Nazwa:	Edytuj istniejący komentarz do posta
Parametry wejściowe:	<ul style="list-style-type: none"> • commentId: Long (identyfikator komentarza w ścieżce URL) • body: PostCommentDto, zawiera: <ul style="list-style-type: none"> – content: String (treść komentarza)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 422 Unprocessable Entity

Dane zwracane:	Brak (pusta odpowiedź)
-----------------------	------------------------

Tabela 7.35: Karta endpointu: /post/comments/{commentId}

KARTA ENDPOINTU API	
Identyfikator:	EP28
Ścieżka:	/post/comments/{commentId}/vote
Nazwa:	Oddaj głos na komentarz (góra/dół)
Parametry wejściowe:	<ul style="list-style-type: none"> • commentId: Long (identyfikator komentarza w ścieżce URL)
Query params:	<ul style="list-style-type: none"> • isUpvote: boolean (true = głos w góre, false = głos w dół)
Kod(y) statusu odpowiedzi:	200 OK, 403 Forbidden, 404 Not Found
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.36: Karta endpointu: /post/comments/{commentId}/vote

KARTA ENDPOINTU API	
Identyfikator:	EP29
Ścieżka:	/comments/{commentId}/replies
Nazwa:	Dodaj odpowiedź na komentarz

Parametry wejściowe:	<ul style="list-style-type: none"> • commentId: Long (identyfikator komentarza nadzędnego w ścieżce URL) • body: PostCommentDto, zawiera: <ul style="list-style-type: none"> – content: String (treść komentarza)
Query params:	Brak
Kod(y) statusu odpowiedzi:	201 Created, 400 Bad Request, 404 Not Found, 409 Conflict, 422 Unprocessable Entity
Dane zwracane:	Brak (pusta odpowiedź)

Tabela 7.37: Karta endpointu: /comments/{commentId}/replies

7.2.2.7 Konto użytkownika – rejestracja, logowanie, hasło

KARTA ENDPOINTU API	
Identyfikator:	EP30
Ścieżka:	/public/account/register
Nazwa:	Zarejestruj nowego użytkownika
Parametry wejściowe:	<ul style="list-style-type: none"> • body: UserRegisterDto, zawiera: <ul style="list-style-type: none"> – username: String – email: String – password: String
Query params:	Brak

Kod(y) statusu odpowiedzi:	201 Created, 401 Unauthorized, 409 Conflict, 422 Unprocessable Entity, 500 Internal Server Error
Dane zwracane:	<ul style="list-style-type: none"> • body: String (komunikat) • JWT tokeny ustawione w ciasteczkach HTTP-only

Tabela 7.38: Karta endpointu: /public/account/register

KARTA ENDPOINTU API	
Identyfikator:	EP31
Ścieżka:	/public/account/login
Nazwa:	Zaloguj użytkownika
Parametry wejściowe:	<ul style="list-style-type: none"> • body: UserLoginDto, zawiera: <ul style="list-style-type: none"> – username: String – password: String
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 422 Unprocessable Entity
Dane zwracane:	<ul style="list-style-type: none"> • Brak (pusta odpowiedź w body) • JWT tokeny zwrócone w ciasteczkach HTTP-only

Tabela 7.39: Karta endpointu: /public/account/login

KARTA ENDPOINTU API	
---------------------	--

Identyfikator:	EP32
Ścieżka:	/public/account/forgot-password
Nazwa:	Rozpocznij procedurę resetu hasła (wyślij link na e-mail)
Parametry wejściowe:	<ul style="list-style-type: none"> • body: String (adres e-mail użytkownika w treści żądania)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 404 Not Found, 422 Unprocessable Entity, 500 Internal Server Error
Dane zwracane:	<ul style="list-style-type: none"> • body: String (komunikat) • Link resetujący hasło wysłany na podany adres e-mail

Tabela 7.40: Karta endpointu: /public/account/forgot-password

KARTA ENDPOINTU API	
Identyfikator:	EP33
Ścieżka:	/public/account/set-new-password
Nazwa:	Ustaw nowe hasło użytkownika na podstawie tokenu resetującego
Parametry wejściowe:	<ul style="list-style-type: none"> • body: UserPasswordResetDto, zawiera: <ul style="list-style-type: none"> – token: String (UUID – token resetu hasła) – password: String
Query params:	Brak

Kod(y) statusu odpowiedzi:	200 OK, 400 Bad Request, 404 Not Found, 422 Unprocessable Entity
Dane zwracane:	<ul style="list-style-type: none"> • body: String (komunikat)

Tabela 7.41: Karta endpointu: /public/account/set-new-password

KARTA ENDPOINTU API	
Identyfikator:	EP34
Ścieżka:	/account/check
Nazwa:	Sprawdź, czy użytkownik jest uwierzytelniony
Parametry wejściowe:	Brak
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 403 Forbidden
Dane zwracane:	<ul style="list-style-type: none"> • Brak (pusta odpowiedź; sam status informuje o uwierzytelnieniu)

Tabela 7.42: Karta endpointu: /account/check

KARTA ENDPOINTU API	
Identyfikator:	EP35

Ścieżka:	/account/login-success
Nazwa:	Obsłuż użytkownika zalogowanego przez OAuth2 i przekieruj go do aplikacji
Parametry wejściowe:	<ul style="list-style-type: none"> Brak klasycznego body – endpoint wywoływany jest jako redirect callback po poprawnym logowaniu przez dostawcę OAuth2. Kontekst użytkownika przekazywany jest w obiekcie <code>OAuth2AuthenticationToken</code>.
Query params:	Brak
Kod(y) statusu odpowiedzi:	302 Found (redirect), 404 Not Found, 409 Conflict, 422 Unprocessable Entity, 500 Internal Server Error
Dane zwracane:	<ul style="list-style-type: none"> Przekierowanie użytkownika na stronę frontendową skonfigurowaną w <code>UrlsProperties.afterLoginPageUrl</code>.

Tabela 7.43: Karta endpointu: /account/login-success

7.2.2.8 GIF-y (Tenor) – integracja czatu

KARTA ENDPOINTU API	
Identyfikator:	EP36
Ścieżka:	/gifs/trending
Nazwa:	Pobierz listę trendujących kategorii GIF-ów z Tenor
Parametry wejściowe:	Brak
Query params:	Brak

Kod(y) statusu odpowiedzi:	200 OK, 500 Internal Server Error
Dane zwracane:	<ul style="list-style-type: none"> • items: List<TenorGifCategoryDto>, każdy element zawiera: <ul style="list-style-type: none"> – searchTerm: String (frazą wyszukiwania powiązana z kategorią) – path: String (ścieżka kategorii w Tenor) – gifUrl: String (URL GIF-a)

Tabela 7.44: Karta endpointu: /gifs/trending

KARTA ENDPOINTU API	
Identyfikator:	EP37
Ścieżka:	/gifs/search
Nazwa:	Wyszukaj GIF-y po frazie tekstowej
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • searchPhrase: String (frazą wyszukiwania) • next: String (token paginacji zwrócony z poprzedniego wywołania; dla pierwszego zapytania może być pusty)
Kod(y) statusu odpowiedzi:	200 OK, 500 Internal Server Error

Dane zwracane:	<ul style="list-style-type: none"> • body: TenorGifSearchWrapperDto (wyniki wyszukiwania GIF-ów), zawiera: <ul style="list-style-type: none"> – gifs: List<TenorGifSearchDto> (lista pasujących GIF-ów), każdy element: <ul style="list-style-type: none"> * url: String (URL GIF-a) – next: String (token do pobrania kolejnej strony wyników)
-----------------------	---

Tabela 7.45: Karta endpointu: /gifs/search

Czat – REST API

KARTA ENDPOINTU API	
Identyfikator:	EP38
Ścieżka:	/chats/{chatId}/messages
Nazwa:	Pobierz stronicowane wiadomości dla wybranego czatu
Parametry wejściowe:	<ul style="list-style-type: none"> • chatId: Long (identyfikator czatu)
Query params:	<ul style="list-style-type: none"> • pageParam: Integer (numer strony wiadomości, domyślnie 1 – pierwsza strona po wstępnym pobraniu) • numberOfMessagesPerPage: Integer (liczba wiadomości na stronę, domyślnie 20)
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 404 Not Found

Dane zwracane:	<ul style="list-style-type: none"> • body: ChatMessageDtoSlice, zawiera: <ul style="list-style-type: none"> – messages: List<ChatMessageDto>, każdy element: <ul style="list-style-type: none"> * id: Long (identyfikator wiadomości) * sender: ChatMessageSenderDto (dane nadawcy wiadomości) * sentAt: LocalDateTime (data i godzina wysłania wiadomości) * content: String (treść wiadomości; dla wiadomości plikowych może być pusty) * chatId: Long (identyfikator czatu, do którego należy wiadomość) * attachedFiles: List<ChatMessageAttachedFileDto> (lista załączonych plików) – hasNextSlice: Boolean (czy istnieje kolejna „strona” / porcja wiadomości) – numberOfMessages: Integer (liczba wiadomości zwróconych w tej odpowiedzi) – sliceNumber: Integer (numer bieżącej porcji wiadomości)
-----------------------	--

Tabela 7.46: Karta endpointu: /chats/{chatId}/messages

KARTA ENDPOINTU API	
Identyfikator:	EP39
Ścieżka:	/chats/get-or-create-private-chat

Nazwa:	Pobierz istniejący lub utwórz nowy prywatny czat z użytkownikiem
Parametry wejściowe:	Brak
Query params:	<ul style="list-style-type: none"> • receiverUsername: String (nazwa użytkownika, z którym chcemy rozpocząć lub kontynuować rozmowę) • chatId: Long (opcjonalnie, identyfikator istniejącego czatu – jeśli jest już znany)
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 404 Not Found
Dane zwracane:	<ul style="list-style-type: none"> • body: ChatDto (szczegóły czatu), zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator czatu) – name: String (nazwa czatu – nazwa grupy lub nazwa rozmówcy) – lastMessage: ChatMessageDto (ostatnia wiadomość w czacie, jeśli istnieje) – imageUrl: String (URL avatara czatu lub rozmówcy) – messages: List<ChatMessageDto> (lista wiadomości zwróconych razem z czatem) – chatType: ChatType (typ czatu: PRIVATE lub GROUP) – participants: List<ChatParticipantDto> (lista uczestników czatu)

Tabela 7.47: Karta endpointu: /chats/get-or-create-private-chat

KARTA ENDPOINTU API	
Identyfikator:	EP40
Ścieżka:	/chats/{chatId}/send-files
Nazwa:	Wyślij jeden lub wiele plików w ramach czatu
Parametry wejściowe:	<ul style="list-style-type: none"> • chatId: Long (identyfikator czatu w ścieżce) • media: List<MultipartFile> (lista załączanych plików do wysłania w wiadomości)
Query params:	Brak
Kod(y) statusu odpowiedzi:	201 Created, 401 Unauthorized, 404 Not Found, 413 Payload Too Large, 415 Unsupported Media Type, 500 Internal Server Error
Dane zwracane:	<ul style="list-style-type: none"> • Brak (pusta odpowiedź; wiadomości z plikami pojawią się w historii czatu)

Tabela 7.48: Karta endpointu: /chats/{chatId}/send-files

KARTA ENDPOINTU API	
Identyfikator:	EP41
Ścieżka:	/chats/create/group
Nazwa:	Utwórz nowy czat grupowy
Parametry wejściowe:	<ul style="list-style-type: none"> • body: CreateGroupChatDto (dane nowego czatu grupowego), zawiera: <ul style="list-style-type: none"> – usernames: List<String> (lista nazw użytkowników, którzy mają zostać uczestnikami czatu) – ownerUsername: String (nazwa właściciela / twórcy czatu)

Query params:	Brak
Kod(y) statusu odpowiedzi:	201 Created, 400 Bad Request, 401 Unauthorized
Dane zwracane:	<ul style="list-style-type: none"> • body: ChatDto (utworzony czat grupowy), zawiera: <ul style="list-style-type: none"> – id: Long (identyfikator czatu) – name: String (nazwa czatu – nazwa grupy) – lastMessage: ChatMessageDto (ostatnia wiadomość w czacie, jeśli istnieje) – imageUrl: String (URL avatara czatu) – messages: List<ChatMessageDto> (lista wiadomości zwróconych razem z czatem) – chatType: ChatType (typ czatu, PRIVATE lub GROUP) – participants: List<ChatParticipantDto> (lista uczestników czatu)

Tabela 7.49: Karta endpointu: /chats/create/group

KARTA ENDPOINTU API	
Identyfikator:	EP42
Ścieżka:	/chats/{chatId}
Nazwa:	Zaktualizuj dane czatu grupowego (nazwa, zdjęcie)

Parametry wejściowe:	<ul style="list-style-type: none"> • chatId: Long (identyfikator czatu grupowego) • updateGroupChatDto: UpdateGroupChatDto (wysyłany jako multipart/form-data, zawiera dane do zmiany, nowa nazwa, nowe zdjęcie)
Query params:	Brak
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 404 Not Found, 413 Payload Too Large, 415 Unsupported Media Type, 500 Internal Server Error
Dane zwracane:	<ul style="list-style-type: none"> • body: UpdatedGroupChatDto (zaktualizowane dane czatu grupowego), zawiera: <ul style="list-style-type: none"> – newName: String (aktualna nazwa czatu po zmianie) – newImgUrl: String (aktualny URL obrazka grupy po zmianie)

Tabela 7.50: Karta endpointu: /chats/{chatId}

KARTA ENDPOINTU API	
Identyfikator:	EP43
Ścieżka:	/chats/group-chat/add/search/{chatId}
Nazwa:	Wyszukaj potencjalnych użytkowników do dodania do czatu grupowego
Parametry wejściowe:	<ul style="list-style-type: none"> • chatId: Long (identyfikator czatu grupowego, do którego chcemy dodać użytkowników)

Query params:	<ul style="list-style-type: none"> • query: String (fraza wyszukiwania po nazwie użytkownika) • page: Integer (numer strony wyników, domyślnie 0) • size: Integer (liczba wyników na stronę, domyślnie 20)
Kod(y) statusu odpowiedzi:	200 OK, 401 Unauthorized, 404 Not Found
Dane zwracane:	<ul style="list-style-type: none"> • body: SimpleSliceDto<PotentialChatMemberDto> (stworzona lista potencjalnych uczestników czatu), zawiera: <ul style="list-style-type: none"> – hasNext: boolean (czy istnieje kolejna „strona” wyników) – collection: Collection<PotentialChatMemberDto> (kolekcja potencjalnych użytkowników), każdy element: <ul style="list-style-type: none"> * username: String (nazwa użytkownika) * profileImg: String (URL zdjęcia profilowego użytkownika)

Tabela 7.51: Karta endpointu: /chats/group-chat/add/search/{chatId}

7.2.3 Integracja z bazą danych

W aplikacji wykorzystano relacyjną **bazę danych** PostgreSQL, która w środowisku deweloperskim uruchamiana jest jako kontener w aplikacji Docker. Komunikacja **backendu** z bazą danych odbywa się z wykorzystaniem wzorca Repository oraz **bibliotek** oferowanych przez Spring Boot, co umożliwia efektywne zarządzanie danymi oraz utrzymanie spójności warstwy dostępu do danych.

W systemie zaimplementowano zestaw najistotniejszych tabel, które opisano poniżej:

- **chat-invitations** — przechowuje zaproszenia do czatów wysyłane użytkownikom.
- **chat-message-attached-file** — przechowuje pliki dołączone do wiadomości w czatach.
- **chat-messages** — zapisuje wiadomości wysyłane w czatach.
- **chat-participants** — zawiera informacje o uczestnikach poszczególnych czatów.
- **chats** — lista czatów dostępnych w systemie.
- **favorite-spots** — informacje o miejscach (spotach) oznaczonych jako ulubione przez użytkowników.
- **forum-categories** — kategorie, do których przypisywane są posty na forum.
- **forum-tags** — tagi przypisywane postom na forum.
- **friendships** — relacje znajomości między użytkownikami.
- **media** — ogólne media przesyłane przez użytkowników na forum (zdjęcia, filmy).
- **post-comment-down-votes** — przechowuje „minusy” nadawane komentarzom do postów.
- **post-comment-reports** — raporty zgłasiane przez użytkowników wobec komentarzy.
- **post-comment-up-votes** — przechowuje „plusy” nadawane komentarzom do postów.
- **post-comments** — komentarze użytkowników do postów.
- **post-down-votes** — „minusy” nadawane postom.
- **post-followers** — informacje o użytkownikach obserwujących dany post.

- **post-reports** — raporty zgłasiane wobec postów.
- **post-tags** — tagi przypisane do konkretnych postów.
- **post-up-votes** — „plusy” nadawane postom.
- **posts** — posty tworzone przez użytkowników.
- **spot-comment-down-votes** — „minusy” nadawane komentarzom do spotów.
- **spot-comment-media** — pliki multimedialne dołączone do komentarzy przy spotach.
- **spot-comment-up-votes** — „plusy” nadawane komentarzom do spotów.
- **spot-comments** — komentarze użytkowników do spotów.
- **spot-media** — pliki multimedialne związane z konkretnymi spotami.
- **spots** — baza spotów w systemie.
- **spots-tags** — tagi przypisane do poszczególnych spotów.
- **tags-of-spots** — alternatywna tabela z tagami dla spotów.
- **user-followed-posts** — lista postów śledzonych przez użytkowników.
- **user-followers** — relacje obserwujących użytkowników.
- **user-liked-spot-media** — informacja o polubieniach mediów powiązanych ze spotami.
- **users** — dane użytkowników systemu.

7.2.4 Obsługa uwierzytelnienia

7.2.5 Konteneryzacja

W celu zapewnienia łatwego uruchamiania aplikacji oraz możliwości jej skalowania zastosowano konteneryzację z wykorzystaniem [Dockera](#). Zarówno relacyjna [baza danych](#) PostgreSQL, jak i [Redis](#) zostały uruchomione w odseparowanych [kontenerach](#). Dołączenia wielu kontenerów jednocześnie wykorzystano narzędzie [Docker Compose](#) (rys. 7.28), dzięki czemu wystarczy użyć jednego pliku konfiguracyjnego, a wszystkie wymagane usługi znajdują się automatycznie uruchomione z odpowiednimi ustawieniami.

Poniżej opisano skonteneryzowane [bazy danych](#):

PostgreSQL – usługa uruchamiana jest na podstawie obrazu `postgres:latest`, pobieranego ze zdalnego repozytorium [Docker Hub](#). Kontener odpowiada za działanie relacyjnej [bazy danych](#) systemu. Parametry połączenia, takie jak nazwa użytkownika, hasło oraz nazwa [bazy danych](#), określono za pomocą zmiennych środowiskowych zdefiniowanych w pliku `postgres.env`. [Baza danych](#) jest udostępniana lokalnie na porcie 5432.

Redis – usługa uruchamiana jest na podstawie obrazu `redis:latest`, pobieranego ze zdalnego repozytorium [Docker Hub](#). Kontener odpowiada za działanie bazy typu in-memory wykorzystywanej do krótkoterminowego przechowywania danych oraz mechanizmów [cache'owania](#). Usługa jest udostępniana lokalnie na porcie 6379.

Zastosowanie [Dockera](#) i [Docker Compose](#) umożliwia łatwe odtworzenie środowiska na dowolnej maszynie, ogranicza liczbę ręcznych kroków konfiguracyjnych oraz ułatwia dalsze skalowanie i automatyzację procesu wdrażania aplikacji.

```
version: '3.8'

services:
  postgres-db:
    container_name: postgres-db
    image: postgres:latest
    env_file:
      - ./postgres/postgres.env
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    container_name: redis
    image: redis:latest
    ports:
      - "6379:6379"

volumes:
  postgres_data:
```

Rysunek 7.28: Plik konfiguracyjny Docker Compose

7.2.6 Cache

W aplikacji wykorzystano mechanizm **cache** do ograniczenia liczby powtarzalnych zapytań do usług zewnętrznych oraz odciążenia warstwy **backendowej** w przypadku często wywoywanych **endpointów**. Jako magazyn danych **cache** zastosowano **bazę klucz-wartość Redis**, uruchamianą w **kontenerze Docker**, co uprościło konfigurację środowiska oraz zapewniło spójność między uruchomieniami lokalnymi.

Do integracji ze środowiskiem [Spring Boot](#) użyto gotowego mechanizmu [Spring cache](#). Konfiguracja menedżera `cache` została oparta o `RedisCacheManager`, a dla poszczególnych przestrzeni `cache` zdefiniowano niezależne czasy życia wpisów ([TTL](#)). Dodatkowo wyłączono [cachowanie](#) wartości pustych (`null`), co zapobiega utrwalaniu błędnych lub niekompletnych odpowiedzi.

Poniżej przedstawiono konfigurację `cache` w projekcie (rys. 7.29).

```
@Configuration
public class RedisConfig {

    @Bean
    public RedisCacheManager cacheManager(RedisConnectionFactory cf) {
        RedisCacheConfiguration defaultConfig = RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofSeconds(30))
            .disableCachingNullValues();

        Map<String, RedisCacheConfiguration> configs = new HashMap<>();
        configs.put("filteredSpots", defaultConfig.entryTtl(Duration.ofSeconds(50)));
        configs.put("filteredSpotsNames", defaultConfig.entryTtl(Duration.ofSeconds(40)));
        configs.put("spotWeatherBasic", defaultConfig.entryTtl(Duration.ofMinutes(30)));
        configs.put("spotWeatherDetailed", defaultConfig.entryTtl(Duration.ofMinutes(30)));
        configs.put("spotWeatherWindSpeeds", defaultConfig.entryTtl(Duration.ofMinutes(30)));
        configs.put("spotWeatherTimelinePlotData", defaultConfig.entryTtl(Duration.ofMinutes(30)));
        configs.put("gifsTrendingTerms", defaultConfig.entryTtl(Duration.ofHours(1)));

        return RedisCacheManager.builder(cf)
            .cacheDefaults(defaultConfig)
            .withInitialCacheConfigurations(configs)
            .build();
    }

}
```

Rysunek 7.29: Konfiguracja `RedisCacheManager` wraz z przypisaniem TTL do wybranych przestrzeni `cache`.

Cachowanie danych realizowane jest [adnotacją](#) `@Cacheable`, która pozwala wskazać nazwę `cache` oraz sposób budowania klucza. W zależności od przypadku stosowane są również warunki `condition` i `unless`, aby pomijać zapytania o niepoprawnych parametrach oraz nie zapisywać w `cache` wyników pustych. Dla wybranych metod włączono tryb `sync=true`, co ogranicza zjawisko równoległego od-

pytywania źródła danych (*cache stampede*) w momencie wygaśnięcia wpisu.

Przykładowe użycie `cache` w projekcie przedstawiono poniżej (rys. 7.30–7.32).

```
@Cacheable(cacheNames = "gifsTrendingTerms", sync = true) 1 usage  Adam Langmesser
public Mono<List<TenorGifCategoryDto>> getTrendingCategories() { return client.getTrendingCategories(); }
```

Rysunek 7.30: Cachowanie danych z zewnętrznego źródła z wykorzystaniem `@Cacheable` oraz synchronizacji.

```
@Cacheable( 3 usages  stanoz
    value = "filteredSpots",
    key = "#name",
    condition = "#name != null && #name.trim().length() > 0",
    unless = "#result == null || #result.isEmpty()"
)
public List<GeneralSpotDto> getSearchedSpotsOnMap(String name) throws SpotsNotFoundException {...}

@Cacheable(value = "filteredSpotsNames", key = "#text", unless = "#result == null") 3 usages  stanoz
public List<String> getFilteredSpotsNames(String text) throws SpotsNotFoundException {...}
```

Rysunek 7.31: Cachowanie wyników wyszukiwania spotów z kontrolą warunków (`condition`, `unless`).

```

@Cacheable( 1 usage  & stanoz
    value = "spotWeatherBasic",
    key = "#latitude + ':' + #longitude"
)
public Mono<BasicSpotWeatherDto> getBasicSpotWeather(double latitude, double longitude) {...}

@Cacheable( 1 usage  & stanoz
    value = "spotWeatherDetailed",
    key = "#latitude + ':' + #longitude"
)
public Mono<DetailedSpotWeatherDto> getDetailedSpotWeather(double latitude, double longitude) {...}

@Cacheable( 1 usage  & stanoz
    value = "spotWeatherWindSpeeds",
    key = "#latitude + ':' + #longitude"
)
public Mono<SpotWeatherWindSpeedsDto> getSpotWeatherWindSpeeds(double latitude, double longitude, Long spotId) {...}

@Cacheable( 1 usage  & stanoz
    value = "spotWeatherTimelinePlotData",
    key = "#latitude + ':' + #longitude"
)
public Mono<List<SpotWeatherTimelinePlotDataDto>> getSpotWeatherTimelinePlotData(double latitude, double longitude, Long spotId)

```

Rysunek 7.32: Cachowanie danych pogodowych z kluczem opartym o współrzędne geograficzne.

Wartości [TTL](#) dobrano na podstawie charakteru danych: dla wyników wyszukiwania zastosowano krótsze czasy (częsta zmienność i większe ryzyko nieaktualności), natomiast dla danych pogodowych lub [gifów](#), które zmieniają się wolniej, ustawiono dłuższe czasy przechowywania. Takie podejście pozwoliło zredukować opóźnienia odpowiedzi oraz ograniczyć liczbę wywołań usług zewnętrznych, przy zachowaniu akceptownego poziomu aktualności danych.

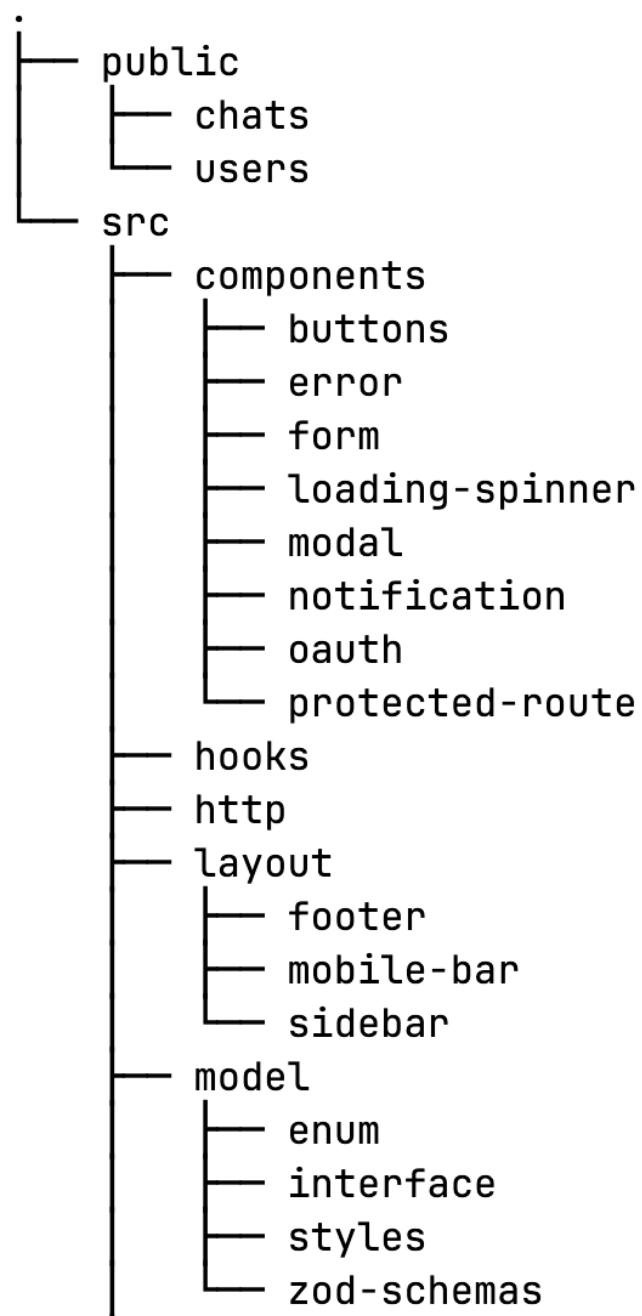
7.3 Implementacja frontendu

W niniejszym rozdziale przedstawiono proces implementacji części [frontendowej](#) aplikacji.

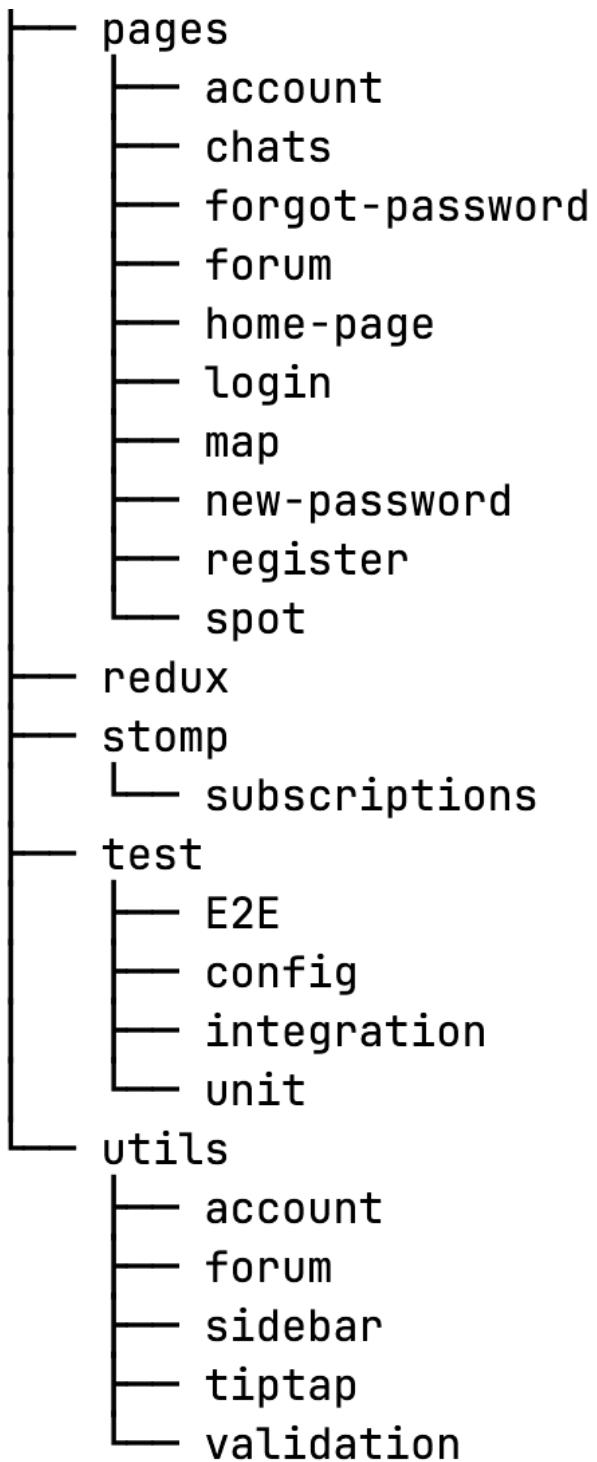
7.3.1 Struktura aplikacji

W niniejszym podrozdziale przedstawiona została struktura aplikacji [frontendowej](#) oraz organizację jej kluczowych elementów.

Architekturę aplikacji **frontendowej** zaprojektowano 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 przedstawiono na rysunkach [7.33](#) oraz [7.34](#).



Rysunek 7.33: Struktura katalogów (1)



Rysunek 7.34: 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 7.35: 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; Show usages ⚡ Adam Langmesser
```

Rysunek 7.36: 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 7.35 oraz 7.36), wybrane ścieżki opakowano w komponent **ProtectedRoute**. Komponent ten pełni

rolę bramki (rysunek 7.37).

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 }) { Show usages & Mredosz
  const isLoggedIn = useSelector(state) => state.account.isLoggedIn;

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

Rysunek 7.37: Implementacja komponentu bramki (`ProtectedRoute`)

7.3.2 Zarządzanie stanem i przepływ danych

W niniejszym podrozdziale opisano zastosowane w projekcie podejście do zarządzania **stanem** oraz organizację przepływu danych w aplikacji frontendowej.

W projekcie postawiono na zrównoważone podejście do zarządzania **stanem**. Korzysta się zarówno z lokalnego **stanu** komponentów (za pomocą **hooka useState**) [15], jak i ze **stanu** globalnego, utrzymywaneego przez bibliotekę **React Redux** [16]. Globalny **stan** wprowadzono w celu możliwie jak największego ograniczenia przekazywanie **propsów** w głąb drzewa komponentów oraz uniknąć niepotrzebnych ponownych renderów.

Do przechowywania **stanu** lokalnego, ograniczonego tylko do danego komponentu (lub jego najbliższych elementów podlegających), wykorzystuje się **hook useState**. Natomiast efekty uboczne i synchronizację realizuje się za pomocą **useEffect**. W przypadku bardziej złożonej logiki lub potrzeby ponownego wykorzystania kodu powstały **hooki** niestandardowe, takie jak **useScreenSize**, **useDarkMode** czy **useClickOutside**. Dzięki temu większość logiki prezentacji wydzielono z warstwy **UI**, co poprawia czytelność i ułatwia utrzymanie kodu.

Z racji tego, że korzystamy z **reacta** w połączeniu z **TypeScriptem**, przygoto-

wano również własne **hooki** wspomagające typowanie, takie jak `useDispatchTyped` oraz `useSelectorTyped`. Pozwalają one na bezpieczne typowanie akcji oraz selektorów **reduxa** bez konieczności powtarzania adnotacji typów w każdym komponencie. Fragmenty tej implementacji przedstawiono na rysunkach 7.38 oraz 7.39.

```

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,
      ],
      spotAddMediaModal: addSpotMediaModalSlice.reducer,
      forum: forumModalSlice.reducer,
      forumReport: forumReportModalSlice.reducer,
    ],
  },
});

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

```

Rysunek 7.38: 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 7.39: Przykładowy slice odpowiedzialny za sprawdzenie czy użytkownik jest zalogowany

7.3.3 Integracja i komunikacja z backendem

W niniejszym podrozdziale opisano sposób integracji aplikacji [frontendowej](#) z [backendem](#) oraz mechanizmy odpowiedzialne za bezpieczną i efektywną komunikację z serwerem.

Jest to kluczowy element aplikacji, ponieważ wymaga bezpiecznego przesyłania danych użytkownika. W celu uproszczenia komunikacji z serwerem zdecydowano się na wykorzystanie biblioteki [axios](#) [17] oraz [biblioteki TanStack Query](#) [18].

W przypadku ścieżek wymagających uwierzytelnienia do zapytania dołączany jest token **JWT**. Token przekazywany jest w ciasteczku **http only**, a jego wysyłanie realizowane jest automatycznie przez przeglądarkę dzięki ustawieniu parametru **withCredentials** na wartość **true**. W razie braku tokenu lub jego nieważności dostęp do danych nie jest przyznawany. Dodatkowo, po odświeżeniu strony lub wejściu na widok w sytuacji, gdy ciasteczko utraciło ważność, sesja jest uznawana za nieaktywną, a użytkownik zostaje automatycznie wylogowany. Podczas wylogowania inicjowanego przez użytkownika ciasteczko jest usuwane, co skutkuje utratą uprawnień do zasobów chronionych.

Przykładem pliku odpowiedzialnego za taką komunikację jest **account.js** (rys. 7.40 i 7.41), który obsługuje operacje związane z logowaniem, rejestracją, zmianą hasła oraz wylogowaniem.

```
export async function loginUser(user) { Show usages ▲ Adam Langmesser +1
  return await axios.post(`${BASE_URL}/public/account/login`, user, {
    withCredentials: true,
  });
}

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

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

Rysunek 7.40: 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 7.41: Implementacja modułu account (2)

Funkcje odpowiedzialne za komunikację z [backendem](#) umieszczone w katalogu `/http`. Dzięki temu są one skoncentrowane i mogą być w prosty sposób wykorzystywane w różnych częściach aplikacji. Zastosowanie TanStack Query umożliwiło znaczące ograniczenie powtarzanego kodu oraz uprościło obsługę błędów i stanów zapytania (takich jak ładowanie danych, błąd czy sukces). [Biblioteka](#) udostępnia m.in. wartość `isLoading`, dzięki czemu komponent może łatwo wyświetlić ekran ładowania bez konieczności ręcznego zarządzania własnym stanem. Dodatkowo [hook](#) `useQuery` pozwala na automatyczne pobieranie danych po wejściu na daną podstronę. Komponent deklaruje jedynie, jakie dane są mu potrzebne, a TanStack Query realizuje ich pobranie, cache'owanie oraz odświeżanie. Do operacji wymagających wywołania akcji po stronie użytkownika (np. wysłania formularza logowania) wykorzystywany jest [hook](#) `useMutation` z TanStack Query. Przykład użycia tego rozwiązania w procesie logowania przedstawiono na rys. 7.42.

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

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

Rysunek 7.42: Wykorzystanie TanStack Query przy logowaniu użytkownika

7.3.4 Style

W niniejszym podrozdziale przedstawiono zastosowane w projekcie rozwiązania dotyczące stylowania interfejsu użytkownika oraz narzędzia wykorzystywane do tworzenia spójnej i **responsywnej** warstwy wizualnej aplikacji.

Do stylowania interfejsu wykorzystano **framework** Tailwind CSS [19]. Dzięki gotowym klasom udostępnianym przez Tailwind wygląd elementów można definiować 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 zachować spójność wizualną w całej aplikacji. W pliku `index.css` zdefiniowano zmienne kolorystyczne (rys. 7.43 i 7.44). Dzięki temu zmiana motywów kolorystycznego w przyszłości sprowadza się do edycji wartości w jednym miejscu.



```
--height-1\10: 10%;  
--breakpoint-3xl: 160rem;  
--color-mainBlue: #4242f0;  
--color-mainBlueDarker: #0d0db5;  
--color-darkText: #e5e5e5;  
--color-darkBg: #0f0f10;  
--color-darkBgSoft: #1b1c1d;  
--color-grayBg: #d9d9d9;  
--color-darkBgMuted: #323539;  
--color-darkBorder: #939394;  
--color-lightText: #222222;  
--color-lightBg: #e4e3e3;  
--color-lightBgDarker: #cccaca;  
--color-lightBgSoft: #ffffff;  
--color-lightBgMuted: #f2f2f2;  
--color-lightBorder: #fbfdff;  
--color-lightGrayishViolet: #f2eef9;  
--color-whiteSmoke: #f6f6f6;  
--color-warmerWhiteSmoke: #ece9e9;  
--color-lightGrayishBlue: #e5e9ee;  
--color-paleBlueGray: #acafbb;  
--color-grayText: #d3d3d3;
```

Rysunek 7.43: Implementacja zmiennych kolorystycznych (1)



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

Rysunek 7.44: 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. Część aplikacji jest **responsywna**. Tailwind udostępnia predefiniowane prefiksy **responsywne** (np. **md:**, **lg:**) (rys. 7.45), utworzono również własny (**3xl:**) na ekrany o rozdzielczości 2560px. Pozwalają one przypisywać style zależnie od szerokości ekranu bez pisania własnych reguł **media queries**. Dzięki temu implementacja widoków mobilnych i desktopowych była znaczco szybsza.

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

Rysunek 7.45: 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. 7.46).

```
<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 7.46: Przykładowe użycie klas Tailwind (w tym wariantu **dark:**)

Aby uzyskać płynniejsze i przyjemniejsze animacje, wykorzystano **bibliotekę Motion** [20]. Pozwala ona w prosty sposób tworzyć animacje elementów interfejsu, bez potrzeby ręcznego pisania złożonych reguł **CSS**. W aplikacji wykorzystano ją

m.in. w polach formularza logowania i rejestracji (rys. 7.47). 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.

```
<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 7.47: Implementacja animacji z wykorzystaniem Motion

7.3.5 Wyszukiwarka spotów

W niniejszym rozdziale przedstawiono sposób implementacji wyszukiwarki spotów.

Jednym z głównych modułów aplikacji jest wyszukiwarka spotów, umożliwiająca szybkie odnalezienie interesujących lokalizacji. Funkcjonuje ona w dwóch wariantach: prostym i zaawansowanym (rys. 7.48 oraz 7.49).

```
<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 7.48: 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 7.49: Implementacja zaawansowanej wersji wyszukiwarki

Przełączanie pomiędzy tymi widokami odbywa się za pomocą przycisku umieszczonego w górnej części strony (rys. 7.50).

```
<div className="■dark:shadow-darkBgSoft flex rounded-full shadow-lg shadow-black/20">
  <NavLink
    to="/"
    className={({ isActive } : NavLinkRenderProps) : string =>
      `■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 =>
      `■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 7.50: Implementacja komponentu do przełączania trybów

W trybie prostym prezentowana jest karuzela (rys. 7.51) z dwunastoma najpopularniejszymi [spotami](#) w całej aplikacji. W tym widoku możliwe jest wyszukiwanie [spotów](#) 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 7.51: Implementacja karuzeli z najpopularniejszymi [spotami](#)

Widok zaawansowany udostępnia rozszerzoną wyszukiwarkę, która umożliwia filtrowanie wyników po mieście, tagach oraz ocenie, a także ich sortowanie według popularności i średniej oceny (rys. 7.49).

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

- `Switch` – służy do przełączania widoku między trybem podstawowym a zaawansowanym,
- `SearchBar` – podstawowa wyszukiwarka `spotów`,
- `Carousel` – wyświetla najpopularniejsze `spotty`,
- `SearchSpotList` – wyświetla wyszukane `spotty`.

W skład zaawansowanej wersji wchodzą następujące komponenty:

- `Switch` – służy do przełączania widoku między trybem podstawowym a zaawansowanym,
- `AdvanceSearchBar` – zaawansowana wyszukiwarka `spotów`,
- `SearchSpotList` – wyświetla wyszukane `spotty`.

Komponent `Switch` (rys. 7.50) zawiera dwa elementy `NavLink` z biblioteki React Router, co pozwala na przełączanie widoków bez konieczności przeładowywania całej strony.

W komponencie `SearchBar` (rys. 7.52) po wpisaniu co najmniej dwóch znaków wyświetlana jest lista podpowiedzi dla kraju, regionu oraz miasta, w zależności od aktualnie uzupełnianego pola. Po pojawienniu się listy możliwe jest wybranie odpowiedniej lokalizacji, co ułatwia określenie lokalizacji dostępnych `spotów`.

```

<div className="■dark:bg-darkBgSoft □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"; readonly id: string} : 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>
    <button
      className="■dark:bg-darkBgMuted ■dark:hover:bg-darkBgMuted/80 □bg-lightBgMuted
      □hover:bg-lightBgMuted/80 flex w-full cursor-pointer justify-center rounded-md p-2 md:w-fit"
      onClick={handleSearchSpots}
    >
      <FaSearch />
    </button>
  </div>
</div>

```

Rysunek 7.52: Implementacja prostej wyszukiwarki

Komponent `SearchSpotList` (rys. 7.53) 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 7.53: 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 prezentowana jest lokalizacja **spota** 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 wspólnie dzielonych.

7.3.6 Mapa

Mapa jest jednym z głównych modułów aplikacji. W niniejszym rozdziale przedstawiono jego implementację po stronie [frontendu](#).

Moduł ten składa się z czterech głównych części:

- mapy z zaznaczonymi `spotami`
- panelu ze szczegółami `spota`
- panelu z informacjami pogodowymi
- dużej galerii zdjęć i filmów

7.3.6.1 Mapa z zaznaczonymi spotami

Mapa zbudowana jest z następujących [komponentów](#):

- `MapPage`
- `Spots`
- `ZoomControlPanel`
- `ZoomControlButton`
- `UserLocationPanel`
- `SearchCurrentViewButton`
- `CurrentViewSpotsList`
- `CurrentViewSpotsFormsContainer`

- `CurrentViewSpotsNameSearchBar`
- `SpotsSortingForm`
- `RatingFromForm`
- `SpotsNameSearchBar`
- `SearchedSpotsList`
- `ListedSpotInfo`
- `SpotTag`

MapPage (rys. 7.54 - 7.56) To główny **komponent** prezentujący mapę. Zawiera w sobie Map z **biblioteki react-maplibre**, w którym ustawia się domyślną lokalizację, na którą ustawiony jest obraz mapy oraz poziom przybliżenia. Przyjmuje on także inne **komponenty**, które mają być w nim wyświetcone. Ustalenie, które z nich mają być widoczne realizowane jest poprzez odczytanie wartości ze stanu **Redux'a**. Mapa ładowana jest z linków ustawionych w pliku `map_style.json`, w nim również opisany jest jej wygląd. **Hook useEffect** uruchamia się tylko przy pierwszej inicjalizacji **komponentu** i sprawdza czy w `url'u` umieszczone są parametry informujące, że link pochodzi z udostępnienia **spota**. Jeśli tak jest, odczytywane są odpowiednie dane, obraz mapy przekierowany jest na wskazaną lokalizację oraz otwierane są panel ze szczegółami **spota** (por. sekcja 7.3.6.2) i jego pogodą (por. sekcja 7.3.6.3).

```

export default function MapPage(): Element {
  Show usages ⌘stanoz *
  const dispatch: ThunkDispatch<{ account: AccountSliceProp... }> = useDispatchTyped();
  const [searchParams] = useSearchParams();
  const handleZoomEnd: (event: any) => void = (event: any): void => {
    Show usages ⌘stanoz
    dispatch(mapAction.setZoomLevel(event.target.getZoom()));
  };
}

const { isLoggedIn } = useSelectorTyped((state: { account: AccountSliceProps; notification... }): AccountSliceProps => state.account);

useEffect((): void => {
  const spotId: number = Number(searchParams.get(name: "spotId"));
  const longitude: number = Number(searchParams.get(name: "longitude"));
  const latitude: number = Number(searchParams.get(name: "latitude"));
  const region: string | null = searchParams.get(name: "region");
  const city: string | null = searchParams.get(name: "city");
  if (
    Number.isInteger(spotId) &&
    Number.isFinite(longitude) &&
    Number.isFinite(latitude) &&
    region &&
    region.trim().length > 0 &&
    city &&
    city.trim().length > 0
  ) {
    dispatch(spotDetailsModalAction.setSpotId(spotId));
    dispatch(
      spotWeatherActions.setSpotCoordinates({
        latitude,
        longitude,
        region,
        city,
      }),
    );
    dispatch(spotDetailsModalAction.handleShowModal());
    dispatch(spotWeatherActions.openBasicWeatherModal());
  }
}, []);

```

Rysunek 7.54: Implementacja komponentu MapPage (1/3)

```

const showSpotDetailsModal :boolean = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :boolean => state.spotDetails.showModal,
);
const showBasicSpotWeatherModal :boolean = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :boolean => state.spotWeather.showBasicWeather,
);
const showSearchedSpotsList :boolean = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :boolean => state.searchedSpotsListModal.showList,
);
const showCurrentViewSpotsList :boolean = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :boolean => state.currentViewSpotsListModal.showList,
);
const showDetailedSpotWeatherModal :boolean = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :boolean => state.spotWeather.showDetailedWeather,
);
const { showExpandedGallery } = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :ExpandedSpotMediaGalleryModals... => state.expandedSpotMediaGalleryModals,
);
const { isFullscreenSize } = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :ExpandedSpotMediaGalleryFullscreenSizeModal => state.expandedSpotMediaGalleryFullscreenSizeMode,
);
const { showAddMediaModal } = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :AddSpotMediaModalProps => state.spotAddMediaModal,
);
const { showAddSpotCommentModal } = useSelectorTyped(
|   (state :{account: AccountSliceProps; notification...} :AddSpotCommentModalInfoProps => state.addSpotCommentModal,
);

```

Rysunek 7.55: Implementacja komponentu MapPage (2/3)

```

    return (
      <Map
        initialViewState={{
          ...defaultPosition,
          zoom: 15,
        }}
        dragRotate={false}
        style={{
          position: "relative",
          width: "100vw",
          height: "100vh",
          overflow: "hidden",
        }}
        mapStyle="/map_style.json"
        attributionControl={false}
        onZoomEnd={handleZoomEnd}
      >
        <SpotsNameSearchBar />
        {showBasicSpotWeatherModal && <BasicSpotWeather />}
        <AnimatePresence>
          {showSpotDetailsModal && <SpotDetails key="spot-details" />}
          {showSearchedSpotsList && (<SearchedSpotsList key="searched-spots-list" />)}
          {showCurrentViewSpotsList && (<CurrentViewSpotsList key="current-view-spots-list" />)}
          {showDetailedSpotWeatherModal && (<DetailedSpotWeather key="detailed-spot-weather-modal" />)}
          {showExpandedGallery && (<ExpandedSpotMediaGallery key="expanded-spot-media-gallery" />)}
          {isFullscreenSize && (<FullscreenMediaModal key="expanded-spot-media-gallery-fullscreen-media-modal" />)}
          {isLogged && showAddMediaModal && (<SpotAddMediaModal key="spot-add-media-modal" />)}
          {isLogged && showAddSpotCommentModal && (<AddSpotCommentModal key="add-spot-comment-modal" />)}
        </AnimatePresence>
        <div className="absolute right-1 bottom-1 flex flex-col items-center space-y-2 sm:right-2 sm:bottom-2 xl:right-5 xl:bottom-5">
          <UserLocationPanel />
          <ZoomControlPanel />
        </div>
        <SearchCurrentViewButton />
        <Spots />
      </Map>
    );
  );
}

```

Rysunek 7.56: Implementacja komponentu MapPage (3/3)

Spots (rys. 7.57 - 7.61) Komponent ten odpowiedzialny jest za nałożenie **spotów** na mapę. Z **backendu** za pomocą **hook'a useQuery** pobierana jest ich lista, a następnie zgodnie z wartością zwróconą przez funkcję **shouldRenderMarker**, przyjmującą pole powierzchni elementu oraz poziom przybliżenia mapy, **spot** wyświetlany jest w formie punktu (**Marker**) lub wielokąta (**Source** oraz **Layer**). Wymienione **komponenty** pochodzą z **biblioteki react-maplibre**. Zawiera ona również **hook useMap**, który zwraca obecną instancję mapy. Pozwala to dodać do wyświetlanych elementów event-listener'ów obsługujących najechanie na dany element oraz funkcji wywoływanej po jego kliknięciu. Jest to funkcja **handleSpotClick**. Powoduje ustawnienie w stanie **Redux'a** danych **spota** oraz otworzenie panelu z jego szczegółami (por. sekcja 7.3.6.2) i pogodą (por. sekcja 7.3.6.3). Za pomocą **mutacji** zostaje również wysłane żądanie zwiększające liczbę wyświetleń elementu.

Informacja o błędzie podczas pobierania danych wyświetlana jest w komponentie [Notification](#).

```
export default function Spots(): Element { Show usages & stanoz +2 *
  const { name } = useSelectorTyped<{state:{account:AccountSliceProps; notification:NotificationSliceProps}}>();
  const { zoomLevel } = useSelectorTyped<{state:{account:AccountSliceProps; notification:NotificationSliceProps}}>();
  const dispatch: ThunkDispatch<{account:AccountSliceProps; notification:NotificationSliceProps}> = useDispatchTyped();
  const { current: map } = useMap();
  const location: Location<any> = useLocation();
  const [searchParams] = useSearchParams();
  const { mutateAsync } = useMutation({
    mutationKey: ["increase-spot-views-count"],
    mutationFn: increaseSpotViewsCount,
    onError: () => {},
  });
  const { data, error } = useQuery({
    queryFn: () => fetchFilteredSpots(name),
    queryKey: ["spots", "filter", name],
    refetchOnWindowFocus: false,
    staleTime: 1000 * 60 * 5,
    retry: false,
  });
  useEffect(() => {
    const axiosError = error as AxiosError<any>;
    if (axiosError?.response?.data) {
      const message: any = axiosError.response?.data?.message || axiosError.response?.data;
      dispatch(notificationAction.addError({message,}));
    }
  }, [dispatch, error]);
  useEffect(() => {
    if (!map) return;
    const handleMouseEnter: () => void = () => {
      Show usages & stanoz
      map.getCanvas().style.cursor = "pointer";
    };
    const handleMouseLeave: () => void = () => {
      Show usages & stanoz
      map.getCanvas().style.cursor = "";
    };
    if (data) {
      data?.forEach((spot: GeneralSpot) => {
        if (!shouldRenderMarker(spot.area, zoomLevel)) {
          map.getCanvas().style.cursor = "";
        }
      });
    }
  }, [data, map, zoomLevel]);
}
```

Rysunek 7.57: Implementacja komponentu Spots (1/5)

```

    const handler :() => void = () :void => Show usages & stanoz
      handleSpotClick(
        spot.id,
        spot.centerPoint,
        spot.region,
        spot.city,
      );
    const layerId :string = spot.id.toString();
    clickHandlers.set(spot.id, handler);
    map?.on( type: "click", layerId, handleMouseEnter);
    map?.on( type: "mouseenter", layerId, handleMouseLeave);
    map?.on( type: "mouseleave", layerId, handleMouseLeave);
  }
}

return () :void => {
  if (data) {
    data?.forEach((spot: GeneralSpot) :void => {
      if (!shouldRenderMarker(spot.area, zoomLevel)) {
        const handler :(() => void) | undefined = clickHandlers.get(spot.id);
        const layerId :string = spot.id.toString();
        if (handler) {
          map?.off( type: "click", layerId, handler);
          clickHandlers.delete(spot.id);
        }
        map?.off( type: "mouseenter", layerId, handleMouseEnter);
        map?.off( type: "mouseleave", layerId, handleMouseLeave);
        map.getCanvas().style.cursor = "";
      }
    });
  };
}, [map, data, zoomLevel]);
useEffect(() :void => {
  if (!map || !location.state) return;
  const { spotCoords } = location.state;

```

Rysunek 7.58: Implementacja komponentu Spots (2/5)

```

        if (spotCoords) {
            map.flyTo({
                center: [spotCoords.y, spotCoords.x],
                zoom: 16,
                speed: 1.5,
            });
        }
    }, [map, location]);
const handleSpotClick : (spotId: number, centerPoint: SpotCoordinatesDto, = ( Show usages  ↳ stanoz +1 *
    spotId: number,
    centerPoint: SpotCoordinatesDto,
    region: string,
    city: string,
): void => {
    dispatch(spotDetailsModalAction.setSpotId(spotId));
    dispatch(
        spotWeatherActions.setSpotCoordinates({
            latitude: centerPoint.x,
            longitude: centerPoint.y,
            region,
            city,
        }),
    );
    dispatch(spotDetailsModalAction.handleShowModal());
    dispatch(spotWeatherActions.openBasicWeatherModal());
    dispatch(currentViewSpotsListModalActions.closeCurrentViewSpotsListModal());
    dispatch(searchedSpotListModalAction.handleCloseList());
    mutateAsync(spotId);
};

useEffect(() : void => {
    const longitude : number = Number(searchParams.get( name: "longitude" ) ?? undefined);
    const latitude : number = Number(searchParams.get( name: "latitude" ) ?? undefined);
    if (Number.isFinite(longitude) && Number.isFinite(latitude)) {
        map?.flyTo({
            center: [longitude, latitude],
            zoom: 15,
            speed: 1.2,
        });
    }
});

```

Rysunek 7.59: Implementacja komponentu Spots (3/5)

```
        curve: 1.42,
        essential: true,
    });
}
}, []);
return (
    <>
    {data?.map((spot: GeneralSpot) :Element =>
        shouldRenderMarker(spot.area, zoomLevel) ? (
            <Marker
                key={spot.id}
                longitude={spot.centerPoint.y}
                latitude={spot.centerPoint.x}
                onClick={() :void =>
                    handleSpotClick(
                        spot.id,
                        spot.centerPoint,
                        spot.region,
                        spot.city,
                    )
                }
            >
                <MdLocationPin
                    className="text-spotLocationMarker cursor-pointer text-3xl"
                    key={spot.id}
                />
            </Marker>
        ) : (
            <Source
                key={spot.id}
                id={spot.id.toString()}
                type="geojson"
                data={createGeoJson(spot)}
            >

```

Rysunek 7.60: Implementacja komponentu Spots (4/5)

```

    <Layer
      id={spot.id.toString()}
      type="fill"
      paint={{
        "fill-color": spot.areaColor,
        "fill-opacity": 0.55,
      }}
    />
  );
}
);

```

Rysunek 7.61: Implementacja komponentu Spots (5/5)

`ZoomControlPanel` Komponent obsługujący ustawianie przybliżenia przybliżenia mapy. Zawiera dwa przyciski wyświetlane przez `ZoomControlButton`, przyjmujące funkcje, które na instancji mapy pobieranej z `hook'a useMap` ([biblioteka react-maplibre](#)), wywołują na niej odpowiednio wbudowane operacje `zoomIn()` lub `zoomOut()`.

`UserLocationPanel` Odpowiada za wyświetlenie na mapie aktualnej lokalizacji użytkownika. Po kliknięciu przycisku za pomocą

`navigator.geolocation.getCurrentPosition` pobiera jego aktualną pozycję, a następnie zapisuje ją w lokalnym stanie `komponentu` (`useState`). Na ich podstawie na mapie wyświetlany jest `Marker` z [biblioteki react-maplibre](#). Pochodzący z niej `hook useMap` zwraca aktualną instancję mapy, na której wywoływana jest funkcja `flyTo` przenosząca płynnie obraz na wskazane koordynaty. Informacje o błędach podczas operacji wyświetlane są w `Notification`.

`SearchCurrentViewButton` (rys. 7.62) Komponent będący przyciskiem, po kliknięciu którego pokazywana jest lista `spotów` znajdujących się w widocznym

obszarze mapy. Naciśnięcie wywołuje funkcję `handleClickSearchCurrentView`, która na obecnej instacji mapy zwróconej przez `hook useMap` ([biblioteka react-maplibre](#)) za pomocą wbudowanej funkcji `getBounds` pobiera dwa punkty wyznaczające granicę szerokości i długości geograficznej obecnie wyświetlanego fragmentu mapy. Następnie na `backend` wysyłane jest odpowiednie zapytanie i otwierany jest panel z listą znalezionych elementów.

```

export default function SearchCurrentViewButton():Element {
  const { map } = useMap();
  const { swing, swLat, neLat, name, sorting, ratingFrom } = useSelectorTyped<{state: (account: AccountSliceProps; notification: CurrentViewSpotParamsProps) => state.currentViewSpotsParams}>();
  const dispatch : ThunkDispatch<{account: AccountSliceProps; notification: CurrentViewSpotParamsProps} , void, void> = useDispatchTyped();
  const queryClient : QueryClient = useQueryClient();
  const handleClickSearchCurrentView: () => void = () => {
    const viewBounds: LngLatBounds | undefined = map?.getBounds();
    dispatch(
      currentViewSpotsActions.setParams({
        swLng: viewBounds?.sw.lng,
        swLat: viewBounds?.sw.lat,
        neLng: viewBounds?.ne.lng,
        neLat: viewBounds?.ne.lat,
      }),
    );
    dispatch(currentViewSpotsActions.clearCurrentViewSpots());
    queryClient.removeQueries([
      {queryKey: ["current-view-spots", swing, swLat, neLat, name, sorting, ratingFrom]},
    ]);
    dispatch(spotDetailsModalAction.handleCloseModal());
    dispatch(spotWeatherActions.closeAllWeatherModals());
    dispatch(searchedSpotsListModalAction.handleCloseList());
    dispatch(currentViewSpotsListModalActions.openCurrentViewSpotsListModal());
  };
  return (
    <button
      className="dark:bg-violetLight bg-fifth hover:bg-whiteSmoke text-violetBrightText dark:text-darkText hover:dark:bg-violetLighter absolute bottom-4 left-1/2 -translate-x-1/2 cursor-pointer rounded-3xl px-14 py-1.5 text-xl font-semibold drop-shadow-md dark:drop-shadow-none"
      onClick={handleClickSearchCurrentView}
    >Search this area</button>
  );
}

```

Rysunek 7.62: Implementacja komponentu SearchCurrentViewButton

`CurrentViewSpotsList` Wyświetla listę `spotów` znajdujących się w widocznym obszarze mapy. Poszczególne elementy wyświetlane są przez `ListedSpotInfo` (por. sekcja [7.3.6.1.1](#)). Lista przewijana jest przy użyciu mechanizmu `infinite scroll`, zrealizowanego przez `IntersectionObserver` nasłuchującego na element powiązany referencją oraz `hook'a useInfiniteQuery` z [biblioteki Tanstack Query](#). Podczas ładowania danych wyświetlany jest `LoadingSpinner`, a o wystąpieniu błędu informuje komunikat *Failed to load spots data..* W `komponencie CurrentViewSpotsFormsContainer` umieszczone są formularze do filtrowania i sortowania listy. Filtrowanie `spotów` po nazwie obsługuje `CurrentViewSpotsNameearchBar`. W trakcie wpisywania nazwy wyświetlana jest lista z podpowiedziami ułatwiająca uzupełnienie pola. Po kliknięciu przycisku do szukania lub wyborze podpowiedzi

wyświetlane są [spoty](#), których nazwa zawiera w sobie wpisaną frazę. Filtrowanie po ocenie „od”, tzn. ustawienia minimalnej wartości tejże, umożliwia [komponent RatingFromForm](#). Za pomocą [Rate](#) z [biblioteki antd](#) wybierana jest wartość oceny, a następnie lista wyników jest aktualizowana. Jeżeli żaden [spot](#) nie spełnia warunków ustawionych przez opisane filtry, wyświetlany jest komunikat *No spots match criteria!*. Sortowanie listy realizowane jest przez [komponent SpotsSortingForm](#) (por. sekcja [7.3.6.1.1](#)). Otworzenie i zamknięcie panelu animowane jest przy użyciu elementów [biblioteki motion](#).

[SpotsNameSearchBar](#) [Komponent](#) umożliwiający wyszukiwanie [spotów](#) po nazwie. Podczas wpisywania frazy z [backendu](#) za pomocą [hook'a useQuery](#) ([biblioteka Tanstack Query](#)) pobierane są nazwy wszystkich [spotów](#) zawierających podany tekst i wyświetlane jako lista z podpowiedziami. Nazwy pobierane są przy wykorzystaniu [debounce'owania](#) zrealizowanego za pomocą [hook'a useDebounce](#), co pozwala ograniczyć liczbę wysyłanych zapytań. Po kliknięciu przycisku wyszukiwania lub wybraniu podpowiedzi wyświetlany jest panel z listą wyników ([SearchedSpotsList](#)). Kliknięcie ikony [IoClose](#) powoduje usunięcie filtra.

[SearchedSpotsList](#) Wyświetla wynik wyszukiwania [spotów](#) po nazwie. Lista jest przewijalna z wykorzystaniem mechanizmu [infinite scroll](#) zrealizowanego za pomocą [IntersectionObserver](#), [hook'a useInfiniteQuery](#) ([biblioteka Tanstack Query](#)) oraz referencji. Podczas pobierania danych prezentowany jest [LoadingSpinner](#), a w wypadku błędu komunikat *Failed to load searched spots data..* Jeżeli lista wyników jest pusta, pojawia się informacja *No spots match criteria!*. Elementy można sortować poprzez [komponent SpotsSortingForm](#) (por. sekcja [7.3.6.1.1](#)). Otworzenie i zamknięcie panelu animowane jest za pomocą elementów [biblioteki motion](#).

7.3.6.1.1 Komponenty wspólne

W niniejszym rozdziale opisano [komponenty](#), z których korzystają inne [komponenty](#) zawarte w tej części.

[ListedSpotInfo](#) Wyświetla pojedynczego [spota](#) będącego wynikiem wyszukiwania po nazwie lub w widocznym obszarze mapy. Zawiera jego pierwsze zdjęcie, nazwę, ocenę w gwiazdkach, liczbę ocen oraz tagi. Ocena zrealizowana jest za pomocą [Rate](#) z [biblioteki antd](#), a tagi prezentowane są przez listę [SpotTag](#).

Po kliknięciu elementu na obecnej instancji mapy zwracanej przez `hook useState` (`react-maplibre`) wywoływana jest funkcja `flyTo`, przenosząca w płynnym przejściu obraz mapy na lokalizację `spota`.

`SpotsSortingForm Komponent` realizujący ustawianie sortowania listy `spotów`. Do wyboru dostępne są opcje: Default, Rating ascending, Rating descending, Rating count ascending oraz Rating count descending, zapisane jako tablica wartości w stałej `options`. Otworzenie i zamknięcie listy z tymi elementami animowane jest przy użyciu `komponentu AnimatePresence` z `biblioteki framer-motion`. W `props` przyjmowane są funkcje obsługujące zmianę wyboru opcji oraz czyszczenia poprzednich wyników. Przekazywana jest również tablica określająca klucz zapytań wymagających unieważnienia po zmianie sortowania, a także opcja sortowania wybrana w danej liście. W celu rozróżnienia gdzie używany jest `komponent` przez `propa variant` określany jest jego typ (SEARCH lub CURRENT_VIEW). Wybrana opcja przechowywana jest w lokalnym stanie (`useState`), `hook useEffect` nasłuchuje na jej zmianę jeśli jest różna od opcji wybranej wcześniej (`prop sorting`), wykonuje przekazaną funkcję do obsługi zmiany wyboru sortowania oraz funkcję czyszczącą. Unieważnianie są również zapytania zawierające podany klucz. `Hook useMemo` zwraca stałą będącą „stabilnym kluczem” – po każdym odświeżeniu `komponentu` nadziedziczonego tablica określająca klucz z zapytaniami wymagającymi unieważnienia jest przekazywana ponownie. `useMemo` porównuje jej wartości z poprzednią i jeśli są takie same, nie zwraca nowej wartości, która jest w tablicy dependencji `useEffect` wspomnianego wcześniej. Pozwala to na optymalizację poprzez uniknięcie niepotrzebnych uruchomień tego `hook'a`.

7.3.6.2 Panel ze szczegółami spota

Panel ze szczegółami `spota` zbudowany jest z czterech głównych części:

- informacji o `spocie`
- galerii zdjęć i filmów
- przycisków akcji
- komentarzy `spota`

SpotDetails (rys. 7.63 - 7.65) To główny komponent, który ustawia ułożenie wyżej wymienionych elementów. Odpowiedzialny jest za pobranie danych o wybranym spotie. W trakcie ich ładowania wyświetlany jest **LoadingSpinner**, a informacja o błędzie w **Notification**. Przejścia otwarcia oraz zamknięcia panelu animowane są za pomocą biblioteki **motion**.

```
export default function SpotDetails(): Element { Show usages & stanoz +2 *  
  const spotId: number | null = useSelectorTyped<(state: AccountSliceProps; notificationAction: ThunkDispatch<AccountSliceProps>) : number | null> => state.spotDetails.spotId;  
  const isSidebarOpen: boolean = useSelectorTyped<(state: AccountSliceProps; notificationAction: ThunkDispatch<AccountSliceProps>) : boolean> => state.sidebar.isOpen;  
  const dispatch: ThunkDispatch<AccountSliceProps> = useDispatchTyped();  
  const { data, error, isLoading } = useQuery({  
    queryFn: () : Promise<SpotDetails> => fetchSpotsDataById(spotId),  
    queryKey: ["spotDetails", spotId],  
    refetchOnWindowFocus: false,  
    staleTime: 1000 * 60 * 5,  
  });  
  useEffect(() : void => {  
    const axiosError = error as AxiosError<any>;  
    if (axiosError?.response?.data) {  
      const message: any =  
        axiosError.response?.data?.message || axiosError.response?.data;  
      dispatch(  
        notificationAction.addError({ message }),  
      );  
    }  
  }, [dispatch, error]);  
  const handleClickCloseModal: () => void = () : void => { Show usages & stanoz  
    dispatch(spotDetailsModalAction.handleCloseModal());  
    dispatch(spotWeatherActions.closeAllWeatherModals());  
  };  
  return (  
    <>  
      <div  
        className={`absolute top-0 z-[5] h-screen w-screen bg-black/85 transition-opacity duration-300 ${  
          isSidebarOpen  
          ? "opacity-100"  
        }>
```

Rysunek 7.63: Implementacja komponentu SpotDetails (1/3)

```
    : "pointer-events-none opacity-0"
  }`}
></div>
<motion.div
  key={spotId}
  initial="hidden"
  animate="visible"
  exit="exit"
  variants={slideVariants}
  transition={{ duration: 0.3 }}
  className="█ dark:bg-violetDarker █ bg-fifth █ text-violetDark
  █ dark:text-darkText 3xl:w-[35rem] 3xl:overflow-y-hidden
  absolute top-10 left-0 z-[2] flex h-full w-[20rem]
  overflow-y-auto p-2 text-lg drop-shadow-md xl:top-0 xl:left-17
  xl:w-[30rem] xl:text-xl dark:drop-shadow-none"
>
  {isLoading && <LoadingSpinner />}
  {data && (
    <div className="mx-3 flex h-full w-full flex-col">
      <div className="mt-3 flex items-center justify-between text-xl">
        <div className="flex justify-start">
          <MdLocationPin className="mr-0.5 text-2xl █ text-red-600" />
          <SpotAddressInfo
            country={data.country}
            city={data.city}
            street={data.street}
          />
        </div>
      </div>
    </div>
  )}
```

Rysunek 7.64: Implementacja komponentu SpotDetails (2/3)

```

        <HiX
          className="text-violetLight dark:text-darkText cursor-pointer text-2xl xl:text-xl"
          onClick={handleClickCloseModal}
        />
      </div>
      <SpotGeneralInfo
        name={data.name}
        description={data.description}
        rating={data.rating}
        ratingCount={data.ratingCount}
        tags={data.tags}
      />
      <SpotDetailsGallery media={data.media} />
      <SpotActionButtonsContainer
        spotId={spotId!}
        centerPoint={data.centerPoint}
      />
      <SpotCommentsList
        spotId={data.id}
        spotName={data.name}
      />
    )}
  </motion.div>
);
}

```

Rysunek 7.65: Implementacja komponentu SpotDetails (3/3)

7.3.6.2.1 Informacje o spotie

Sekcja ta składa się z trzech komponentów:

- `SpotAddressInfo`
- `SpotGeneralInfo`
- `SpotTag`

`SpotAddressInfo` Wyświetla informacje o lokalizacji `spota`: państwo, miasto oraz nazwę ulicy.

`SpotGeneralInfo` (rys. 7.66 i 7.67) Komponent odpowiedzialny za wyświetlanie danych takich jak: nazwa `spota`, opis, ocena, liczba ocen, lista tagów, które przyjmuje w `propsach`. Do prezentacji oceny w formie gwiazdek użyto `Rate` z biblioteki `antd`, a jego konfiguracja ustawiana jest w `ConfigProvider`. Responsywność tego komponentu realizowana jest poprzez ustawianie wielkości gwiazdek. Własny

`hook` `useScreenSize` podaje aktualne rozmiary ekranu, a `useEffect` reaguje na zmiany i za pomocą funkcji `calculateRateStarSize` ustawia odpowiednią wartość przechowywaną w lokalnym stanie `komponentu`. Tagi wyświetlane są przez listę `SpotTag`.

```
export default function SpotGeneralInfo({ name, description, rating, ratingCount, tags, Show usages ⌘ stanoz +1* }: SpotGeneralInfoProps) :Element {
  const { height, width } = useScreenSize();
  const [rateStarSize, setRateStarSize] = useState<number>(
    calculateRateStarSize(width, height),
  );
  useEffect(() :void => {
    setRateStarSize(calculateRateStarSize(width, height));
  }, [height, width]);
  return (
    <div className="mt-5 mb-2 flex-col space-y-6">
      <div className="flex flex-col items-center space-y-2 xl:flex-row xl:justify-between">
        <p className="text-violetBrightText dark:text-darkText 3xl:text-2xl 2xl:text-xl">
          {name}
        </p>
        <div className="flex justify-end space-x-2">
          <div className="custom-rate mx-2 mt-1 inline-flex min-w-fit">
            <ConfigProvider theme={{ components: { Rate: { starBg: "#ffffff", starColor: "var(--color-ratingStarColor)", starSize: rateStarSize, starHoverScale: "scale(1,1)" } } }}>
```

Rysunek 7.66: Implementacja komponentu `SpotGeneralInfo` (1/2)

```

        <Rate
          data-testid="spot-rating"
          disabled
          allowHalf
          value={rating}
        />
      </ConfigProvider>
    </div>
    <p className="text-xl 2xl:text-2xl">({ratingCount})</p>
  </div>
</div>
<ul className="flex flex-wrap space-x-4">
  {tags.map((tag : TagDto ) : Element  => (
    <li className="my-1" key={tag.id}>
      <SpotTag name={tag.name} />
    </li>
  )))
</ul>
<div className="text-violetDarkText dark:text-darkText flex-col space-y-0.5">
  <p>Description:</p>
  <p>{description}</p>
</div>
</div>
);
}
}

```

Rysunek 7.67: Implementacja komponentu SpotGeneralInfo (2/2)

7.3.6.2.2 Galeria zdjęć i filmów

Zbudowana jest z następujących komponentów:

- `SpotDetailsGallery`
- `Photo`
- `Video`

`SpotDetailsGallery` (rys. 7.68 i 7.69) Komponent odpowiedzialny za wyświetlanie galerii zdjęć i filmów spota. Galeria w formie karuzeli została zrealizowana poprzez `Carousel`, który jest elementem biblioteki `antd`. Poprzez komponent konfiguracyjny (`ConfigProvider`) ustawiane jest położenie strzałek przełączających media. Za wyświetlanie odpowiednio zdjęć i filmów odpowiadają `Photo` i `Video`.

W lokalnym stanie (`useState`) ustawiany jest obecny indeks elementu, poprzez porównanie go z indeksem właściwym elementu ustalana jest jedna ze składowych określających czy video powinno być odtworzone. Pozwala to na automatyczne zatrzymanie filmu po zmianie media.

```
export default function SpotDetailsGallery({ media }: PhotoGalleryProps) : Element { Show usages & stanoz
  const [current, setCurrent] = useState<number>({ initialState: 0 });

  return (
    <div className="my-10 flex h-full items-center justify-center">
      {media && media.length > 0 ? (
        <ConfigProvider
          theme={{{
            components: {
              Carousel: {
                arrowOffset: -25,
                dotOffset: -20,
              },
            },
          }}>
        <Carousel
          arrows={true}
          className="carousel-rounded 3xl:w-[30rem] w-[15rem] 2xl:w-[25rem]"
          afterChange={(idx: number) : void => {
            setCurrent(idx);
          }}
        >
          {media.map((media: SpotMediaDto, idx: number) : Element =>
            media.genericMediaType === MediaType.VIDEO ? (
              <Video
                key={media.id}
                video={media}
                shouldPlayVideo={idx === current}
              />
            ) : (

```

Rysunek 7.68: Implementacja komponentu SpotDetailsGallery (1/2)

```

        <Photo key={media.id} photo={media} />
      ),
    )
  )
</Carousel>
</ConfigProvider>
) : (
  <div className="my-1">
    <p className="text-center text-lg">
      There are no media for this spot.
    </p>
  </div>
)
</div>
);
}

```

Rysunek 7.69: Implementacja komponentu SpotDetailsGallery (2/2)

Photo Odpowiada za wyświetlenie zdjęcia oraz obsługę jego kliknięcia. Funkcja `handleClickPhoto` poprzez **mutację** wysyła na **backend** żądanie ustalające pozycję zdjęcia w liście wszystkich zdjęć `spota` z uwzględnieniem sortowania. **Hook** `useEffect` reaguje na zmianę danych i ustawia w stanie `Redux'a` id zdjęcia, jego pozycję w całej liście, typ media na *PHOTO* oraz otwiera **modal ExpandedSpotMediaGallery** (por. sekcja 7.3.6.4).

Video Obsługuje wyświetlanie oraz odtwarzanie filmów. Operacja zrealizowana jest za pomocą **komponentu** `ReactPlayer` z **biblioteki** `react-player`. Panel sterujący filmem (czas trwania wraz z możliwością przesuwania, powiększenie, ustawienie głośności, przyciski do odtwarzania i puzowania) zbudowany został z elementów **biblioteki** `media-chrome`.

7.3.6.2.3 Przyciski akcji

Sekcja składa się z trzech **komponentów**:

- `SpotActionButtonsContainer`
- `SpotActionButton`

- `SpotAddMediaModal`

`SpotActionButtonsContainer` Zawiera w sobie logikę obsługującą akcje możliwe do wykonania na danym `spocie`. Przyciski wyświetlane są poprzez `SpotActionButton`, którym poszczególne funkcje przekazywane są przez `propsy`. Nawigacja do wybranego `spota` zrealizowana jest poprzez pobranie lokalizacji użytkownika (`navigator.geolocation.getCurrentPosition`), a następnie przeniesienie na `google maps` z wyznaczoną trasą. Funkcja `clickShareSpotHandler` obsługuje udostępnienie `spota` przez skopiowanie odpowiedniego linka do pamięci podręcznej systemu korzystając z `navigator.clipboard`. Dane zapisane w `query params` pozwalają po wklejeniu linka na otworzenie panelu ze szczegółami `spota` oraz przybliżenie jego lokalizacji na mapie. Przycisk z ikoną `MdOutlineAddPhotoAlternate` sprawdza czy użytkownik jest zalogowany i w `komponencie Notification` wyświetla odpowiedni komunikat. Gdy ten warunek jest spełniony, otwierany jest `modal` `SpotAddMediaModal`, zwierający formularz do dodawania media do `spota`. Poprzez `hook useQuery` wysyłane jest zapytanie sprawdzające czy wybrany `spot` znajduje się w liście polubionych użytkownika. Od wyniku operacji uzależniony jest wygląd przycisku oraz typ wysyłanego żądania po jego kliknięciu (wysyłanego `mutacją`). Wynik operacji dodania lub usunięcia `spota` z listy ulubionych pokazywany jest w `komponencie Notification`. Zapytanie dotyczące listy ulubionych `spotów` użytkownika jest unieważniane w celu wymuszenia ponownego pobrania danych i ich aktualizacji.

`SpotAddMediaModal` `Komponent` zawierający formularz do dodawania mediów do `spota`. Pozwala na wybór plików (zdjęć oraz filmów), które wraz z podglądem wyświetlane są w formie listy. Kliknięcie elementu na liście powoduje jego usunięcie. Dane przysłane są `mutacją` na `backend`, a zapytanie dotyczące szczegółów `spota` są unieważnianie, dzięki czemu dodane media będą znajdować się w galerii zdjęć i filmów. Komunikat o wyniku operacji wyświetlany jest w `Notification`. Otworzenie oraz zamknięcie elementu animowane jest `biblioteką motion`.

7.3.6.2.4 Komentarze spota

Część panelu szczegółów `spota`, w której znajdują się zarówno komentarze, jak i `modal` do ich dodawania. Zbudowana jest z następujących `komponentów`:

- `SpotCommentsList`
- `SpotCommentHeader`
- `AddSpotCommentModal`
- `SpotComment`
- `SpotCommentAuthor`
- `SpotCommentMediaGallery`
- `SpotCommentVotesPanel`
- `SpotCommentVoteDisplay`

`SpotCommentsList` Komponent wyświetlający listę komentarzy przewijalnej przy użyciu `infinite scroll`. Mechanizm ten jest realizowany za pomocą `IntersectionObserver`, który nasłuchuje na element powiązany referencją i wywołuje funkcję `fetchNextPage` z `hook'a` `useInfiniteQuery` ([biblioteka tanstack query](#)). Na początku pobierana jest tylko pierwsza strona komentarzy oraz wyświetlany jest przycisk `Show More`. Po jego kliknięciu jest ukrywany i wysyłane jest zapytanie o drugą stronę. Pobieranie kolejnych odbywa się w sposób opisany powyżej. W trakcie ładowania wyświetlany jest `LoadingSpinner`, a w razie błędu komunikat *Failed to load comments..*

`SpotCommentHeader` Zawiera w sobie nagłówek „Comments” oraz przycisk otwierający `modal` będący formularzem do dodania nowego komentarza (`AddSpotCommentModal`). Po jego kliknięciu jeżeli użytkownik nie jest zalogowany, w [komponencie Notification](#) wyświetlany jest komunikat z odpowiednią informacją.

`AddSpotCommentModal` (rys. 7.70 - 7.73) `Modal` będący formularzem do dodania nowego komentarza. Zawiera w sobie pola do określenia oceny w gwiazdkach, dodania tekstu oraz zdjęć i filmów. Ustawienie oceny `spota` jest polem obowiązkowym z domyślną wartością 0. Zrealizowane jest przy użyciu [komponentu Rate skonfigurowanego przez ConfigProvider](#). Oba elementy pochodzą z [biblioteki antd](#). Innym obowiązkowym polem jest pole tekstowe, w którym użytkownik umieszcza swoją opinię. Wartości wpisane przechodzą proces walidacji przez

`addSpotCommentSchema` i funkcje biblioteki zod. Informacje o błędach wyświetlane są w `Notification`. Dodawanie zdjęć oraz filmów jest opcjonalne i zaimplementowane zostało za pomocą komponentu `UploadButton`, który otwiera eksplorator plików i umożliwia wybranie odpowiednich plików. Wskazane elementy wyświetlane są w formie listy, a ich kliknięcie usuwa je z niej. Możliwe jest dodanie maksymalnie dwudziestu elementów, o czym użytkownik jest informowany odpowiednim komunikatem. Wartości wpisane w poszczególne pola formularza przechowywane są w lokalnym stanie komponentu. Wysłanie danych do backendu odbywa się poprzez `mutację`, a w oczekiwaniu na odpowiedź wyświetlany jest `LoadingSpinner`. Pomyślne zakończenie operacji skutkuje unieważnieniem zapytań powiązanych z danym spotem oraz zamknięciem modalu. Kliknięcie przycisku „Cancel” powoduje wywołanie funkcji `handleCancelAddSpotComment`, która zamyka formularz, wpisane dane nie zostają zapisane.

```
export default function AddSpotCommentModal(): Element { Show usages & stanoz *
  const [spotRating, setSpotRating] = useState<number>(initialState: 0);
  const [mediaFiles, setMediaFiles] = useState<File[]>([[]]);
  const [commentText, setCommentText] = useState<string>(initialState: "");
  const { spotName } = useSelectorTyped<{ state: { account: AccountSliceProps; notification: any } }>(
    state: AddSpotCommentModalInfoProps
  ) => state.addSpotCommentModal;
  const { spotId } = useSelectorTyped<{ state: { account: AccountSliceProps; notification: any } }>(
    state: AddSpotCommentModalInfoProps
  ) => state.spotDetails;
  const dispatch: ThunkDispatch<{ account: AccountSliceProp... }> = useDispatchTyped();
  const handleCancelAddSpotComment: () => void = () => {
    Show usages & stanoz
    setSpotRating( value: 0 );
    setCommentText( value: "" );
    setMediaFiles( [] );
    dispatch( addSpotCommentModalInfoActions.closeAddSpotCommentModal() );
  };
  const handleFileSelect: (files: File[]) => void = (files: File[]) => {
    Show usages & stanoz
    setMediaFiles( files );
  };
  const queryClient: QueryClient = useQueryClient();
  const { mutateAsync, isPending } = useMutation<{
    mutationKey: ["addSpotComment", spotId],
    mutationFn: addSpotComment,
    onSuccess: async (): Promise<void> => {
      dispatch(
        notificationAction.addSuccess({
          message: "Spot comment added successfully!",
        })
      );
      await queryClient.invalidateQueries({
        queryKey: ["spot-comments", spotId],
      });
      await queryClient.invalidateQueries({
        queryKey: ["spotDetails", spotId],
      });
      setSpotRating( value: 0 );
      setCommentText( value: "" );
      setMediaFiles( [] );
      dispatch( addSpotCommentModalInfoActions.closeAddSpotCommentModal() );
    },
  },
```

Rysunek 7.70: Implementacja komponentu `AddSpotCommentModal` (1/4)

```

    onError: () : void => {
      dispatch(
        notificationAction.addError({
          message: "Failed to add spot comment.",
        }),
      );
    },
  );
}

const handleAddSpotComment: () => Promise<void> = async () : Promise<void> => { Show usages ⌂ stanoz *
  if (mediaFiles.length > 20) {
    dispatch(
      notificationAction.addError({
        message: "Maximum amount of media is 20.",
      }),
    );
    return;
  }
  const validationResult: ZodSafeParseResult<{ commentText: string;... }> = addSpotCommentSchema.safeParse({
    commentText,
    spotRating,
  });
  if (!validationResult.success) {
    validationResult.error.issues.forEach((i: $ZodIssue) : void => {
      dispatch(notificationAction.addError({ message: i.message }));
    });
    return;
  }
  await mutateAsync({
    text: commentText,
    spotId: spotId!,
    mediaFiles: mediaFiles,
    rating: spotRating,
  });
};

return (
  <motion.div
    initial="hidden"
    animate="visible"

```

Rysunek 7.71: Implementacja komponentu AddSpotCommentModal (2/4)

```
exit="exit"
variants={slideVariants}
transition={{ duration: 0.3 }}
className="absolute top-0 z-[4] flex h-full w-full items-center justify-center bg-black/80"

> <div className="dark:bg-darkBgSoft bg-fifth dark:text-darkText text-violetDark flex w-[40rem]
flex-col items-center space-y-5 rounded-2xl p-8">
  <h1 className="mb-4 text-3xl">{spotName}</h1>
  <ConfigProvider
    theme={{
      components: {
        Rate: {
          starBg: "#ffffff",
          starColor: "#fadbd4",
          starSize: 50,
          starHoverScale: "scale(1,1)",
        },
      },
    }}
  >
    <Rate
      allowHalf
      value={spotRating}
      onChange={(value: number) : void => setSpotRating(value)}
    />
  </ConfigProvider>
  <div className="dark:bg-darkBg mt-5 flex w-full flex-col items-center rounded-md bg-white px-4 py-1.5">
    <div className="h-96 max-w-56">
      <UploadButton onFileSelect={handleFileSelect} />
    </div>
    <hr className="w-full" />
    <textarea
      className="dark:placeholder-darkText placeholder-violetDark h-40 w-full text-sm focus:outline-0"
      placeholder="Type here..."
      value={commentText}
      onChange={(e : ChangeEvent<HTMLTextAreaElement>) : void => setCommentText(e.target.value)}
    />
  </div>

```

Rysunek 7.72: Implementacja komponentu AddSpotCommentModal (3/4)

```

        </div>
      <div className="text-darkText flex h-28 items-center justify-center space-x-5 text-xl">
        <button
          disabled={isPending}
          onClick={handleCancelAddSpotComment}
          className="bg-violetDarker hover:bg-violetDark border-violetLightDarker
          dark:hover:bg-violetLightDarker cursor-pointer rounded-xl border px-3 py-1.5"
        >
          Cancel
        </button>
        {isPending ? (
          <LoadingSpinner />
        ) : (
          <button
            disabled={isPending}
            onClick={handleAddSpotComment}
            className="bg-violetBrighter hover:bg-violetBright cursor-pointer rounded-xl px-3 py-1.5"
          >
            Publish
          </button>
        )}
      </div>
    </div>
  );
}

```

Rysunek 7.73: Implementacja komponentu AddSpotCommentModal (4/4)

SpotComment Komponent wyświetlający pojedynczy komentarz. Zbudowany jest z wyspecjalizowanych komponentów odpowiedzialnych za prezentację jego poszczególnych fragmentów. **SpotCommentAuthor** zawiera informacje o autorze komentarza: nazwę oraz zdjęcie profilowe, kliknięcie go powoduje przeniesienie na stronę profilu danego użytkownika. Ocena komentarza zrealizowana jest poprzez **Rate** z biblioteki **antd**. Pozostałe dwa **komponenty**, **SpotCommentMediaGallery** oraz **SpotCommentVotesPanel** zostały opisane poniżej.

SpotCommentMediaGallery Wyświetla zdjęcia i filmy dodane do komentarza. Pobieranie danych zrealizowane jest za pomocą **hook'a useQuery** z **biblioteki tanstack query**. W czasie ich ładowania prezentowany jest **komponent LoadingSpinner**, a o niepowodzeniu operacji informuje komunikat *Failed to download rest of the photos!*. Pierwsze trzy elementy są widoczne od razu, jeżeli ich ogólna liczba jest większa (informacja o tym jest przesyłana z **backendu**), to na trzeci z nich nakładany jest częściowo przezroczysty przycisk z napisem „See more”. Po jego kliknięciu funkcja **handleShowMoreMedia** powoduje wysłanie zapytania i wyświetlenie wszystkich

multimedów. Wybór elementu z listy powoduje wysłanie żądanie za pomocą **mutacji** ustalającej jego położenie pośród wszystkich multimedów **spota**. Na wynik operacji nasłuchuje **hook useEffect**, który po otrzymaniu danych w stanie **Redux** ustawia ID elementu, jego pozycję w całej liście oraz typ (zdjęcie lub film). Następnie otwiera dużą galerię multimedów **spota** (por. sekcja [7.3.6.4](#)).

SpotCommentVotesPanel (rys. [7.74 - 7.76](#)) **Komponent** odpowiedzialny za wyświetlanie polubień i niepolubień komentarza oraz obsługę operacji oddania głosu przez użytkownika. Za pomocą **hook'a useQuery** (**biblioteka tanstack query**) pobierana jest informacja czy i jaki głos oddał użytkownik. Na jej podstawie ustawiany jest wygląd odpowiedniej ikony. Kliknięcie przycisku powoduje następujące operacje:

- jeśli użytkownik polubił komentarz, ponowne kliknięcie przycisku usuwa je; analogicznie jest z niepolubieniami
- jeśli użytkownik kliką przycisk przeciwny do wcześniejszego, poprzednia operacja jest cofana na rzecz wykonania nowej
- jeśli użytkownik nie oddał wcześniej głosu na komentarz, wykonywana jest wybrana akcja

Zagłosowanie wymaga zalogowania, o czym informuje komunikat w **komponentie Notification**. W powyższych operacjach zastosowano optymistyczne aktualizowane **UI** (por. sekcja [7.1.2.2](#)) poprzez przechowywanie liczby głosów w stanie lokalnym (**hook useState**) i zwiększanie lub pomniejszanie jej w zależności od wykonanej akcji. Żądanie na **backend** wysyłane jest za pomocą **mutacji**. Po pozytywnej odpowiedzi unieważniane są wszystkie zapytania powiązane z danym **spotem** i komentarzem, co gwarantuje pobranie aktualnych danych. Natomiast jeśli wystąpi błąd, informacja o tym wyświetlana jest w **Notification**. **Komponent SpotCommentVoteDisplay** wyświetla liczbę oddanych głosów wraz z odpowiednią ikoną.

```

export default function SpotCommentVotesPanel({ Show usages stanoz *
  upvotes,
  downvotes,
  commentId,
}: SpotCommentVotesProps) :Element {
  const [voteType, setVoteType] = useState<SpotCommentVoteType>(
    |  SpotCommentVoteType.NONE,
  );
  const [voteCounter, setVoteCounter] = useState<VoteCounterType>({
    upVotes: upvotes,
    downVotes: downvotes,
  });
  const { isLoggedIn } = useSelectorTyped<{state : {account: AccountSliceProps; notification: NotificationSliceProps}}>();
  const { spotId } = useSelectorTyped<{state : {account: AccountSliceProps; notification: NotificationSliceProps}}>();
  const dispatch : ThunkDispatch<{account: AccountSliceProps; notification: NotificationSliceProps}> = useDispatchTyped();
  const queryClient : QueryClient = useQueryClient();
  const { data, isSuccess } = useQuery({
    queryKey: ["get-spot-comment-vote-info", commentId],
    queryFn: () :Promise<SpotCommentVoteInfoDto> => getSpotCommentVoteInfo(commentId),
    enabled: isLoggedIn,
  });
  const { mutateAsync } = useMutation({
    mutationKey: ["vote-comment", commentId],
    mutationFn: voteComment,
    onSuccess: async () :Promise<void> => {
      await queryClient.invalidateQueries({ queryKey: ["get-spot-comment-vote-info", commentId], });
      await queryClient.invalidateQueries({ queryKey: ["spot-comments", spotId], });
    },
    onError: () :void => {
      dispatch(notificationAction.addError({ message: "Failed to vote comment.", }));
    },
  });
  const handleUpVoteComment :()=> Promise<void> = async () :Promise<void> => {
    Show usages stanoz *
    if (!isLoggedIn) {
      dispatch(notificationAction.addInfo({ message: "Please sign in to vote comment.", }));
    } else {
  
```

Rysunek 7.74: Implementacja komponentu SpotCommentVotesPanel (1/3)

```

        setVoteCounter(({prevState : VoteCounterType} : { upVotes: number; downVotes: number } ) => ({
          upVotes:
            voteType === SpotCommentVoteType.UP_VOTE
              ? prevState.upVotes - 1
              : prevState.upVotes + 1,
          downVotes:
            voteType === SpotCommentVoteType.DOWN_VOTE
              ? prevState.downVotes - 1
              : prevState.downVotes,
        }));
        await mutateAsync({ commentId, isUpvote: true });
      };
    );
  const handleDownVoteComment :() => Promise<void> = async () :Promise<void> => { Show usages ⌘stanoz *
    if (!isLogged) {
      dispatch(notificationAction.addInfo({ message: "Please sign in to vote comment.", }));
    } else {
      setVoteCounter(({prevState : VoteCounterType} : { upVotes: number; downVotes: number } ) => ({
        upVotes:
          voteType === SpotCommentVoteType.UP_VOTE
            ? prevState.upVotes - 1
            : prevState.upVotes,
        downVotes:
          voteType === SpotCommentVoteType.DOWN_VOTE
            ? prevState.downVotes - 1
            : prevState.downVotes + 1,
      }));
      await mutateAsync({ commentId, isUpvote: false });
    }
  };
  useEffect(() :void => {
    if (isSuccess && data) {
      setVoteType(data.voteInfo);
    }
  }, [data, isSuccess]);

```

Rysunek 7.75: Implementacja komponentu SpotCommentVotesPanel (2/3)

```

        return (
            <div className="mt-3 flex w-full justify-start space-x-3">
                <SpotCommentVoteDisplay votes={voteCounter.upVotes}>
                    <BiLike
                        className={`${baseIconClass} ${{
                            voteType === SpotCommentVoteType.UP_VOTE
                            ? 'hidden' : 'visible'
                        }}`}
                        onClick={handleUpVoteComment}
                    />
                    <BiSolidLike
                        className={`${baseIconClass} ${{
                            voteType === SpotCommentVoteType.UP_VOTE
                            ? 'visible' : 'hidden'
                        }}`}
                        onClick={handleUpVoteComment}
                    />
                </SpotCommentVoteDisplay>
                <SpotCommentVoteDisplay votes={voteCounter.downVotes}>
                    <BiDislike
                        className={`${baseIconClass} ${{
                            voteType === SpotCommentVoteType.DOWN_VOTE
                            ? 'hidden' : 'visible'
                        }}`}
                        onClick={handleDownVoteComment}
                    />
                    <BiSolidDislike
                        className={`${baseIconClass} ${{
                            voteType === SpotCommentVoteType.DOWN_VOTE
                            ? 'visible' : 'hidden'
                        }}`}
                        onClick={handleDownVoteComment}
                    />
                </SpotCommentVoteDisplay>
            </div>
        );
    }
}

```

7.3.6.3 Panel z informacjami pogodowymi

Panel ten jest odpowiedzialny za wyświetlanie informacji pogodowych wybranego [spota](#). Składa się z dwóch głównych części:

- panelu z ogólnymi danymi takimi jak ikona stanu pogody, temperatura, prędkość wiatru
- zakładki zawierającej szczegółowe informacje: aktualny czas, ikonę stanu pogody, prawdopodobieństwo opadów, wskaźnik UV, punkt rosy, wilgotność, prędkość wiatru na różnych wysokościach, wykres z prognozowaną pogodą na trzy dni

7.3.6.3.1 Panel z danymi ogólnymi

Składa się z dwóch postawowych [komponentów](#):

- `BasicSpotWeather`
- `WeatherIcon`

`BasicSpotWeather` (rys. [7.77](#) i [7.78](#)) [Komponent](#) jest odpowiedzialny za wyświetlanie następujących informacji pogodowych: temperatura, prędkość wiatru, także ikony obrazującej obecny stan warunków atmosferycznych (komponent `WeatherIcon` (por. [sekcja 7.3.6.3.3](#))). Dane pobierane są z odpowiedniego [endpointu backendu](#). W czasie ich ładowania wyświetlany jest [komponent LoadingSpinner](#). Jeśli operacja zakończy się niepowodzeniem, użytkownikowi wyświetlany jest komunikat *Failed to load weather for this spot!*. Przycisk *Show more* powoduje zamknięcie obecnego panelu i otworzenie zakładki ze szczegółowymi informacjami (komponent `DetailedSpotWeather`)

```
export default function BasicSpotWeather(): Element | undefined { Show usages ⌘stanoz *
  const { latitude, longitude } = useSelectorTyped(
    (state: { account: AccountSliceProps; notification: any }): SpotWeatherSliceProps => state.spotWeather,
  );

  const { data, isLoading, isError } = useQuery({
    queryFn: () : Promise<SpotBasicWeatherDto> => getBasicSpotWeather(latitude, longitude),
    queryKey: ["spot", "weather", latitude, longitude],
  });

  const dispatch: ThunkDispatch<{ account: AccountSliceProps }> = useDispatchTyped();

  const handleClickShowMore: () => void = () => { Show usages ⌘stanoz *
    dispatch(spotWeatherActions.openDetailedWeatherModal());
  };

  if (isLoading) {
    return <LoadingSpinner />;
  }

  if (isError) {
    return (
      <div className="dark:bg-violetDarker bg-fifth text-violetDark dark:text-darkText rounded-2xl px-3 py-1 text-lg">
        <p>Failed to load weather for this spot!</p>
      </div>
    );
  }
}
```

Rysunek 7.77: Implementacja komponentu BasicSpotWeather (1/2)

```
return (
  data && (
    <div className="dark:bg-violetDarker bg-fifth text-violetDark
      dark:text-darkText 3xl:right-1/4 absolute top-2 right-1/8 flex
      items-center space-x-2 rounded-3xl px-3 py-2 text-xl drop-shadow-md dark:drop-shadow-none">
      <WeatherIcon code={data.weatherCode} isDay={data.isDay} />
      <VerticalLine />
      {data?.temperature}
      <span>&deg;C</span>
      <VerticalLine />
      <div className="flex items-baseline">
        {data.windSpeed}
        <span className="text-sm">m/s</span>
      </div>
      <BiWind />
      <VerticalLine />
      <button
        className="cursor-pointer"
        onClick={handleClickShowMore}
        type="button"
      >
        Show more
      </button>
    </div>
  )
);
}
```

Rysunek 7.78: Implementacja komponentu BasicSpotWeather (2/2)

7.3.6.3.2 Panel z danymi szczegółowymi

Został zbudowany z dziesięciu komponentów:

- DetailedSpotWeather
- WeatherOverview
- WeatherDetails
- WeatherTile
- WindSpeeds
- WindSpeedDisplay
- SelectHeightButton

- `WeatherTimelinePlot`
- `CustomLabel`
- `WeatherIcon`

`DetailedSpotWeather` (rys. 7.79 i 7.80) Odpowiedzialny jest za układ wszystkich części panelu. Zawiera w sobie komponenty `WeatherOverview`, `WeatherDetails`, `WindSpeeds` oraz `WeatherTimelinePlot`, które wyświetlają poszczególne części informacji pogodowych. W celu zapewnienia płynnych przejść, renderowanie i zamiany panelu animowane jest za pomocą biblioteki `motion`. Dane przekazywane komponentom `WeatherOverview` i `WeatherDetails` pobierane są z odpowiedniego endpointu backendu przy użyciu hook'a `useQuery` z `tanstack query`. Podczas ich pobierania wyświetlany jest `LoadingSpinner`, a jeśli operacja zakończy się błędem, pokazywany jest komunikat *Failed to load weather..* Zamknięcie panelu spowoduje wyświetlenie komponentu `BasicSpotWeather`.

```

const slideVariants = {
  hidden: { x: "100%", opacity: 0 },
  visible: { x: 0, opacity: 1 },
  exit: { x: "100%", opacity: 0 },
};

export default function DetailedSpotWeather() : Element { Show usages & stanoz *
  const spotId :number | null = useSelectorTyped((state:{ account: AccountSliceProps; notificatio... }) : number | null => state.spotDetails.spotId);

  const dispatch : ThunkDispatch<{ account: AccountSliceProp... }> = useDispatchTyped();

  const handleCloseModal :() => void = () :void => { Show usages & stanoz
    dispatch(spotWeatherActions.closeDetailedWeatherModal());
  };

  const { latitude, longitude } = useSelectorTyped(
    (state:{ account: AccountSliceProps; notificatio... }) : SpotWeatherSliceProps => state.spotWeather,
  );

  const { data, isError, isLoading, isSuccess } = useQuery({
    queryKey: ["spot-detailed-weather", latitude, longitude, spotId],
    queryFn: () :Promise<SpotDetailedWeatherDto> => getDetailedSpotWeather(latitude, longitude),
  });

  return (
    <motion.div
      key={spotId}
      variants={slideVariants}
      initial="hidden"
      animate="visible"
      exit="exit"
      transition={{ duration: 0.3 }}
      className=" bg-lightGrayishViolet dark:bg-violetDark dark:text-darkText text-violetDark absolute
        right-0 z-[2] flex h-full w-[29rem] flex-col overflow-y-auto px-2 pt-2 pb-16 text-lg xl:top-0 xl:p-2"
    >
      <div className="mt-2 flex">

```

Rysunek 7.79: Implementacja komponentu DetailedSpotWeather (1/2)

```

        <HiX
          className="cursor-pointer text-2xl"
          onClick={handleCloseModal}
        />
        <h1 className="mx-auto text-2xl">Weather</h1>
      </div>
      {isLoading && <LoadingSpinner />}
      {isError && <p>Failed to load weather.</p>}
      {isSuccess && data && (
        <div className="3xl:space-y-4 3xl:mt-2 mt-1 flex flex-col space-y-2">
          <WeatherOverview
            temperature={data.temperature}
            weatherCode={data.weatherCode}
            isDay={data.isDay}
          />
          <WeatherDetails
            rainChance={data.precipitationProbability}
            dewPoint={data.dewPoint}
            uvIndex={data.uvIndexMax}
            humidity={data.relativeHumidity}
          />
          <WindSpeeds />
          <WeatherTimelinePlot />
        </div>
      )}
    </motion.div>
  );
}

```

Rysunek 7.80: Implementacja komponentu DetailedSpotWeather (2/2)

`WeatherOverview` (rys. 7.81 i 7.82) Wyświetla informacje ogólne informacje o spocie i jego pogodzie: temperaturę, aktualny czas, ikonę wraz ze słownym opisem, nazwę, lokalizację. Do poprawnego wyświetlania czasu, z `backendu` pobierana jest strefa czasowa `spota`. Gdy operacja zakończy się sukcesem w `hook'u useEffect` ustawiany jest interwał czasowy, aktualizujący się co minutę. Po usunięciu `komponentu` z drzewa renderowania, funkcja czyszcząca usunie interwał. Obrazowanie stanu pogody realizowane jest za pomocą `WeatherIcon` (por. sekcja 7.3.6.3.3) oraz funkcji `getWeatherAdjective`, która na podstawie kodu pogody oraz parametru `isDay` zwraca słowny opis warunków atmosferycznych. Nazwa oraz lokalizacja `spota` pobierane są ze stanu `Redux'a` za pomocą `hook'a useSelectorTyped`.

```

type WeatherOverviewProps = {
  temperature: number;
  weatherCode: number;
  isDay: boolean;
};

export default function WeatherOverview({ Show usages  ✘stanoz
  temperature,
  weatherCode,
  isDay,
}: WeatherOverviewProps): Element {
  const [currentTime, setCurrentTime] = useState<string>(
    formatISOToAmpm(new Date().toISOString(), timeZone: "Europe/Warsaw"),
  );

  const { city, region } = useSelectorTyped(state: { account: AccountSliceProps; notification: SpotWeatherSliceProps } : SpotWeatherSliceProps => state.spotWeather);
  const { spotId } = useSelectorTyped(state: { account: AccountSliceProps; notification: SpotWeatherSliceProps } : spotModalInitialState => state.spotDetails);

  const { data, isSuccess, isError, isLoading } = useQuery({
    queryKey: ["spot-time-zone", spotId],
    queryFn: () : Promise<SpotTimeZoneDto> => getSpotTimeZone(spotId!),
    enabled: !!spotId,
    retry: false,
  });

  useEffect(() : () => void | undefined => {
    if (isSuccess && data) {
      setCurrentTime(
        formatISOToAmpm(new Date().toISOString(), data.timeZone),
      );
    }

    const intervalId : Timeout = setInterval(() : void => {
      setCurrentTime(
        formatISOToAmpm(new Date().toISOString(), data.timeZone),
      );
    }, 60 * 1000);
  });
}

```

Rysunek 7.81: Implementacja komponentu WeatherOverview (1/2)

```

        return () :void => {
          clearInterval(intervalId);
        };
      },
      [data, isSuccess]);
    }

    return (
      <div className="text-darkText 3xl:mt-2.5 mt-1.5 flex flex-col rounded-lg bg-gradient-to-r
      from-slate-500 via-blue-950 to-blue-900 px-3 py-2.5">
        <div className="ml-12 flex">
          <div>
            <h2>{city}</h2>
            <h3 className="text-grayText text-xs">{region}</h3>
          </div>
          {isLoading && <LoadingSpinner />}
          {isError && (
            <p className="ml-24 text-4xl">
              Failed to get spot current time.
            </p>
          )}
          {isSuccess && <p className="ml-24 text-4xl">{currentTime}</p>}
        </div>
        <div className="mt-3 ml-12 flex text-xl">
          <span className="text-2xl">{temperature}&deg;C</span>
          <div className="ml-22 flex items-center">
            <WeatherIcon
              isDay={isDay}
              code={weatherCode}
              textSize="text-4xl"
            />
            {getWeatherAdjective(weatherCode, isDay)}
          </div>
        </div>
      </div>
    );
  }
}

```

Rysunek 7.82: Implementacja komponentu WeatherOverview (2/2)

`WeatherDetails` Komponent odpowiedzialny za wyświetlanie następujących danych pogodowych: prawdopodobieństwo opadów, punkt rosy, wskaźnik indeksu UV, poziom wilgotności. Indeks UV przedstawiony jest słownie poprzez funkcję `getUvIndexTextLevel`, która określa go na podstawie wartości liczbowej przekazywanej w jej argumencie. Poszczególne dane prezentowane są w formie kafelek przez komponenty `WeatherTile`, razem tworzących siatkę 2x2.

`WindSpeeds` (rys. 7.83 i 7.84) W tym komponencie wyświetlane są prędkości

wiatru na różnych wysokościach w km

h lub m

s. Dane pobierane są z odpowiedniego endpointu backendu przy użyciu hook'a `useQuery`. W trakcie ich ładowania wyświetlany jest `LoadingSpinner`, a w razie błędu komunikat *Failed to load wind speeds data*. W lokalnym stanie (`hook useState`) przechowywana jest informacja, która wysokość została wybrana. Natomiast za jego zmianę odpowiedzialny jest komponent `SelectHeightButton`. Oprócz tego, przycisk ten informuje o wybranej wysokości poprzez warunkową zmianę wyglądu. Za obsługę zmiany jednostek prędkości wiatru oraz wyświetlanie jej wartości odpowiada `WindSpeedDisplay`. Komponent w stanie lokalnym przechowuje informacje o wybranej jednostce, a po każdej jej zmianie przelicza nową wartość wyświetlaną z dokładnością do części dziesiętnej.

```
type windSpeedsSelectionType = {
  label: string;
  value: number;
};

export default function WindSpeeds() : Element { Show usages & stanoz
  const { latitude, longitude } = useSelectorTyped(
    (state) :{account: AccountSliceProps; notificatio...} : SpotWeatherSliceProps => state.spotWeather,
  );
  const { spotId } = useSelectorTyped((state) :{account: AccountSliceProps; notificatio...} : spotModalInitialState => state.spotDetails);
  const { data, isLoading, isError } = useQuery({
    queryKey: ["spot-weather", "wind-speeds", latitude, longitude, spotId],
    queryFn: () :Promise<SpotWeatherWindSpeedsDto> => getWindSpeeds(latitude, longitude, spotId!),
  });
  const windSpeedsSelection: windSpeedsSelectionType[] = [
    { label: "100m", value: data?.windSpeeds100m ?? 0 },
    { label: "200m", value: data?.windSpeeds200m ?? 0 },
    { label: "300m", value: data?.windSpeeds300m ?? 0 },
    { label: "500m", value: data?.windSpeeds500m ?? 0 },
    { label: "750m", value: data?.windSpeeds750m ?? 0 },
    { label: ">1000m", value: data?.windSpeeds1000m ?? 0 },
  ];
  const [selectedHeight, setSelectedHeight] = useState(
    windSpeedsSelection[0].label,
  );
}
```

Rysunek 7.83: Implementacja komponentu WindSpeeds (1/2)

```

const selected :windSpeedsSelectionType = windSpeedsSelection.find(
  (ws :windSpeedsSelectionType ) :boolean => ws.label === selectedHeight,
)!:;

if (isLoading) {return <LoadingSpinner />};

if (isError) {return <p>Failed to load wind speeds data</p>};

return (
  <div className="bg-whiteSmoke dark:bg-second 3xl:mt-4 mt-2 flex items-center rounded-lg py-3 shadow-md">
    <WindSpeedDisplay value={selected.value} />
    <div className="ml-1">
      <h3 className="mb-1.5">Height</h3>
      <ul className="grid grid-cols-2 gap-2">
        {windSpeedsSelection.map((ws :windSpeedsSelectionType ) :Element => (
          <li key={ws.label}>
            <SelectHeightButton
              name={ws.label}
              onClick={() :void => setSelectedHeight(ws.label)}
              selected={ws.label === selectedHeight}
            />
          </li>
        ))}
      </ul>
    </div>
  </div>
);
}

```

Rysunek 7.84: Implementacja komponentu WindSpeeds (2/2)

[WeatherTimelinePlot](#) (rys. 7.85 - 7.88) Ten [komponent](#) przedstawia wykres prezentujący prognozę pogody na trzy dni. Zawiera informacje o temperaturze, prawdopodobieństwie opadów, godzinie oraz ikonę symbolizującą warunki atmosferyczne. Dane pobierane są z [endpointu backendu](#) poprzez [hook useQuery](#). Pobieranie danych oznaczone jest przez [LoadingSpinner](#), a błąd komunikatem *Failed to load data for timeline plot..* Wykres został stworzony przy użyciu elementów [biblioteki victory](#). Głównym [komponentem](#) jest [VictoryChart](#), który przyjmuje dane na osie X i Y, a także pozostałe elementy wykresu. Linia obrazująca wartości temperatury zbudowana jest przy użyciu [VictoryLine](#) oraz [VictoryScatter](#). Natomiast sposób wyświetlania osi poziomej określany jest przez [VictoryAxis](#), przyjmujący [CustomTickLabel](#), w którym dokładnie poszczególne elementy opisuje. W celu zachowania stałej wysokości obszaru wyświetlanych danych niezależnie od ich rozpiętości, wartości temperatur są znormalizowane do wirtualnego zakresu od

0 do 100.

```
const VIRTUAL_Y_MIN = 0;
const VIRTUAL_Y_MAX = 100;

export default function WeatherTimelinePlot() : false | Element { Show usages & stanoz *
  const { latitude, longitude } = useSelectorTyped(
    | (state :{account:AccountSliceProps; notificatio... } :SpotWeatherSliceProps => state.spotWeather,
  );

  const { spotId } = useSelectorTyped(state :{account:AccountSliceProps; notificatio... } :spotModalInitialState => state.spotDetails);

  const { data, isLoading, isError, isSuccess } = useQuery({
    queryKey: [
      "spot-weather",
      "timeline-plot",
      latitude,
      longitude,
      spotId,
    ],
    queryFn: () :Promise<SpotWeatherTimelinePlotData[]> =>
      | getWeatherDataForTimelinePlot(latitude, longitude, spotId!),
  });

  if (isLoading) {
    | return <LoadingSpinner />;
  }

  if (isError) {
    | return <p>Failed to load data for timeline plot.</p>;
  }
}
```

Rysunek 7.85: Implementacja komponentu WeatherTimelinePlot (1/4)

```

const actualMax :number = data
  ? Math.max(...data.map((d :SpotWeatherTimelinePlotData) :number => d.temperature)) + 1
  : 100;
const span :number = actualMax - actualMin || 1;
const mapY :(temp:number)=> number = (temp: number) :number => Show usages & stanoz
  VIRTUAL_Y_MIN +
  ((temp - actualMin) / span) * (VIRTUAL_Y_MAX - VIRTUAL_Y_MIN);

return (
  isSuccess &&
  data && (
    <div className="3xl:mt-8 mt-2">
      <h2 className="text-2xl">Timeline</h2>
      <div className="bg-mediumDarkBlue text-darkText scrollbar-thin scrollbar-track-rounded-lg
        scrollbar-track-sky-950 hover:scrollbar-thumb-sky-800 scrollbar-thumb-sky-700
        scrollbar-thumb-rounded-full scroll-grab mx-auto mt-4 w-[27.5rem] overflow-x-auto rounded-lg">
        <VictoryChart
          domain={{
            x: [
              new Date(
                Math.min(
                  ...data.map((d :SpotWeatherTimelinePlotData) :number =>
                    new Date(d.time).getTime(),
                  ),
                ),
              ),
            ],
          },
        >
      </div>
    </div>
  )
)

```

Rysunek 7.86: Implementacja komponentu WeatherTimelinePlot (2/4)

```
    new Date(
      Math.max(
        ...data.map((d : SpotWeatherTimelinePlotData) : number =>
          new Date(d.time).getTime(),
        ),
      ),
    ],
    y: [VIRTUAL_Y_MIN, VIRTUAL_Y_MAX],
)
scale={{ x: "time" }}
theme={VictoryTheme.clean}
padding={{ top: 90, bottom: 60, left: 50, right: 50 }}
width={5000}
height={460}
containerComponent={
  <VictoryContainer responsive={false} />
}
>
<VictoryAxis
  orientation="top"
  tickValues={data.map((d : SpotWeatherTimelinePlotData) : Date => new Date(d.time))}
  tickLabelComponent={<CustomTickLabel data={data} />}
  style={{
    tickLabels: { fontSize: 12 },
    axis: { strokeWidth: 0 },
  }}
}
```

Rysunek 7.87: Implementacja komponentu WeatherTimelinePlot (3/4)

```

        />
      <VictoryLine
        data={data.map((d : SpotWeatherTimelinePlotData ) :{ x: Date; y: number } => {
          return {
            x: new Date(d.time),
            y: mapY(d.temperature),
          };
        })}
        style={{ data: { stroke: "#ece9e9" } }}
        interpolation={"natural"}
      />

      <VictoryScatter
        data={data.map((d : SpotWeatherTimelinePlotData ) :{ x: Date; y: number } => {
          return {
            x: new Date(d.time),
            y: mapY(d.temperature),
          };
        })}
        style={{ data: { fill: "white" } }}
      />
    </VictoryChart>
  </div>
</div>
);
}

```

Rysunek 7.88: Implementacja komponentu WeatherTimelinePlot (4/4)

7.3.6.3.3 Komponent wspólny

W tym rozdziale przedstawiono komponent, z którego korzystają obydwie części modułu pogody.

WeatherIcon To komponent, którego zadaniem jest na podstawie kodu pogody (*code*) i informacji czy jest dzień (*isDay*) wybrać odpowiednią ikonę symbolizującą stan pogody. W przypadku braku znalezienia odpowiedniego elementu zwracana jest ikona znaku zapytania. Komponent opcjonalnie przyjmuje parametr określający rozmiar ikony (*fontSize*), będący klasą Tailwind. Jeśli nie zostanie podany, ustawiiona zostanie domyślna opcja *text-3xl*. Poniżej przedstawiono implementację tego komponentu.

```
const weatherIcons: Record<number, IconType[]> = {
    0: [PiMoonLight, WiDaySunny],
    1: [CiCloudMoon, WiDayCloudy],
    2: [CiCloudMoon, WiDayCloudy],
    3: [WiCloud, WiCloud],
    45: [WiFog, WiFog],
    48: [WiFog, WiFog],
    51: [WiRaindrops, WiRaindrops],
    53: [WiRaindrops, WiRaindrops],
    55: [WiRaindrops, WiRaindrops],
    56: [WiRainMix, WiRainMix],
    57: [WiRainMix, WiRainMix],
    61: [WiRaindrops, WiRaindrops],
    63: [WiRaindrops, WiRaindrops],
    65: [WiRaindrops, WiRaindrops],
    66: [WiRainMix, WiRainMix],
    67: [WiRainMix, WiRainMix],
    71: [WiSnow, WiSnow],
    73: [WiSnow, WiSnow],
    75: [WiSnow, WiSnow],
    77: [WiSnow, WiSnow],
    80: [WiRaindrops, WiRaindrops],
    81: [WiRaindrops, WiRaindrops],
    82: [WiRaindrops, WiRaindrops],
    85: [WiSnow, WiSnow],
    86: [WiSnow, WiSnow],
    95: [WiThunderstorm, WiThunderstorm],
    96: [WiThunderstorm, WiThunderstorm],
    99: [WiThunderstorm, WiThunderstorm],
};
```

Rysunek 7.89: Implementacja komponentu WeatherIcon (1/2)

```
export default function WeatherIcon({ Show usages  ↵ stanoz
  code,
  textSize,
  isDay,
}: {
  code: number;
  textSize?: string;
  isDay: boolean;
}): Element {
  const Icon :IconType = weatherIcons[code][isDay ? 1 : 0] || FaQuestion;
  return <Icon className={`${textSize ?? "text-3xl"}`}>;
}
```

Rysunek 7.90: Implementacja komponentu WeatherIcon (2/2)

7.3.6.4 Duża galeria zdjęć i filmów

Wyświetlane są w niej wszystkie zdjęcia i filmy dodane do [spota](#), zarówno poprzez komentarze jak i dedykowany formularz.

Duża galeria została zbudowana z dwunastu [komponentów](#):

- [ExpandedSpotMediaGallery](#)
- [ExpandedGallerySidebar](#)
- [SortingAndFilterPanel](#)
- [OptionButton](#)
- [ExpandedMediaGalleryActions](#)
- [ExpandedMediaDisplay](#)
- [MediaInfoDisplay](#)
- [MediaDisplayFilterPanel](#)
- [ExpandedGalleryPhoto](#)

- `ExpandedGalleryVideo`
- `ExpandedGalleryPanel`
- `FullscreenMediaModal`

`ExpandedSpotMediaGallery` Komponent, który jest kontenerem ustalającym położenie pozostałych elementów. Jego otworzenie oraz zamknięcie jest animowane przy użyciu biblioteki motion.

`ExpandedGallerySidebar` (rys. 7.91 - 7.98) Wyświetla listę zdjęć lub filmów przewijalną przy użyciu infinite scroll. Dane pobierane są z paginacją z backendu przy pomocy hook'a `useInfiniteQuery`. Mechanizm ten został zrealizowany za pomocą `IntersectionObserver`, który nasłuchuje na element powiązany z referencją `containerRef`. Zbliżenie się do `loadNextPageRef` lub `loadPreviousPageRef` określa odpowiedni kierunek pobierania danych (kolejne lub poprzednie strony), który jest zapisywany w `fetchDirection`. W czasie ładowania stron wyświetlany jest `LoadingSpinner`. W hook'u `useEffect` po zmianie danych na podstawie zapisanej wartości, dane dodawane są na początek lub koniec listy przechowywanej w stanie `Redux`. Pierwszy element z listy jest wyświetlany w komponencie `ExpandedMediaDisplay`. Po zmianie sortowania lub typu medii (zdjęcia lub filmy), lista jest resetowana, a element ustalany na nowo. Kliknięcie w dowolnego media spowoduje wyświetlenie go w `ExpandedMediaDisplay`. Lista może być zwijana lub rozwijana przy użyciu odpowiedniego przycisku, a przejścia są animowane za pomocą biblioteki motion.

```

return (
  <div className="flex items-center bg-black">
    <LayoutGroup id="spot-expanded-media-gallery-layout-group">
      <AnimatePresence mode="wait"
        onExitComplete={() : void => handleCloseSidebar()}>
        >
          {showExpandedGallerySidebar && (
            <motion.div key="expanded-spot-gallery-sidebar"
              layout
              variants={sidebarVariants}
              initial="closed"
              exit="exit"
              animate={
                showExpandedGallerySidebar ? "open" : "closed"
              }
              transition={{ duration: 0.4, ease: "easeInOut" }}
              ref={containerRef}
              className="dark:bg-violetHeavyDark bg-fifth 3xl:w-[30rem] h-full w-[20rem] overflow-y-auto p-2 xl:w-[25rem] xl:overflow-y-hidden"
            >
              <div className="mt-1 grid w-full grid-cols-3 items-center">
                <div>
                  <h2 className="dark:text-darkText text-violetBrightText text-center text-2xl">
                    Gallery
                  </h2>
                  <IoClose onClick={handleCloseSidebar}>
                    <span> </span>
                  </IoClose>
                </div>
              </div>
            </motion.div>
          )}
        >
      </AnimatePresence>
    </LayoutGroup>
  </div>
)

```

Rysunek 7.91: Implementacja komponentu ExpandedSpotMediaGallery (1/8)

```

    </div>
  </div>
<SortingAndFilterPanel />
{isLoading && <LoadingSpinner />}
{isError && <p>Failed to fetch list of media.</p>}
<div className="dark:scrollbar-track-violetDark dark:hover:scrollbar-thumb-violetLight
  scrollbar-thumb-rounded-full scrollbar-thin 3xl:h-[71rem] overflow-y-auto xl:h-[35rem] 2xl:h-[56rem]">
  <div
    ref={loadPreviousPageRef}
    className="invisible h-1"
  >
    {isFetchingPreviousPage && <LoadingSpinner />}
    {mediaList.length === 0 ? (
      <p className="text-violetDark dark:text-darkText text-center">
        No{" "}
        {mediaType === MediaType.PHOTO
          ? "photos"
          : "films"}{" "}
        to display.
      </p>
    ) : (
      <ul className="flex flex-col items-center space-y-2">
        {mediaList.map((media : SpotExpandedGallerySidebarMedi... ) : Element =>
          media.mediaType ===
            MediaType.PHOTO ? (
              <li>

```

Rysunek 7.92: Implementacja komponentu ExpandedSpotMediaGallery (2/8)

```
<li
  key={media.id}
  className="flex h-72 w-[28rem] cursor-pointer items-center
justify-center overflow-hidden bg-zinc-800 first:rounded-t-2xl last:rounded-b-2xl"
onClick={() : void} =>
  handleClicksetCurrentMedia(
    media,
  )
>
<img
  className="max-h-full max-w-full object-contain"
  src={media.url}
  alt={media.url}
/>
</li>
) : (
<li
  key={media.id}
  className="relative overflow-hidden first:rounded-t-2xl last:rounded-b-2xl"
onClick={() : void} =>
  handleClicksetCurrentMedia(
    media,
  )
>
```

Rysunek 7.93: Implementacja komponentu ExpandedSpotMediaGallery (3/8)

```

        <div className="bg-darkBg/80 absolute inset-0 z-10 flex cursor-pointer items-center justify-center text-2xl 2xl:text-4xl">
          | <FaRegCirclePlay />
        </div>
        <div className="2xl:[w-26rem] 3xl:w-[28rem] z-10 flex h-72 items-center justify-center">
          <ReactPlayer
            | playing={false}
            | src={media.url}
            | controls={false}
            | style={{
            |   width: "100%",
            |   height: "100%",
            |   aspectRatio:
            |     "16/9",
            |   "--controls":
            |     "none",
            | }}
```

Rysunek 7.94: Implementacja komponentu ExpandedSpotMediaGallery (4/8)

```

{mediaList.length === 0 ? (
  <p className="text-center">
    No{" "}
    {mediaType === MediaType.PHOTO
      ? "photos"
      : "films"}{" "}
    to display.
  </p>
) : (
  <ul className="flex flex-col items-center space-y-2">
    {mediaList.map((media : SpotExpandedGallerySidebarMedia) : Element =>
      media.mediaType ===
        MediaType.PHOTO ? (
          <li
            | key={media.id}
            | className="flex h-72 w-[28rem] cursor-pointer items-center justify-center overflow-hidden bg-zinc-800 first:rounded-t-2xl last:rounded-b-2xl"
            | onClick={() : void =>
            |   handleClickSetCurrentMedia(
            |     media,
            |   )
            | }
```

Rysunek 7.95: Implementacja komponentu ExpandedSpotMediaGallery (5/8)

```
        <img
          className="max-h-full max-w-full object-contain"
          src={media.url}
          alt={media.url}
        />
      </li>
    ) : (
      <li
        key={media.id}
        className="relative overflow-hidden first:rounded-t-2xl last:rounded-b-2xl"
        onClick={() =>
          handleClickSetCurrentMedia(
            media,
          )
        }
      >
        <div className="■bg-darkBg/80 absolute inset-0 z-10 flex cursor-pointer
          items-center justify-center text-2xl 2xl:text-4xl">
          <FaRegCirclePlay />
        </div>
        <div className="2xl:[w-26rem] 3xl:w-[28rem] z-10 flex h-72 items-center justify-center">
          <ReactPlayer
            playing={false}
            src={media.url}
            controls={false}
            style={{

```

Rysunek 7.96: Implementacja komponentu ExpandedSpotMediaGallery (6/8)

```
    width: "100%",  
    height: "100%",  
    aspectRatio:  
      "16/9",  
    "--controls":  
      "none",  
  }]  
  >/>  
  </div>  
  </li>  
) ,  
)  
</ul>  
)  
{isFetchingNextPage && <LoadingSpinner />}  
<div  
  ref={loadNextPageRef}  
  className="invisible h-1"  
/>  
</div>  
</motion.div>  
)  
</AnimatePresence>  
<motion.div  
  layout  
  animate={showExpandedGallerySidebar ? "open" : "closed"}>
```

Rysunek 7.97: Implementacja komponentu ExpandedSpotMediaGallery (7/8)

```

<motion.div
  layout
  animate={showExpandedGallerySidebar ? "open" : "closed"}
  transition={{
    duration: 0.4,
    ease: "easeInOut",
  }}
  className="flex h-full items-center bg-black"
>
  <div
    className="dark:bg-violetLightDark hover:bg-darkText bg-fifth
    dark:hover:bg-violetLightDarker w-fit cursor-pointer rounded-r-2xl py-3.5"
    onClick={handleClickToggleSidebar}
  >
    <motion.div
      initial="closed"
      animate={{
        showExpandedGallerySidebar ? "open" : "closed"
      }}
      key="expanded-spot-gallery-sidebar-arrow"
      variants={arrowVariants}
      transition={{ duration: 0.3 }}
    >
      <FaChevronLeft
        className={`text-darkBorder text-3xl dark:text-black`}
      />
    </motion.div>
  </div>
</motion.div>

```

Rysunek 7.98: Implementacja komponentu ExpandedSpotMediaGallery (8/8)

SortingAndFilterPanel Komponent ten odpowiedzialny jest za obsługę ustawiania filtrów i sortowania. Dostępne opcje wyświetlane są w **OptionButton**, a aktywne oznaczane poprzez wyróżniający się wygląd. Obecnie wybrane wartości pobierane są ze stanu **Redux'a**, a nowe w nim ustawiane. **Komponent** umożliwia jednoczesne wybranie jednego filtru i sortowania. Typ początkowy filtru określany jest na podstawie klikniętego media, natomiast sortowanie ustawiane jest jako od najnowszych.

ExpandedMediaDisplay Zbudowany jest z **komponentów MediaInfoDisplay**,

`MediaDisplayFilterPanel` oraz `ExpandedGalleryPanel`. Pobiera wcześniej wybrane media z `backendu`, a następnie w zależności od typu wyświetla je w `ExpandedGalleryPhoto` (zdjęcie) lub `ExpandedGalleryVideo` (film). Obsługa video odbywa się za pomocą elementów `biblioteki react-player`. W trakcie pobierania danych wyświetlany jest `LoadingSpinner`, a błędzie informuje komunikat *Failed to fetch media..*

`MediaInfoDisplay Komponent` odpowiedzialny za wyświetlanie informacji o wybranym zdjęciu. Zawiera dane o autorze: nazwę użytkownika i jego zdjęcie profilowe oraz datę publikacji media.

`MediaDisplayFilterPanel` Służy do przełączania typu mediów (zdjęcia lub filmy), a wybrana opcja zapisywana jest w stanie `Redux'a`.

`ExpandedGalleryPanel Komponent` obsługujący akcje, które użytkownik może wykonać w stosunku do wyświetlanego media. Polubienie elementu realizowane jest poprzez funkcję `handleClickEditSpotMediaLikes`. Jeżeli użytkownik nie jest zalogowany, w `Notification` wyświetlany jest odpowiedni komunikat. W przeciwnym wypadku w zależności od tego czy polubienie jest dodawane, czy usuwane, wykonywana jest optymistyczna zmiana `UI` (por. sekcja [7.1.2.2](#)). Wysłanie odpowiedniego żądania odbywa się za pomocą mutacji z `biblioteki Tanstack Query`. W przypadku błędu w `komponencie Notification` wyświetlany jest komunikat *Failed to edit spot media likes..* W celu utrzymania aktualności danych, powodzenie operacji spowoduje unieważnienie wszystkich zapytań (queries) powiązanych z wybranym media, co wymusi ponowne pobranie danych. Funkcja `handleSetMediaFullscreen` obsługuje powiększenie media na cały ekran poprzez otworzenie `modalu FullscreenMediaModal`. Kliknięcie przycisku z ikoną `BiDownload` powoduje przekierowanie do wybranego zasobu, który jest pobierany przez przeglądarkę. Montowanie `komponentu` wykonuje kod w `useEffect`, wysyłający przy użyciu `mutateAsync` z `hook'a useMutation` asynchroniczne żądanie zwiększające liczbę wyświetleń media.

`ExpandedMediaGalleryActions` Element realizujący akcje udostępnienia media oraz zamknięcia dużej galerii zdjęć i filmów. Udostępnienie obsługuje funkcja `handleClickShare`, która przy użyciu `navigator.clipboard.writeText` kopiuje link do wybranego elementu do pamięci podręcznej systemu użytkownika. Komuni-

katy o stanie zakończenia operacji wyświetlane są w komponencie `Notification`.

7.3.7 Chat

7.3.8 Forum

7.3.9 Panel użytkownika

W niniejszym rozdziale przedstawiono implementację panelu użytkownika po stronie `frontendu`.

Jednym z głównych modułów aplikacji jest panel użytkownika. Został on podzielony na osiem zakładek:

- **Profile** – profil użytkownika wraz z możliwością zmiany zdjęcia profilowego;
- **Spots** – listy `spotów` dodanych do różnych kategorii (ulubione, planowane do odwiedzenia, odwiedzone i ocenione pozytywnie, odwiedzone i ocenione negatywnie);
- **Photos** – lista zdjęć dodanych przez użytkownika do `spotów`, komentarzy pod `spotami` oraz na forum;
- **Movies** – lista filmów dodanych przez użytkownika do `spotów`, komentarzy pod `spotami` oraz na forum;
- **Social** – listy znajomych, obserwowanych i obserwujących;
- **Add spot** – lista `spotów` dodanych przez użytkownika oraz formularz dodawania nowego `spota`;
- **Comments** – lista komentarzy dodanych przez użytkownika pod `spotami`;
- **Settings** – ustawienia konta użytkownika (zmiana nazwy użytkownika, adresu e-mail oraz hasła).

Wszystkie zakładki korzystają z uniwersalnego komponentu `AccountWrapper`, który zapewnia spójny układ i styl widoków (por. sekcja [7.3.9.9](#)).

7.3.9.1 Profile

Zakładka **Profile** odpowiada za prezentację profilu użytkownika. Została zbudowana z pięciu podstawowych komponentów:

- `UserOwnProfile`,
- `ProfileForViewer`,
- `Profile`,
- `ProfileStat`,
- `MostPopularImage`.

`UserOwnProfile` (rys. 7.99) Ten komponent służy do wyświetlania profilu załogowanego użytkownika. Dane profilu są pobierane za pomocą `hooka useQuery` z biblioteki `TanStack Query`, który odwołuje się do odpowiedniego `endpointu backendu`. Na czas pobierania danych wyświetlany jest komponent `LoadingSpinner`, a po poprawnym załadowaniu profilu renderowany jest komponent `Profile` z przekazanym obiektem `userData`.

```
export default function UserOwnProfile(): Element { Show usages ▾ Mredosz
  const { data, isLoading } = useQuery({
    queryFn: getUserOwnProfile,
    queryKey: ["userProfile"],
  });

  if (isLoading) {
    return <LoadingSpinner />;
  }

  return <Profile userData={data!} />;
}
```

Rysunek 7.99: Komponent `UserOwnProfile`.

`ProfileForViewer` (rys. 7.100) Komponent wyświetla profil innego użytkownika. Nazwa użytkownika pobierana jest ze ścieżki `URL` za pomocą `hooka useParams`, a dane profilu – za pomocą `useQuery`, które wysyła zapytanie do `backendu` z przekazaną nazwą.

Oprócz wspólnej części logicznej z widokiem własnego profilu wykorzystywane są:

- przyciski `Button` przekazywane do `Profile` jako `children`, obsługujące akcje `follow` oraz zarządzanie znajomymi;
- komponent `Modal` do potwierdzania usunięcia znajomego lub zaprzestania obserwowania.

Zachowanie przycisków zależy od tego, czy użytkownik jest zalogowany, czy obserwuje dany profil oraz czy osoba ta znajduje się na liście znajomych. W przypadku błędu autoryzacji wyświetlany jest komunikat zachęcający do zalogowania, a inne błędy są mapowane na komunikaty w komponencie `Notification`.

Relacje społecznościowe modyfikowane są przy użyciu kilku operacji po stronie `API`, odpowiadających za dodawanie lub usuwanie znajomych, zarządzanie listą obserwowanych oraz akceptowanie lub odrzucanie zaproszeń. Po powodzeniu odświeżane są dane profilu użytkownika. Jeżeli z `backendu` zwrócona zostanie informacja, że jest to profil zalogowanego użytkownika, w `hooku useEffect` następuje przekierowanie na adres `/account/profile`. W sytuacji, gdy profil o podanej nazwie użytkownika nie istnieje, renderowany jest komunikat: "No profile data available".

```

<>
  <Profile
    userData={data.profile}
    username={username}
    isProfileForViewer
  >
    <div className="text-darkText flex w-full flex-wrap justify-center gap-5 xl:flex-wrap">
      <Button
        variant={ButtonVariantType.PROFILE}
        onClick={
          data.isFollowing
            ? confirmRemoveFromFollow
            : handleEditToFollowed
        }
      >
        {data.isFollowing ? "unfollow" : "follow"}
      </Button>
      <Button
        variant={ButtonVariantType.PROFILE}
        onClick={getFriendActionHandler(data.friendStatus)}
      >
        {statusToMessage[data.friendStatus] ?? "unknown status"}
      </Button>
    </div>
  </Profile>
  <Modal
    onClose={closeModal}
    onClick={handleConfirm}
    isOpen={isModalOpen}
  >
    <h2 className="text-xl text-shadow-md">
      {modalAction === SocialListType.FRIENDS
        ? `Are you sure you want to remove ${username} as a friend?`
        : `Are you sure you want to unfollow ${username}?`}
    </h2>
  </Modal>
</>

```

Rysunek 7.100: Komponent ProfileForViewer.

Profile Jest to główny komponent odpowiedzialny za prezentację profilu i wykorzystywany zarówno w widoku własnym, jak i w widoku innego użytkownika. Cały widok opakowany jest w **AccountWrapper** z wariantem **PROFILE** (por. sekcja [7.3.9.9](#)).

Po lewej stronie wyświetlane jest zdjęcie profilowe. Dla własnego profilu wy-

korzystano `hook useMutation`, który umożliwia zmianę zdjęcia. Po najechaniu na nie kursorem pojawia się półprzezroczyste tło z ikoną edycji i napisem "Change profile photo.", a kliknięcie uruchamia wybór nowego pliku. Po udanej aktualizacji zdjęcia odświeżane są dane profilu oraz wyświetlany jest komunikat sukcesu. W przypadku błędu prezentowany jest odpowiedni komunikat informujący o niepowodzeniu aktualizacji.

Po prawej stronie wyświetlana jest nazwa użytkownika oraz cztery kafelki `ProfileStat`, przedstawiające liczbę znajomych, obserwowanych, obserwujących oraz zdjęć. Kliknięcie w wybraną statystykę powoduje przejście do odpowiedniej sekcji społecznościowej (por. sekcja [7.3.9.5](#)); nawigacja realizowana jest przez `useNavigate` oraz akcję `socialAction.setType`.

Pod główną częścią profilu wyświetlany jest nagłówek "most popular photos". Jeżeli użytkownik posiada popularne zdjęcia, wyświetlane są maksymalnie cztery miniatury za pomocą komponentu `MostPopularImage`. W przeciwnym razie prezentowany jest komunikat: "This user hasn't added any photos." (dla profilu innego użytkownika) lub "You haven't added any photos." (dla własnego profilu).

`ProfileStat` Komponent prezentuje pojedynczą statystykę profilu (liczbę znajomych, obserwujących, obserwowanych oraz zdjęć) w formie kafelka, który pełni rolę przycisku nawigacyjnego do odpowiedniej listy społecznościowej (por. sekcja [7.3.9.5](#)).

`MostPopularImage` Komponent odpowiada za wyświetlenie pojedynczego zdjęcia wraz z liczbą polubień i wyświetleń. Informacje umieszczone są na półprzezroczystym pasku w dolnej części kafelka.

7.3.9.2 Spots

Zakładka `Spots` odpowiada za prezentację list `spotów` przypisanych do różnych kategorii (ulubione, planowane do odwiedzenia, odwiedzone itp.). Składa się z trzech głównych komponentów:

- `FavoriteSpots`,
- `FavoriteSpotTile`,

- `FavoriteSpotTags`.

`FavoriteSpots` (rys. 7.101) Jest to główny komponent zakładki.

Wykorzystuje tablicę `menuTypes`, która definiuje dostępne typy list (wszystkie, ulubione, planowane do odwiedzenia, odwiedzone i ocenione pozytywnie, odwiedzone i ocenione negatywnie), a aktualnie wybrany typ przechowywany jest w stanie za pomocą `hooka useState`.

Dane `spotów` są pobierane za pomocą `hooka useInfiniteQuery`, który wysyła zapytania do `backendu` z uwzględnieniem wybranego typu listy. Mechanizm `infinite scrolla` realizowany jest przy użyciu `IntersectionObserver`, który nasłuchuje na element powiązany z referencją. Widok opakowany jest w `AccountWrapper` z wariantem `FAVORITE_SPOTS` oraz nagłówkami `AccountTitle` z tekstem "spots lists". Pod nagłówkiem renderowane są przyciski (komponent `Button`) do zmiany typu listy.

Podczas ładowania danych wyświetlany jest `LoadingSpinner`. Jeżeli użytkownik nie posiada żadnych `spotów` w wybranej liście, prezentowany jest komunikat: "You don't have any spots in your list.", w przeciwnym razie renderowana jest lista kafelków `FavoriteSpotTile`.

```

<AccountWrapper variant={AccountWrapperType.FAVORITE_SPOTS}>
  <AccountTitle text="spots lists" />
  <div className="flex max-w-full flex-col items-center gap-5 lg:flex-row xl:mx-27">
    {menuTypes.map((m : {label: string; type: FavoriteSpotsListT... }) : Element => (
      <Button
        key={m.label}
        variant={ButtonVariantType.FAVORITE_SPOT_MENU}
        onClick={() : void => handleSetSelectedType(m.type)}
        className={
          selectedType === m.type
            ? "dark:bg-violetLight bg-violetDark/87"
            : "dark:bg-violetDark bg-violetLight"
        }
      >
        {m.label}
      </Button>
    )))
  </div>
  {isLoading && <LoadingSpinner />}
  <div className="flex flex-col items-center space-y-5 lg:mx-27">
    {allItems?.length
      ? allItems?.map((spot : FavoriteSpot) : Element => (
          <FavoriteSpotTile
            spot={spot}
            key={spot.id}
            selectedType={selectedType}
          />
        ))
      : !isLoading && (
        <p className="mt-10 text-center text-gray-500">
          You don't have any spots in your list.
        </p>
      )}
    <div ref={loadMoreRef} className="h-10" />
    {isFetchingNextPage && <LoadingSpinner />}
  </div>
</AccountWrapper>

```

Rysunek 7.101: Komponent FavoriteSpots.

Komponent FavoriteSpotTile reprezentuje pojedynczy kafelek spota. Wyświetlane są: zdjęcie, liczba wyświetleń, lokalizacja, średnia ocena (komponent

`Rate`), nazwa, tagi (komponent `FavoriteSpotTags`), krótki opis oraz przyciski otwarcia `spota` na mapie i usunięcia z listy.

Do modyfikacji listy wykorzystywana jest `mutacja` obsługująca usuwanie spotów z danej kategorii. Po powodzeniu odświeżane są dane listy, tak aby kafelek został usunięty z widoku. Przed usunięciem wyświetlany jest modal potwierdzenia (komponent `Modal`), którego stan widoczności kontrolowany jest za pomocą `hooka useBoolean`.

`FavoriteSpotTags` Komponent prezentuje listę tagów `spota` jako niewielkie etykiety z nazwą tagu.

7.3.9.3 Photos

Zakładka `Photos` odpowiada za prezentację zdjęć dodanych przez użytkownika. Korzysta z następujących głównych komponentów:

- `Photos`,
- `Media`,
- `Photo`.

`Photos` (rys. 7.102) Komponent pełni rolę warstwy konfigurującej widok zdjęć. Do zarządzania sortowaniem i filtrowaniem po dacie wykorzystywany jest `czysty hook useDateSortFilter`, który zwraca typ sortowania oraz zakres dat. Dane są pobierane za pomocą `useInfiniteQuery`, które wysyła zapytania do `baczkendu` z uwzględnieniem aktualnych filtrów. Mechanizm `infinite scroll` zrealizowano przy użyciu `IntersectionObserver`.

W przypadku błędu podczas pobierania zdjęć w `hooku useEffect` ustawiany jest komunikat błędu w komponencie `Notification`. Na końcu komponent przekazuje dane do `Media` z wariantem `AccountWrapperType.PHOTOS`.

```

export default function Photos() :Element { Show usages ▾ Mredosz
  const dispatch :ThunkDispatch<{account: AccountSliceProp...} = useDispatchTyped();
  const { sortType, searchDate, handleSelectType, handleChangeDate } =
    useDateSortFilter();
  const loadMoreRef :MutableRefObject<HTMLDivElement | null> = useRef<HTMLDivElement | null>(null);

  const {
    data,
    isLoading,
    isError,
    fetchNextPage,
    hasNextPage,
    isFetchingNextPage,
  } = useInfiniteQuery([ 4 elements... ]);

  const allItems :DatedMediaGroup[] | undefined = data?.pages.flatMap((page :DatedMediaGroupPageDto) :DatedMediaGroup[] => page.items);

  useEffect(() :() => void | undefined => {...}, [hasNextPage, isFetchingNextPage, fetchNextPage]);

  useEffect(() :void => {...}, [isError]);

  return (
    <Media
      variant={AccountWrapperType.PHOTOS}
      searchDate={searchDate}
      onSortChange={handleSelectType}
      onChangeDate={handleChangeDate}
      isLoading={isLoading}
      mediaList={allItems}
      loadMoreRef={loadMoreRef}
      isFetchingNextPage={isFetchingNextPage}
    />
  );
}

```

Rysunek 7.102: Komponent Photos.

Photo Komponent reprezentuje pojedyncze zdjęcie użytkownika. Zdjęcie prezentowane jest jako kwadratowy kafelek z cieniem, a w dolnej części na półprzezroczystym pasku wyświetlana jest liczba polubień i wyświetleń wraz z odpowiednimi ikonami.

Media (rys. 7.103) Jest to uniwersalny komponent używany zarówno w zakładce **Photos**, jak i **Movies** (por. sekcja 7.3.9.4). Widok jest opakowany w **AccountWrapper**, a nagłówek sekcji wyświetlany przez **AccountTitle** z tekstem zależnym od wariantu.

Poniżej nagłówka znajduje się panel **SortAndDateFilters** (por. sekcja 7.3.9.9) umożliwiający zmianę sortowania oraz zakresu dat. Lista mediów grupowana jest po dacie; dla każdej grupy wyświetlany jest **DateBadge** z datą, a poniżej siatka miniatur. W zależności od wariantu wykorzystywany jest komponent **Photo** lub

Movie.

Jeżeli użytkownik nie posiada żadnych mediów, wyświetlany jest komunikat: "You haven't added any photos." lub "You haven't added any movies.". Podczas ładowania danych początkowych oraz kolejnych stron prezentowany jest **LoadingSpinner**.

```

<AccountWrapper variant={variant}>
  <div className="flex flex-wrap items-center justify-between space-y-2 space-x-3">
    <AccountTitle
      text={
        variant === AccountWrapperType.PHOTOS
        ? "Photos"
        : "Movies"
      }
    />
    <SortAndDateFilters
      onSortChange={onSortChange}
      onDateChange={onDateChange}
      from={searchDate.from}
      to={searchDate.to}
    />
  </div>
  <div className="flex flex-col gap-3 lg:mx-27">
    {isLoading && <LoadingSpinner />}
    {mediaList?.length
      ? mediaList?.map(({ date, media }) : DatedMediaGroup : Element => (
          <div className="flex flex-col space-y-3" key={date}>
            <DateBadge date={date} />
            <ul className="3xl:grid-cols-5 grid gap-3 md:grid-cols-2 xl:grid-cols-3
              2xl:grid-cols-4">
              {media?.map((m : Media) : Element => {...})}
            </ul>
          </div>
        ))
      : null
    }
    {!mediaList?.length && !isLoading ? (
      <p className="text-center text-lg">
        You haven't added any{" "}
        {variant === AccountWrapperType.PHOTOS
          ? "photos"
          : "movies"}
        .
      </p>
    ) : null}
    <div ref={loadMoreRef} className="h-10" />
    {isFetchingNextPage && <LoadingSpinner />}
  </div>
</AccountWrapper>

```

Rysunek 7.103: Komponent Media.

7.3.9.4 Movies

Zakładka **Movies** odpowiada za prezentację filmów dodanych przez użytkownika. Pod względem struktury jest bardzo podobna do zakładki **Photos** (por. sekcja [7.3.9.3](#)) i korzysta z tych samych komponentów ogólnych:

- **Movies**,
- **Media**,
- **Movie**.

Movies (rys. [7.104](#)) Komponent konfiguruje widok filmów i przekazuje dane do **Media** z wariantem `AccountWrapperType.MOVIES`. Ponownie wykorzystywany jest hook `useDateSortFilter`, a dane pobierane są za pomocą `useInfiniteQuery` z uwzględnieniem wybranego sortowania i zakresu dat. W przypadku błędu ustawiany jest komunikat w komponencie **Notification**.

```

export default function Movies() :Element { Show usages ▾ Mredosz
  const dispatch :ThunkDispatch<{account: AccountSliceProp...} = useDispatchTyped();
  const { sortType, searchDate, handleSelectType, handleChangeDate } =
    useDateSortFilter();
  const loadMoreRef :MutableRefObject<HTMLDivElement | null> = useRef<HTMLDivElement | null>(null);

  const {
    data,
    isLoading,
    isError,
    fetchNextPage,
    hasNextPage,
    isFetchingNextPage,
  } = useInfiniteQuery([ 4 elements... ]);

  const allItems :DatedMediaGroup[] | undefined = data?.pages.flatMap((page :DatedMediaGroupPageDto) :DatedMediaGroup[] => page.items);

  useEffect(() :()=> void | undefined => {...}, [hasNextPage, isFetchingNextPage, fetchNextPage]);

  useEffect(() :void => {...}, [isError]);

  return (
    <Media
      variant={AccountWrapperType.MOVIES}
      searchDate={searchDate}
      onSortChange={handleSelectType}
      onChangeDate={handleChangeDate}
      isLoading={isLoading}
      mediaList={allItems}
      loadMoreRef={loadMoreRef}
      isFetchingNextPage={isFetchingNextPage}
    />
  );
}

```

Rysunek 7.104: Komponent Movies.

Movies Komponent reprezentuje pojedynczy film. Do odtwarzania wykorzystano bibliotekę **react-player** oraz zestaw komponentów kontrolnych z **biblioteki media-chrome**. Stan odtwarzania i ewentualnego błędu zarządzany jest za pomocą customowego hooka **useBoolean**. Po zakończeniu odtwarzania film jest zatrzymywany, a w razie problemu z odczytem wyświetlany jest komunikat: "Failed to load the video. Please try again later.". Na filmie wyświetlany jest także pasek z liczbą polubień i wyświetleń.

7.3.9.5 Social

Zakładka **Social** odpowiada za obsługę funkcji społecznościowych: znajomych, obserwowanych, obserwujących oraz (w widoku innego użytkownika) zdjęć tego użytkownika. Składa się z następujących głównych komponentów:

- `UserOwnSocial`,
- `SocialForViewer`,
- `Social`,
- `SocialCardList`,
- `SocialCard`,
- `SocialButton`,
- `SearchFriendsList`,
- `PhotosList`,
- `FriendInvitesList`.

`UserOwnSocial` (rys. 7.105 i 7.106) Komponent odpowiada za wyświetlanie list społecznościowych zalogowanego użytkownika.

Aktualnie wybrany typ listy (FRIENDS, FOLLOWED, FOLLOWERS) jest pobierany ze store `Reduxa` za pomocą `useSelectorTyped`. W celu uniknięcia powielania logiki zapytań użyto mapowania `queryConfigMap`, które łączy typ listy z odpowiednią funkcją pobierającą.

Dane pobierane są za pomocą `useInfiniteQuery`, a kolejne strony ładowane przy wykorzystaniu `IntersectionObserver`. Po zebraniu wszystkich stron dane są spłaszczone i przekazywane do uniwersalnego komponentu `Social` z parametrem `isSocialForViewer = false`.

```

type SupportedSocialType =
  | SocialListType.FRIENDS
  | SocialListType.FOLLOWED
  | SocialListType.FOLLOWERS;

const queryConfigMap: Record<
  SupportedSocialType,
  { key: string; fn: (page: number, size: number) => Promise<SocialPageDto> }
> = {
  [SocialListType.FRIENDS]: {
    key: "friends",
    fn: getUserOwnFriends,
  },
  [SocialListType.FOLLOWED]: {
    key: "followed",
    fn: getUserOwnFollowed,
  },
  [SocialListType.FOLLOWERS]: {
    key: "followers",
    fn: getUserOwnFollowers,
  },
};

export default function UserOwnSocial(): Element {
  const type = useSelectorTyped(
    (state: { account: AccountSliceProps; notifications: any }) : SocialListType => state.social.type,
  ) as SupportedSocialType;
  const loadMoreRef: MutableRefObject<HTMLDivElement | null> = useRef<HTMLDivElement | null>(null);
  const { key, fn } = queryConfigMap[type];

  const { data, isLoading, fetchNextPage, hasNextPage, isFetchingNextPage } =
    useInfiniteQuery({
      queryKey: [key],
      queryFn: ({ pageParam = 0 }: { client: QueryClient; queryKey: string[] }) : Promise<SocialPageDto> => fn(pageParam, 12),
      getNextPageParam: (lastPage: SocialPageDto, allPages: SocialPageDto[]) : number | undefined =>
        lastPage.hasNext ? allPages.length : undefined,
      initialPageParam: 0,
    });
}

const allItems: SocialDto[] | undefined = data?.pages.flatMap((page: SocialPageDto) : SocialDto[] => page.items);

```

Rysunek 7.105: Komponent Social (1).

```

useEffect(() : () => void | undefined => [...], [hasNextPage, isFetchingNextPage, fetchNextPage]);

if (isLoading) {
  return <LoadingSpinner />;
}

return (
  <Social
    list={allItems ?? []}
    isSocialForViewer={false}
    loadMoreRef={loadMoreRef}
    isFetchingNextPage={isFetchingNextPage}
    type={type}
  />
);
}

```

Rysunek 7.106: Komponent Social (2).

`SocialForViewer` Komponent realizuje analogczną funkcjonalność, ale dla profilu innego użytkownika. Nazwa użytkownika pobierana jest z parametrów ścieżki (`useParams`), a dane list społecznościowych oraz zdjęć tego użytkownika – z odpowiednich `endpointów` powiązanych w rozszerzonym `queryConfigMap`.

W zależności od typu listy pobierane są dane użytkowników (`SocialDto`) lub zdjęć (`DatedMediaGroup`). Po spłaszczeniu danych całość przekazywana jest do komponentu `Social` z parametrem `isSocialForViewer = true`.

`Social` (rys. 7.107) Jest to główny, uniwersalny komponent wykorzystywany zarówno dla widoku własnego, jak i cudzego profilu. Widok opakowany jest w `AccountWrapper` z wariantem `SOCIAL`, a nagłówek sekcji wyświetlany przez `AccountTitle` z tekstem "social list".

W przypadku widoku własnego i wybranego typu `FRIENDS` po prawej stronie nagłówka pojawiają się dwa przyciski:

- "See friend invites" – otwiera modal z listą zaproszeń (`FriendInvitesList`);
- "Add new friend" – otwiera modal z wyszukiwarką znajomych (`SearchFriendsList`).

Poniżej wyświetlane jest menu z przyciskami `SocialButton`, które pozwalają przełączać się pomiędzy listą znajomych, obserwowanych, obserwujących oraz (w widoku innego użytkownika) listą zdjęć. Zmiana typu listy zapisywana jest w `store Reduxa` za pomocą akcji `socialAction.setType`.

Dalsza część widoku zależy od typu listy:

- dla `PHOTOS` używany jest komponent `PhotosList`;
- dla pozostałych typów – komponent `SocialCardList`.

Na dole listy znajduje się element powiązany z `loadMoreRef`, obsługujący *infinite scroll*, oraz `LoadingSpinner` wyświetlany podczas pobierania kolejnych stron.

```

<>
  <AccountWrapper variant={AccountWrapperType.SOCIAL}>
    <div className="flex items-center justify-between">
      <AccountTitle text="social list" />
      <div className="flex items-center gap-x-5" ...>
    </div>
    <div className="flex gap-3 text-2xl lg:mx-27">
      {menuTypes.map(({ label, type: btnType }) : {label: string; type: SocialListType} | Element | => (
        <SocialButton
          key={label}
          onClick={() : void => setType(btnType)}
          isActive={type === btnType}
        >
          <p className="font-semibold capitalize">{label}</p>
        </SocialButton>
      ))}
    </div>
    {type === SocialListType.PHOTOS ? (
      <PhotosList photos={list as DatedMediaGroup[]} />
    ) : (
      <SocialCardList
        list={list as SocialDto[]}
        type={type}
        isSocialForViewer={isSocialForViewer}
      />
    )}
    <div ref={loadMoreRef} className="h-10" />
    {isFetchingNextPage && <LoadingSpinner />}
  </AccountWrapper>
  <EmptyModal onClose={close} isOpen={isOpen} className="h-4/5 w-4/5">
    <SearchFriendsList onClose={close} />
  </EmptyModal>
  <EmptyModal
    onClose={closeInvites}
    isOpen={isOpenInvites}
    className="h-96 w-96"
  >
    <FriendInvitesList onClose={closeInvites} />
  </EmptyModal>
</>

```

Rysunek 7.107: Komponent Social.

SocialCardList Komponent wyświetla listę kafelków SocialCard oraz prezentuje odpowiednie komunikaty w przypadku pustych list. Treść komunikatu zależy od typu listy oraz tego, czy jest to profil własny, czy innego użytkownika.

nika. Dla trybu wyszukiwania znajomych (`isSearchFriend`) wyświetlany jest komunikat: "We can't find a user with this username.". W przypadku typu `POTENTIAL_GROUP_CHAT_MEMBER` stosowany jest komunikat: "There are no other people in the world besides you.".

`SocialCard` Komponent reprezentuje pojedynczego użytkownika na listach społecznościowych. Wyświetlane są: zdjęcie profilowe, nazwa użytkownika oraz zestaw przycisków akcji które służą do przejścia do profilu, otwarcia prywatnego czatu oraz dodania lub usunięcia z listy znajomych bądź obserwowanych.

Relacje znajomości i obserwowania modyfikowane są za pomocą `mutacji` po stronie `API`, odpowiedzialnych za zarządzanie listą znajomych oraz obserwowanych. Po ich powodzeniu odświeżane są odpowiednie listy. Otwarcie lub utworzenie prywatnego czatu realizowane jest przez osobną `mutację`, która zwraca obiekt czatu; następnie jest on zapisywany w store `Reduxa`, ustawiany jako aktualnie wybrany, a aplikacja przełącza widok na `/chat`.

Przed usunięciem znajomego lub zakończeniem obserwowania wyświetlany jest modal potwierdzenia, którego stan zarządzany jest przez `hook` `useBoolean`. Treść komunikatu ma postać: "Are you sure you want to remove {username} as a friend?" lub

"Are you sure you want to remove {username} as a follower?".

`SocialButton` Jest to uniwersalny przycisk wykorzystywany jako element nawigacji w `Social` oraz jako przycisk akcji w `SocialCard`. Umożliwia wyróżnienie przycisku aktywnego oraz dostosowanie szerokości do treści.

`SearchFriendsList` Komponent odpowiada za wyszukiwanie potencjalnych znajomych. Wykorzystywany jest w modalu otwieranym z poziomu sekcji `Social`. Zapytanie tekstowe jest `debounceowane` za pomocą `hooka` `useDebounce`, a dane pobierane są z użyciem `useInfiniteQuery`, które wysyła zapytania wyszukujące użytkowników po nazwie. Interfejs zawiera przycisk zamknięcia, nagłówek, pole wyszukiwania oraz listę wyników w postaci `SocialCardList`.

`PhotosList` Komponent wyświetla zdjęcia innego użytkownika, gdy z widoku profilu następuje przejście do listy jego zdjęć (por. sekcja [7.3.9.1](#)). Zdjęcia grupowane są po dacie; dla każdej daty wyświetlany jest `DateBadge`, a poniżej siatka

zdjęć z wykorzystaniem komponentu Photo. W przypadku braku zdjęć wyświetlany jest komunikat: "This user hasn't added any photos.".

FriendInvitesList Komponent służy do wyświetlania listy zaproszeń do znajomych otrzymanych przez użytkownika.

Dane pobierane są za pomocą `useInfiniteQuery`, a kolejne strony ładowane są przy użyciu `IntersectionObserver`. Zmiana statusu zaproszenia (zaakceptowanie lub odrzucenie) realizowana jest przez `mutację`, która wysyła odpowiednie żądanie do `backendu`. Interfejs zawiera przycisk zamknięcia, nagłówek, listę zaproszeń oraz przyciski akceptacji i odrzucenia dla każdego elementu. Kliknięcie w awatar użytkownika powoduje przejście do jego profilu. Jeżeli brak zaproszeń, wyświetlany jest komunikat: "You don't have any invites yet".

7.3.9.6 Add spot

Zakładka `Add spot` służy do zarządzania `spotami` dodanymi przez użytkownika oraz do dodawania nowych `spotów`. Składa się z następujących komponentów:

- `AddedSpot`,
- `UploadButton`,
- `SpotMap`,
- `PolygonDrawer`,
- `AddSpotModal`,
- `AddSpotInput`,
- `AddedSpotTile`.

Dodatkowo wykorzystywany jest schemat walidacji `spotSchema` utworzony z użyciem `biblioteki zod`.

`AddedSpot` (rys. 7.108) Jest to główny komponent zakładki. Odpowiada za wyświetlenie listy spotów dodanych przez użytkownika oraz za otwieranie `okna modalnego` `AddSpotModal` służącego do dodania nowego `spota`. Dane pobierane są

za pomocą `useInfiniteQuery`, które pobiera kolejne strony wyników z `backendu`, a mechanizm `infinite scroll` zaimplementowano przy użyciu `IntersectionObserver`.

Stan widoczności modala zarządzany jest przez `hook useState`. Dodatkowo sprawdzana jest szerokość okna przeglądarki; formularz dodawania `spota` dostępny jest tylko na ekranach o szerokości co najmniej 900 px. W przypadku niespełnienia tego warunku wyświetlany jest komunikat informacyjny w komponentie `Notification`.

Widok opakowany jest w `AccountWrapper` z wariantem `ADD_SPOT` oraz nagłówek `AccountTitle`. Jeżeli użytkownik nie dodał żadnych `spotów`, wyświetlany jest komunikat: "You haven't added any spots yet.", w przeciwnym razie renderowana jest lista kafelków `AddedSpotTile`. Podczas ładowania danych prezentowany jest `LoadingSpinner`.

```

<>
<AccountWrapper variant={AccountWrapperType.ADD_SPOT}>
  <div className="flex items-center justify-between space-y-2 space-x-3 lg:mr-27">
    <AccountTitle text="Add spot" />
    <Button
      variant={ButtonVariantType.ADD_SPOT}
      onClick={handleOpenModal}
    >
      Add spot
    </Button>
  </div>

  {isLoading && <LoadingSpinner />}

  <div className="flex flex-col items-center space-y-5 lg:mx-27">
    {allItems?.length
      ? allItems?.map((addedSpot : AddSpotDto) : Element => (
          <AddedSpotTile
            key={addedSpot.id}
            spot={addedSpot}
          />
        ))
      : !isLoading && (
        <p className="mt-10 text-center text-gray-500">
          You haven't added any spots yet.
        </p>
      )}
  </div>

  <div ref={loadMoreRef} className="h-10" />
  {isFetchingNextPage && <LoadingSpinner />}
</AccountWrapper>
{!isMobile && <AddSpotModal onClose={close} isOpen={isOpen} />}
</>

```

Rysunek 7.108: Komponent AddedSpot.

UploadButton Komponent służy do wyboru i podglądu multimedialów (zdjęć i filmów) dodawanych do **spota**. W lokalnym stanie utrzymywana jest lista wybranych plików wraz z tymczasowymi adresami tworzonymi za pomocą `URL.createObjectURL`. Użytkownik może usuwać pojedyncze elementy listy, a przy usuwaniu zwalniane są zasoby (`URL.revokeObjectURL`). Dozwolone są wy-

brane typy plików graficznych oraz wideo.

SpotMap Komponent wyświetla miniaturową mapę z zaznaczoną lokalizacją spota. Wykorzystywana jest [biblioteka maplibregl](#) oraz przygotowany styl mapy `map_style.json`. Na mapie umieszczany jest marker w pozycji [spota](#).

PolygonDrawer Komponent umożliwia interaktywne narysowanie konturu [spota](#) na mapie. Zbudowany jest w oparciu o [@vis.gl/react-maplibre](#). Kliknięcia w mapę dodają kolejne wierzchołki, a po dodaniu co najmniej trzech punktów prezentowany jest zamknięty wielokąt. Użytkownik może cofnąć ostatni punkt lub zatwierdzić wielokąt, przekazując współrzędne do komponentu nadziednego. W razie błędu walidacji konturu wyświetlany jest komunikat pod mapą.

AddSpotModal Komponent odpowiada za dodawanie nowego [spota](#). W stanie przechowywany jest obiekt z danymi [spota](#), komunikaty błędów walidacyjnych oraz aktualna pozycja mapy. Adres budowany jest z pól formularza (ulica, miasto, region, kraj) i [debouncowany](#) za pomocą [hooka useDebounce](#). Na podstawie pełnego adresu [hook useQuery](#) pobiera współrzędne geograficzne, a mapa przesuwana jest do odpowiedniej lokalizacji.

Walidacja danych formularza realizowana jest przy użyciu `spotSchema`. W przypadku niepowodzenia walidacji wypełniane są pola błędów oraz wyświetlany jest komunikat: "Please fill in all fields correctly". Po poprawnej walidacji wywoływana jest [mutacja](#) odpowiedzialna za dodanie [spota](#) po stronie [backendu](#). Po jej powodzeniu [modal](#) jest zamykany, formularz resetowany, dane listy odświeżane, a użytkownik otrzymuje komunikat o poprawnym dodaniu [spota](#).

Wyświetlenie [modala](#) zrealizowano za pomocą `createPortal`, co umożliwia umieszczenie go w dedykowanym elemencie `DOM` (`<div id="modal">`). Za animacje otwierania i zamykania odpowiadają komponenty `AnimatePresence` oraz `motion.div` z [biblioteki motion](#).

AddSpotInput Komponent odpowiada za wyświetlanie i obsługę listy pól formularza. Korzysta z lokalnego [stanu](#) błędów oraz schematu `spotSchema` (za pomocą `pick`) do walidacji pojedynczych pól. Do prezentacji pól wykorzystuje uniwersalny komponent `FormInput`.

AddedSpotTile Komponent wyświetla pojedynczy kafelek z informacjami o do-

danym [spocie](#). Prezentowane są: zdjęcie spota, mini mapa (SpotMap) z lokalizacją, nazwa, opis oraz adres (kraj, miasto, ulica).

7.3.9.7 Comments

Zakładka **Comments** wyświetla wszystkie komentarze dodane przez użytkownika pod [spotami](#). Składa się z dwóch głównych komponentów:

- **Comments**,
- **CommentsList**.

Comments (rys. 7.109) Komponent korzysta z [hooka useDateSortFilter](#), analogicznie jak zakładki **Photos** i **Movies**. Dane pobierane są za pomocą [useInfiniteQuery](#), które pobiera kolejne strony komentarzy z [backendu](#), a [infinite scroll](#) realizowany jest przez [IntersectionObserver](#).

W przypadku błędu podczas pobierania komentarzy w [hooku useEffect](#) ustalany jest komunikat błędu, wyświetlany przez [Notification](#). Widok opakowany jest w [AccountWrapper](#) z wariantem **COMMENTS** i nagłówkiem [AccountTitle](#). Poniżej znajduje się panel filtrów [SortAndDateFilters](#), a następnie lista komentarzy pogrupowanych po dacie i nazwie [spota](#).

Dla każdej grupy wyświetlany jest [DateBadge](#) z datą oraz informacją o [spocie](#), a poniżej lista komentarzy renderowana przez [CommentsList](#). Jeżeli użytkownik nie dodał żadnych komentarzy, wyświetlany jest komunikat: "You haven't added any comments.". Podczas ładowania danych oraz kolejnych stron prezentowany jest [LoadingSpinner](#).

```

<AccountWrapper variant={AccountWrapperType.COMMENTS}>
  <div className="flex flex-wrap items-center justify-between space-y-2 space-x-3">
    <AccountTitle text="Comments" />
    <SortAndDateFilters
      onSortChange={handleSelectType}
      onDateChange={handleChangeDate}
      from={searchDate.from}
      to={searchDate.to}
    />
  </div>
  {isLoading && <LoadingSpinner />}
  <ul className="space-y-5 md:mx-27">
    {allItems?.length
      ? allItems?.map((d : DatedCommentsGroup) : Element => (
        <li
          key={`${d.spotName}-${d.date}`}
          className="flex flex-col space-y-5"
        >
          <DateBadge date={d.date}>
            <span className="flex flex-wrap">
              Comments to
              <p className="text-violetLight mx-2 font-semibold">
                {d.spotName}
              </p>
              spot
            </span>
          </DateBadge>
          <CommentsList comments={d.comments} />
        </li>
      ))
      : null
    {!allItems?.length && !isLoading ? (
      <p className="text-center text-lg">
        You haven't added any comments.
      </p>
    ) : null}
    <div ref={loadMoreRef} className="h-10" />
    {isFetchingNextPage && <LoadingSpinner />}
  </ul>
</AccountWrapper>

```

Rysunek 7.109: Komponent Comments.

`CommentsList` Komponent otrzymuje listę komentarzy i wyświetla je w postaci kafelków zawierających godzinę dodania oraz treść komentarza.

7.3.9.8 Settings

Zakładka `Settings` umożliwia użytkownikowi edycję podstawowych danych konta. Została zrealizowana za pomocą dwóch głównych komponentów:

- `Settings`,
- `DisableInput`.

`Settings` (rys. 7.110) Komponent zarządza procesem zmiany danych użytkownika oraz wyświetlaniem odpowiednich formularzy. Typ danych podlegających edycji (`USERNAME`, `EMAIL`, `PASSWORD`) przechowywany jest w stanie (`editType`).

Na początku, za pomocą `hooka useQuery`, pobierane są dane użytkownika (nazwa, e-mail, dostawca logowania). Dopóki dane są ładowane, wyświetlany jest `LoadingSpinner`. Podstawowe informacje prezentowane są przez komponenty `DisableInput`, które wyświetlają aktualne wartości oraz przycisk "Edit".

Po wyborze rodzaju edycji po prawej stronie pojawia się formularz dopasowany do aktualnego `editType`. Walidacja realizowana jest przy użyciu schematów `baseSchemas` zdefiniowanych w `bibliotece zod`, a obsługę formularza zapewnia `react-hook-form` w połączeniu z `zodResolver`. Dane wysyłane są do `backendu` za pomocą `mutacji`, która aktualizuje odpowiednie ustawienia użytkownika. W przypadku błędu prezentowany jest komunikat z odpowiedzi `backendu`; po sukcesie wyświetlany jest komunikat o poprawnej zmianie ustawień oraz odświeżane są dane użytkownika.

Dla kont utworzonych przez formularz rejestracji (`Provider.NONE`) dostępna jest edycja nazwy użytkownika, adresu e-mail oraz hasła. Jeżeli konto zostało utworzone za pomocą zewnętrznego dostawcy (Google, GitHub), wyświetlany jest komunikat: "Your account was created via (Google/Github)" oraz "Your email address is (...) and cannot be changed.", a formularze edycyjne są wówczas ukryte.

```

<AccountWrapper variant={AccountWrapperType.SETTINGS}>
  <AccountTitle text="Settings" />
  {data?.provider === Provider.NONE && (
    <div className="flex flex-col space-y-6 lg:flex-row lg:space-y-0">
      <div className="mt-5 flex w-full flex-col space-y-4 lg:ml-27 lg:w-1/3">
        <h1 className="text-3xl font-semibold">
          Account details
        </h1>
        <div className="flex w-full flex-col space-y-3">
          {inputFields.map((field :{label: string; value: string | undefined}) : Element => ...)}
        </div>
      </div>
      {editType !== UserSettingsType.NONE && (
        <div className="mt-5 flex w-full flex-col space-y-4 lg:mx-27 xl:mr-0 xl:w-1/2">
          <h1 className="text-3xl font-semibold">
            {getHeaderForEditType(editType)}
          </h1>
          <div className="flex w-full flex-col space-y-3 xl:w-1/2">
            <form
              className="flex w-full flex-col gap-4"
              onSubmit={handleSubmit(onSubmit)}>
              ...
            </div>
          </div>
        )})
      </div>
    )}
  )
  {data?.provider !== Provider.NONE && (
    <div className="flex flex-col items-center rounded-md p-3 text-center text-lg">
      <div className="mb-2 flex space-x-3">
        <p>Your account was created via {data?.provider}</p>
        {getOAuth2ProviderIcon(data?.provider)}
      </div>
      <p>
        Your email address is {data?.email} <br /> and cannot be
        changed.
      </p>
    </div>
  )}
</AccountWrapper>

```

Rysunek 7.110: Komponent `Settings`.

`DisableInput` Komponent prezentuje pojedyncze pole z aktualną wartością danej informacji (nazwy użytkownika, e-mail lub hasła) w trybie tylko do odczytu

oraz przyciskiem "Edit". Dla pola hasła wyświetlane są jedynie gwiazdki. Logika wyboru rodzaju edycji pozostaje w komponencie nadziednym.

7.3.9.9 Komponenty wspólne

Panel użytkownika wykorzystuje także zestaw komponentów uniwersalnych, z których część była już wspomniana w poprzednich sekcjach. Dla przejrzystości poniżej zebrano ich krótkie opisy.

AccountWrapper (rys. 7.111) Komponent odpowiada za wspólny układ i podstawowe stylowanie poszczególnych zakładek panelu. Przyjmuje wariant widoku i na jego podstawie stosuje odpowiednie klasy [Tailwind](#), dzięki czemu zachowana jest spójność wizualna przy jednoczesnej możliwości dostosowania przestrzeni danej sekcji.

```

interface AccountWrapperProps { Show usages ▾ Mredosz
  children: ReactNode;
  variant: AccountWrapperType;
}

const baseClasses =
  "dark:bg-darkBg bg-lightBg dark:text-darkText text-lightText w-full flex flex-col";

const socialCommentsSettingsAddSpot = "h-full space-y-8 p-10 pt-17 xl:pt-10";

const variantClasses = {
  photos: "min-h-full space-y-8 p-2 pt-20 md:p-10 md:pt-20 xl:pt-10",
  profile: "min-h-full items-center gap-20 p-6 lg:justify-center xl:p-0",
  favorite_spots: "h-full space-y-10 p-10 pt-17",
  social: socialCommentsSettingsAddSpot,
  comments: socialCommentsSettingsAddSpot,
  settings: socialCommentsSettingsAddSpot,
  movies: "min-h-full space-y-8 p-2 pt-20 md:p-10 md:pt-20 xl:pt-10",
  add_spot: socialCommentsSettingsAddSpot,
};

export default function AccountWrapper({ Show usages ▾ Mredosz
  children,
  variant,
}: AccountWrapperProps): Element {
  return (
    <div className={`${baseClasses} ${variantClasses[variant]}`}>
      {children}
    </div>
  );
}

```

Rysunek 7.111: Komponent AccountWrapper.

AccountTitle (rys. 7.112) Komponent wyświetla nagłówek danej sekcji jako element **h1**. Dodatkowo ustawia atrybut **data-testid**, co ułatwia testowanie widoków.

```
interface AccountTileProps { Show usages & Mredosz
  text: string;
}

export default function AccountTitle({ text }: AccountTileProps) : Element { Show usages & Mredosz
  return (
    <h1
      data-testid={text}
      className="text-center text-4xl font-semibold capitalize lg:ml-27 lg:text-start"
    >
      {text}
    </h1>
  );
}
```

Rysunek 7.112: Komponent AccountTitle.

SortDropdown Komponent udostępnia możliwość wyboru typu sortowania po dacie (malejąco/rosnąco). **Stan** rozwinięcia listy zarządzany jest przez **hook useBoolean**, a animacje zrealizowano za pomocą **biblioteki framer-motion**. Komponent używany jest pośrednio poprzez **SortAndDateFilters**.

DateChooser (rys. 7.113) Jest to wrapper wokół komponentu **DatePicker** z **biblioteki** Ant Design. Służy do wyboru daty w polach "From" oraz "To" w panelu filtrowania.

```
export default function DateChooser({ Show usages & Mredosz
  text,
  onChange,
  value,
}: DateChooserProps): Element {
  return (
    <div className="flex flex-col items-center sm:flex-row">
      <p>{text}</p>
      <ConfigProvider
        theme={{
          token: {
            colorPrimary: "#363041",
            colorBgContainer: "#363041",
            colorText: "#e5e5e5",
            borderRadius: 8,
          },
          components: {
            DatePicker: {
              colorBorder: "transparent",
              colorBgContainer: "#363041",
              colorTextPlaceholder: "#e5e5e5",
              colorTextDisabled: "#939394",
              colorBgElevated: "#363041",
            },
          },
        }}
      >
        <DatePicker
          onChange={onChange}
          value={value}
          style={{
            border: "none",
            boxShadow: "none",
            backgroundColor: "#363041",
            color: "#e5e5e5",
          }}
        />
      </ConfigProvider>
    </div>
  );
}
```

SortAndDateFilters (rys. 7.114) Komponent łączy SortDropdown oraz dwa DateChooser, tworząc panel sterujący sortowaniem i filtrowaniem po dacie. Wykorzystywany jest w zakładkach Photos, Movies oraz Comments.

```
interface SortAndDateFiltersProps { Show usages ▾ Mredosz
  onSortChange: (type: DateSortType) => void;
  onDateChange: (value: Dayjs | null, id: "from" | "to") => void;
  from: string | null;
  to: string | null;
}

export default function SortAndDateFilters({ Show usages ▾ Mredosz
  onSortChange,
  onDateChange,
  from,
  to,
}: SortAndDateFiltersProps): Element {
  return (
    <div className="text-darkText flex flex-wrap space-y-3 space-x-3 md:space-y-0 lg:mx-27">
      <SortDropdown onSelectType={onSortChange} />
      <div className="bg-violetDark flex h-15 items-center space-x-3 rounded-full px-2.5 py-1 md:h-12">
        <DateChooser
          text="From:"
          onChange={(val: Dayjs | null) : void => onDateChange(val, "from")}
          value={from ? dayjs(from) : null}
        />
        <DateChooser
          text="To:"
          onChange={(val: Dayjs | null) : void => onDateChange(val, "to")}
          value={to ? dayjs(to) : null}
        />
      </div>
    </div>
  );
}
```

Rysunek 7.114: Komponent SortAndDateFilters.

DateBadge (rys. 7.115) Komponent służy do czytelnej prezentacji daty (formatowanej zgodnie z lokalizacją "pl-PL") oraz opcjonalnej dodatkowej informacji przekazywanej jako children. Wykorzystywany jest do grupowania zdjęć, filmów oraz komentarzy po dacie.

```

interface DateBadgeProps { Show usages & Mredosz
  date: string;
  children?: ReactNode;
}

export default function DateBadge({ date, children }: DateBadgeProps): Element { Show usages & Mredosz *
  return (
    <div className="dark:bg-darkBgMuted bg-lightBgMuted flex w-full flex-wrap space-x-4 rounded-md px-2 py-1.5 text-xl">
      <p className="font-semibold">
        {new Date(date).toLocaleDateString("pl-PL")}
      </p>
      {children}
    </div>
  );
}

```

Rysunek 7.115: Komponent DateBadge.

Komponent `LoadingSpinner` jest wspólnym komponentem informującym o trwającym ładowaniu danych i pojawia się w większości widoków w stanach `isLoading` oraz `isFetchingNextPage`.

`Notification` (przywoływany pośrednio poprzez akcje `Reduxa notificationAction.addSuccess, addError oraz addInfo`) odpowiada za prezentację komunikatów informacyjnych, błędów oraz potwierdzeń operacji.

7.3.10 Panel logowania

W niniejszym rozdziale przedstawiono implementację panelu logowania po stronie [frontendu](#). Panel składa się z czterech głównych części:

- formularza logowania,
- formularza rejestracji,
- formularza resetu hasła,
- formularza ustawiania nowego hasła.

Każda z opisanych poniżej sekcji korzysta z dwóch uniwersalnych komponentów: `FormContainer` oraz `FormInput`, które zapewniają spójny układ, walidację

i obsługę komunikatów. Dodatkowo, w niektórych formularzach wykorzystywany jest komponent `SubmitFormButton` do obsługi wysyłania danych.

7.3.10.1 Logowanie

W skład formularza logowania wchodzą następujące komponenty:

- `Login`,
- `FormContainer`,
- `FormInput`,
- `SubmitFormButton`.

`Login` (rys. 7.116 i 7.117) Komponent służy do logowania w aplikacji za pomocą nazwy użytkownika oraz hasła albo z wykorzystaniem zewnętrznych dostawców (Google i GitHub). Dane logowania są obsługiwane przez hook `useMutation`, który wysyła żądanie do `backendu`. Po poprawnym zalogowaniu następuje przekierowanie na ostatnio odwiedzoną podstronę przed przejściem do formularza, realizowane za pomocą `useNavigate`. Równocześnie, przy użyciu `Reduxa` (akcje `accountAction`), w store aplikacji zapisywana jest informacja, że użytkownik został poprawnie zalogowany oraz jaka jest jego nazwa.

Validacja pól formularza opiera się na hooku `useUserDataValidation`, który przechowuje wartości pól, informacje o tym, czy zostały one edytowane, oraz wynik walidacji. Na tej podstawie przycisk potwierdzający logowanie jest aktywowany dopiero po poprawnym uzupełnieniu wymaganych pól.

Główny układ formularza zapewnia komponent `FormContainer`, wewnątrz którego renderowana jest lista dwóch pól wejściowych `FormInput` (`username` i `password`). Poniżej pól wyświetlany jest link prowadzący do formularza resetu hasła, a na samym dole znajduje się przycisk potwierdzający logowanie w postaci komponentu `SubmitFormButton`.

```

const signInFields :string[] = ["username", "password"];

function Login() :Element { Show usages & Mredosz +1
  const dispatch :ThunkDispatch<{account: AccountSliceProp...} = useDispatchTyped();
  const navigate :NavigateFunction = useNavigate();

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

  const {
    enteredValue,
    didEdit,
    isValid,
    handleInputChange,
    handleInputBlur,
  } = useUserDataValidation({ username: "", password: "" }, false);

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

  useEffect(() : void => {
    if (isSuccess) {
      dispatch(accountAction.setIsLogged());
      dispatch(accountAction.setUsername(enteredValue.username));
    }
  }, [isSuccess, dispatch, enteredValue.username]);
}

```

Rysunek 7.116: Komponent Login (1).

```

return (
  <FormContainer
    error={error}
    isSuccess={isSuccess}
    navigateTo="/register"
    linkCaption="Don't have an account?"
    header="Sign in"
    navigateOnSuccess="/"
    showOauth={true}
    showLink={true}
    notificationMessage="Signed in successfully!"
  >
  <form onSubmit={handleSubmit} className="flex flex-col gap-4">
    {[inputs[0], inputs[2]].map(({ name, type, id }: InputsProps) : Element => (
      <FormInput
        key={id}
        label={name}
        type={type}
        id={id}
        onChange={(event : ChangeEvent<HTMLInputElement>} : void => handleInputChange(id, event)}
        value={enteredValue[id]}
        onBlur={() : void => handleInputBlur(id)}
        isValid={isValid[id]}
      />
    )));
    <div className="flex justify-between">
      <Link
        to="/forgot-password"
        className="text-sm text-blue-700 hover:underline"
      >
        Forgot Password?
      </Link>
    </div>
    <SubmitFormButton
      label="sign in"
      fields={signInFields}
      didEdit={didEdit}
      isValid={isValid}
    />
  </form>
</FormContainer>
);

```

Rysunek 7.117: Komponent Login (2).

7.3.10.2 Rejestracja

W skład formularza rejestracji wchodzą następujące komponenty:

- Register,

- `FormContainer`,
- `FormInput`,
- `SubmitFormButton`.

Register (rys. 7.118 i 7.119) Komponent służy do rejestracji nowego użytkownika z wykorzystaniem formularza zawierającego nazwę użytkownika, adres e-mail oraz hasło (z potwierdzeniem), a także umożliwia skorzystanie z logowania przy pomocy kont Google lub GitHub. Wygląd formularza jest zbliżony do widoku logowania: wykorzystywany jest ten sam układ `FormContainer`, jednak w środku renderowane są cztery pola `FormInput` zamiast dwóch.

Proces rejestracji obsługiwany jest przez `useMutation`, która wysyła dane nowego konta do `backendu`. Logika walidacji korzysta z `useUserDataValidation`, dzięki czemu przycisk potwierdzenia rejestracji (`SubmitFormButton`) aktywuje się dopiero po poprawnym uzupełnieniu wszystkich wymaganych pól, w tym zgodności hasła z jego potwierdzeniem. Po pomysłnej rejestracji w store `Reduxa` zapisywana jest informacja o zalogowaniu użytkownika, a `FormContainer` wyświetla odpowiedni komunikat potwierdzający.

```
const signUpFields :string[] = ["username", "password", "email", "confirm-password"];
```

```
export default function Register(): Element { Show usages ▾ Mredosz +1
  const { mutateAsync, isSuccess, error } = useMutation({
    mutationFn: registerUser,
  });

  const dispatch: ThunkDispatch<{ account: AccountSliceProp... }> = useDispatchTyped();

  const {
    enteredValue,
    didEdit,
    isValid,
    handleInputChange,
    handleInputBlur,
  } = useUserDataValidation({
    password: "",
    username: "",
    email: "",
    "confirm-password": "",
  });

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

  useEffect(() : void => {
    if (isSuccess) {
      dispatch(accountAction.setIsLogged());
    }
  }, [isSuccess, dispatch, enteredValue.username]);
}
```

Rysunek 7.118: Komponent Register (1).

```

return (
  <FormContainer
    isSuccess={isSuccess}
    error={error}
    navigateTo="/login"
    linkCaption="Already have an account?"
    header="Create Account"
    navigateOnSuccess="/"
    showOauth={true}
    showLink={true}
    notificationMessage="Account created successfully!"
  >
  <form className="flex flex-col gap-4" onSubmit={handleSubmit}>
    {inputs.map(({ name, type, id }: InputsProps) : Element => (
      <FormInput
        key={id}
        label={name}
        type={type}
        id={id}
        onChange={(event : ChangeEvent<HTMLInputElement>} : void => handleInputChange(id, event)}
        value={enteredValue[id]}
        onBlur={() : void => handleInputBlur(id)}
        isValid={isValid[id]}
      />
    )));
    <SubmitFormButton
      label="sign up"
      fields={signUpFields}
      didEdit={didEdit}
      isValid={isValid}
    />
  </form>
</FormContainer>
);
}

```

Rysunek 7.119: Komponent Register (2).

7.3.10.3 Reset hasła

W skład formularza resetu (przypomnienia) hasła wchodzą następujące komponenty:

- `ForgotPassword`,
- `FormContainer`,
- `FormInput`.

ForgotPassword (rys. 7.120) Komponent służy do inicjowania procesu resetu hasła w sytuacji, gdy zostało ono zapomniane. Po podaniu adresu e-mail, z którym powiązane jest konto, i zatwierdzeniu formularza wysyłane jest żądanie do **backendu** z użyciem **useMutation**. Na tej podstawie generowana jest wiadomość e-mail zawierająca odnośnik do formularza ustawiania nowego hasła.

Validacja pola adresu e-mail realizowana jest przez **useUserDataValidation**. Przycisk potwierdzający wysłanie wiadomości pozostaje nieaktywny, dopóki adres e-mail nie zostanie poprawnie wprowadzony. Układ formularza oparty jest na **FormContainer**; wewnątrz znajduje się pojedyncze pole **FormInput** dla adresu e-mail oraz przycisk potwierdzenia. Po pomyślnym wysłaniu żądania **FormContainer** wyświetla komunikat potwierdzający wysłanie wiadomości.

```

export default function ForgotPassword() { Show usages ⌘ stanoz +3
  const { mutate, isSuccess, error } = useMutation({
    mutationFn: sentEmailWithNewPasswordLink,
  });

  const {
    enteredValue,
    didEdit,
    isValid,
    handleInputChange,
    handleInputBlur,
  } = useUserDataValidation({ email: "" });

  const handleSubmit = async (event) => {...};

  return (
    <FormContainer
      isSuccess={isSuccess}
      error={error}
      header="Forgot your password?"
      notificationMessage="Reminder email sent!"
    >
      <form onSubmit={handleSubmit} className="flex flex-col gap-y-5">
        <FormInput
          id="email"
          value={enteredValue.email}
          onBlur={() => handleInputBlur("email")}
          onChange={(event) => handleInputChange("email", event)}
          type="email"
          isValid={isValid?.email}
          label="email"
        />
        <button
          type="submit"
          className="mt-2 mb-2 w-full rounded-lg bg-black p-1 text-white"
          disabled={!didEdit.email || !isValid.email.value}
        >
          Remind me
        </button>
      </form>
    </FormContainer>
  );
}

```

Rysunek 7.120: Komponent ForgotPassword.

7.3.10.4 Nowe hasło

W skład formularza ustawiania nowego hasła wchodzą następujące komponenty:

- `NewPassword`,
- `FormContainer`,
- `FormInput`.

`NewPassword` (rys. 7.121 i 7.122) Komponent służy do podania nowego hasła po wcześniejszym zainicjowaniu procesu resetu. Za pomocą hooka `useSearchParams` odczytywany jest token przekazany w parametrze adresu [URL](#). Następnie, po zatwierdzeniu formularza, token wraz z nowym hasłem jest wysyłany na [backend](#) przy pomocy `useMutation`.

W formularzu znajdują się dwa pola `FormInput`: dla nowego hasła oraz jego potwierdzenia. Walidacja realizowana jest przez `useUserDataValidation`. Przycisk zatwierdzający pozostaje nieaktywny do momentu poprawnego uzupełnienia obu wartości. Po pomyślnym ustawieniu nowego hasła wyświetlany jest komunikat potwierdzający, a użytkownik przekierowywany jest do formularza logowania.

```
export default function NewPassword() { Show usages ⌘ stanoz +3
  const [searchParams] = useSearchParams();
  const token = searchParams.get("token");

  const {
    enteredValue,
    didEdit,
    isValid,
    handleInputChange,
    handleInputBlur,
  } = useUserDataValidation({ password: "", "confirm-password": "" });

  const { isSuccess, error, mutate } = useMutation({
    mutationFn: changePassword,
  });

  const handleSubmit = async (event) => { Show usages ⌘ KacperBadek +1
    event.preventDefault();

    mutate({ token, password: enteredValue.password });
  };
}
```

Rysunek 7.121: Komponent NewPassword (1).

```

<FormContainer
  header="Set new Password"
  isSuccess={isSuccess}
  error={error}
  navigateOnSuccess="/login"
  notificationMessage="New password set successfully!"
>
  <form onSubmit={handleSubmit}>
    <FormInput
      id="password"
      value={enteredValue.password}
      onChange={(event) => handleInputChange("password", event)}
      onBlur={() => handleInputBlur("password")}
      type="password"
      isValid={isValid?.password}
      label="new password"
    />
    <FormInput
      id="confirm-password"
      value={enteredValue["confirm-password"]}
      onChange={(event) =>
        handleInputChange("confirm-password", event)
      }
      onBlur={() => handleInputBlur("confirm-password")}
      type="password"
      isValid={isValid["confirm-password"]}
      label="confirm password"
    />
    <button
      type="submit"
      className="mx-1 my-2 w-full rounded-md bg-black p-1 text-white"
      disabled={
        !didEdit.password ||
        !didEdit["confirm-password"] ||
        !isValid.password.value ||
        !isValid["confirm-password"].value
      }
    >
      Set Password
    </button>
  </form>
</FormContainer>

```

386

Rysunek 7.122: Komponent NewPassword (2).

7.3.10.5 Komponenty wspólne panelu logowania

Panel logowania wykorzystuje zestaw komponentów uniwersalnych, wspólnych dla wszystkich opisanych formularzy. Zapewniają one spójny wygląd, jednolite mechanizmy walidacji oraz obsługę komunikatów.

FormContainer (rys. 7.123 i 7.124) Komponent odpowiada za główny układ formularzy uwierzytelniania. Wyświetla nagłówek sekcji, otacza właściwą zawartość odpowiednim tłem oraz rozmieszczeniem elementów, zgodnym z resztą aplikacji (z użyciem klas Tailwind CSS). Dodatkowo obsługuje logikę związaną z sukcesem i błędami operacji:

- w przypadku błędu pochodzącego z żądania HTTP (np. `AxiosError`) odpowiedni komunikat jest pobierany z odpowiedzi `backendu` i wyświetlany poprzez system powiadomień (`notificationAction.addError`);
- po pomyślnym zakończeniu operacji wyświetlany jest komunikat sukcesu (`notificationAction.addSuccess`), a opcjonalnie wykonywane jest przekierowanie na wskazaną ścieżkę.

Komponent umożliwia również wyświetlenie dodatkowych elementów, takich jak przyciski logowania zewnętrznego (`OauthForm`) oraz linki prowadzące do innych formularzy (np. przejście z logowania do rejestracji).

```
export default function FormContainer({ Show usages  ↗ Mredosz *
  isSuccess,
  error,
  header,
  navigateTo = "",
  linkCaption = "",
  showOauth = false,
  showLink = false,
  navigateOnSuccess = null,
  notificationMessage,
  children,
}: FormContainerProps) : Element {
  const navigate : NavigateFunction = useNavigate();
  const dispatch : ThunkDispatch<{ account: AccountSliceProp... }> = useDispatchTyped();

  useEffect(() : void => {...}, [isSuccess, navigate, navigateOnSuccess]);
  useEffect(() : void => {...}, [dispatch, error]);
  useEffect(() : void => {...}, [dispatch, isSuccess, notificationMessage]);
}
```

Rysunek 7.123: Komponent `FormContainer` (1).

```

return (
  <div className="■dark:bg-darkBg □bg-lightBg flex h-screen w-full items-center justify-center
    bg-cover bg-center bg-no-repeat">
    <div className="■dark:bg-darkBgSoft □bg-lightBgSoft mx-10 flex h-fit w-full flex-col
      justify-center rounded-md px-10 py-8 sm:mx-0 sm:w-[30rem]">
      <h1 className="■dark:text-darkText □text-lightText pb-8 text-center text-2xl
        font-bold text-shadow-md dark:text-shadow-white/20">
        {header}
      </h1>
      {children}
      {showOAuth && (
        <>
          <div>
            <div className="inline-flex w-full items-center justify-center">
              <hr className="■dark:bg-darkBorder □bg-lightBgMuted my-8 h-px w-96 border-0" />
              <span className="■dark:text-darkText □dark:bg-darkBgSoft □bg-lightBgSoft
                ■text-lightText absolute left-1/2 -translate-x-1/2 px-2 text-lg font-bold uppercase
                text-shadow-md dark:text-shadow-white/20">
                or
              </span>
            </div>
          </div>
          <OauthForm />
        </>
      )}
      {showLink && (
        <Link
          to={navigateTo}
          className="■dark:text-darkBorder ■text-lightText w-fit pt-8 text-sm text-shadow-md
          hover:underline dark:text-shadow-white/15"
        >
          {linkCaption}
        </Link>
      )}
    </div>
  </div>
);

```

Rysunek 7.124: Komponent `FormContainer` (2).

`FormInput` (rys. 7.125 i 7.126) Komponent realizuje pojedyncze pole wejściowe formularza z animowaną etykieta. Etykieta jest pozycjonowana i animowana za pomocą biblioteki `framer-motion`, dzięki czemu po aktywacji pola lub wpisaniu wartości „unosi się” ponad polem tekstowym, poprawiając czytelność formularza. Pole wejściowe korzysta ze wspólnego stylowania (ciemny/jasny motyw) oraz przekazanych funkcji obsługi zdarzeń `onChange` i `onBlur`, powiązanych z logiką walidacji.

Poniżej pola, w przypadku błędnej walidacji, wyświetlany jest komunikat o błęd-

dzie. Sekcja ta jest animowana przy użyciu `AnimatePresence` i `motion.p`, co zapewnia płynne pojawianie się i znikanie komunikatów walidacyjnych. Dzięki temu komponent `FormInput` jest używany we wszystkich formularzach panelu logowania, zapewniając spójne doświadczenie użytkownika.

```
interface InputProps { Show usages ▾ Mredosz
  label: string;
  id: string;
  isValid: { value: boolean; message: string };
  type: string;
  value: string;
  onChange: (e: ChangeEvent<HTMLInputElement>) => void;
  onBlur: () => void;
}

export default function FormInput({ Show usages ▾ Mredosz
  label,
  id,
  isValid,
  value,
  onBlur,
  type,
  onChange,
}: InputProps) : Element {
  const [isFocused, setFocusedToTrue, setFocusedToFalse] = useState(false);

  const shouldFloat :boolean = isFocused || Boolean(value);

  const handleOnBlur :() => void = () :void => { Show usages ▾ Mredosz
    setFocusedToFalse();
    onBlur();
  };
}
```

Rysunek 7.125: Komponent `FormInput` (1).

```

return (
  <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"
    />
    <AnimatePresence initial={false}>
      {!isValid?.value && isValid?.message && (
        <motion.p
          key="formInputError"
          initial={{ opacity: 0, y: -4 }}
          animate={{ opacity: 1, y: 0 }}
          exit={{ opacity: 0, y: -4 }}
          transition={{ duration: 0.2 }}
          className="text-xs font-bold break-words text-red-500"
        >
          {isValid.message}
        </motion.p>
      )}
    </AnimatePresence>
  </div>
);

```

Rysunek 7.126: Komponent `FormInput` (2).

`SubmitFormButton` (rys. 7.127) Komponent reprezentuje przycisk potwierdzający wysłanie formularza. Przyjmuje etykietę przycisku, listę pól formularza oraz informacje o tym, czy dane pola zostały edytowane i czy przeszły validację. Na tej podstawie wyznaczany jest stan `disabled`: przycisk pozostaje nieaktywny, dopóki wszystkie wymagane pola nie zostaną poprawnie uzupełnione.

Komponent wykorzystuje wspólne stylowanie (zaokrąglone rogi, cień, animacje `hover`) i jest stosowany w formularzach logowania oraz rejestracji. Dzięki temu logika aktywacji przycisku jest scentralizowana, a kod poszczególnych formularzy pozostaje prostszy i bardziej czytelny.

```
interface SubmitFormButtonProps { Show usages ▾ Mredosz
  label: string;
  fields: string[];
  didEdit: Record<string, boolean>;
  isValid: Record<string, { value: boolean }>;
}

export default function SubmitFormButton({ Show usages ▾ Mredosz *
  label,
  fields,
  didEdit,
  isValid,
}: SubmitFormButtonProps): Element {
  const isEnabled = !fields.every((f: string) : boolean => didEdit[f] && isValid[f].value);

  return (
    <button
      type="submit"
      className="■ dark:bg-violetDark ■ bg-violetLight/80 ■ not-disabled:hover:bg-violetLight
      ■ text-darkText ■ not-disabled:dark:hover:bg-violetDarker mt-3 w-full rounded-lg p-3
      font-semibold capitalize shadow-md shadow-violet-900/20 transition-all duration-300
      not-disabled:cursor-pointer dark:shadow-black/50"
      disabled={isEnabled}
    >
      {label}
    </button>
  );
}
```

Rysunek 7.127: Komponent `SubmitFormButton`.

7.3.11 Sidebar

`Sidebar` stanowi główny komponent nawigacyjny aplikacji. Implementacja obejmuje:

- zestaw komponentów interfejsu odpowiedzialnych za renderowanie pozycji, podmenu, tooltipów oraz obsługę interakcji użytkownika,

- pliki pomocnicze służące do budowania listy linków i rozpoznawania typów elementów nawigacji (`functions`, `sidebarLinks`),
- definicje typów i interfejsów opisujących strukturę pozycji paska bocznego (`link`),
- moduł Redux przechowujący stan rozwinięcia paska (`sidebar.ts`).

7.3.11.1 Komponenty

Sidebar

Komponent odpowiada za złożenie całej nawigacji oraz sterowanie jej zachowaniem (rys. 7.128). Wykorzystywane są `hooki` `useDarkMode` (obsługa motywów), `useSelectorTyped` (odczyt `isLoggedIn` oraz `isSidebarOpen`) oraz `useLocation` (sprawdzenie aktualnej ścieżki). Na podstawie lokalizacji ustalany jest tryb pozyjonowania: dla stron o układzie „sticky” (panel użytkownika, czat, strona główna) stosowana jest klasa `xl:sticky`, w pozostałych przypadkach `absolute`. Zestawy linków nawigacyjnych budowane są w pliku `sidebarLinks` (opis w podrozdz. 7.3.11.2).

W strukturze komponentu renderowane są:

- `SidebarToggleButton`,
- sekcja nawigacyjna: `SidebarSection` → `SidebarList` (linki główne),
- sekcja opcji: `SidebarSection` → `SidebarList` (akcje i opcje).

```

export default function Sidebar() : Element { Show usages ⌘Mredosz +2 *
  const [isDark, toggleDarkMode] = useDarkMode();
  const isLoggedIn : boolean = useSelectorTyped<State>({account: AccountSliceProps; notification: NotificationSliceProps; sidebar: SidebarSliceProps}) : boolean => state.account.isLoggedIn;
  const isSidebarOpen : boolean = useSelectorTyped<State>({account: AccountSliceProps; notification: NotificationSliceProps; sidebar: SidebarSliceProps}) : boolean => state.sidebar.isOpen;
  const location : Location<any> = useLocation();
  const isAccountPage : boolean = location.pathname.includes("/account");
  const isChatPage : boolean = location.pathname.includes("/chat");
  const isHomePage : boolean =
    location.pathname === "/" || location.pathname === "/advanced";
  const isStickyLayoutPage : boolean = isAccountPage || isChatPage || isHomePage;

  const allLinks : SidebarItemType[] = isLoggedIn
    ? [...staticLinks(isLoggedIn), ...userLoggedLinks]
    : [...staticLinks(isLoggedIn)];

  const optionsLinks : SidebarItemType[] = getOptionsLinks(isLoggedIn, isDark);

  return (
    <aside
      className={`bg-violetDark text-darkText fixed top-0 left-0 z-50 flex h-screen shrink-0 flex-col justify-between py-2 transition-all duration-300 ${isStickyLayoutPage ? "xl:sticky" : "absolute"}`}
      ${isSidebarOpen ? "w-full translate-x-0 xl:w-[220px]" : "w-[220px] -translate-x-full p-0 xl:w-[70px] " +
        "xl:translate-x-0`}
    >
      <div className="flex flex-col space-y-10">
        <SidebarToggleButton />
        <SidebarSection showBottomHr={true}>
          <SidebarList
            links={allLinks}
            isSidebarOpen={isSidebarOpen}
          />
        </SidebarSection>
      </div>
      <SidebarSection showTopHr={true}>
        <SidebarList
          links={optionsLinks}
          isSidebarOpen={isSidebarOpen}
          onChangeTheme={toggleDarkMode}
        />
      </SidebarSection>
    </aside>
  );
}

```

Rysunek 7.128: Fragment kodu komponentu `Sidebar`.

Tooltip

Komponent służy do prezentacji nazwy pozycji oraz (w przypadku submenu) listy pozycji podrzędnych, gdy pasek boczny pozostaje zwinięty. Po kliknięciu w element następuje przejście do przypisanej podstrony (rys. 7.129).

```

interface TooltipProps { Show usages & Mredosz
  links: SidebarSubmenuLink[];
  name: string;
}

export default function Tooltip({ links, name }: TooltipProps): Element { Show usages & Mredosz *
  return (
    <div className="bg-violetLight text-darkText absolute top-0 left-full z-50 ml-3.5 rounded-r-md pt-2 text-start text-base font-semibold whitespace nowrap capitalize">
      <p
        className={`${links?.length ? "cursor-auto" : "cursor-pointer"} px-3 pb-2`}>
        {name}
      </p>
      {links?.map((link: SidebarSubmenuLink) : Element => (
        <NavLink
          key={link.name}
          to={link.to}
          className="hover:bg-violetDark block px-3 py-0.5 font-normal text-gray-300 last:rounded-br-md"
        >
          {link.name}
        </NavLink>
      ))}
    </div>
  );
}

```

Rysunek 7.129: Fragment kodu komponentu `Tooltip`.

SidebarToggleButton

Komponent wyświetla przycisk zwijania/rozwijania paska bocznego (rys. 7.130). Po kliknięciu wywoływana jest akcja `toggleSidebar` z `Redux`. Na mniejszych ekranach (poniżej `xl` czyli 1280px) prezentowany jest również napis „Merkury”.

```
export default function SidebarToggleButton() : Element { Show usages ⚡ Mredosz *
  const dispatch : ThunkDispatch<{account: AccountSliceProp...} = useDispatchTyped();

  const handleToggle : () => { payload: undefined; type: "sideba... = () : { payload: undefined; type: "sidebar/togg... =>
    dispatch(sidebarAction.toggleSidebar());

  return (
    <div className="bg-violetDark mx-2 flex items-center justify-between">
      <button
        type="button"
        className="ml-2 w-fit cursor-pointer"
        onClick={handleToggle}
      >
        <IoMenu size={40} />
      </button>
      <span className="text-darkText mr-2 font-semibold xl:hidden">
        Merkury
      </span>
    </div>
  );
}
```

Rysunek 7.130: Fragment kodu komponentu SidebarToggleButton.

SidebarSection

Komponent grupuje listy nawigacyjne przekazane przez `children` oraz opcjonalnie dodaje separator u góry lub na dole (rys. 7.131).

```
interface SidebarSectionProps { Show usages & Mredosz
  children: ReactElement;
  showTopHr?: boolean;
  showBottomHr?: boolean;
}

const hrClasses = "border-violetLight mx-2 mt-1 mb-2";

export default function SidebarSection({ Show usages & Mredosz:
  children,
  showTopHr,
  showBottomHr,
}: SidebarSectionProps): Element {
  return (
    <nav className="flex flex-col space-y-1">
      {showTopHr && <hr className={hrClasses} />}
      {children}
      {showBottomHr && <hr className={hrClasses} />}
    </nav>
  );
}
```

Rysunek 7.131: Fragment kodu komponentu SidebarSection.

SidebarList

Komponent renderuje listę elementów nawigacji poprzez mapowanie na SidebarItem (rys. 7.132).

```
interface SidebarListProps { Show usages  ↳ Mredosz
  links: BaseLink[];
  isSidebarOpen: boolean;
  onChangeTheme?: () => void;
}

export default function SidebarList(props: SidebarListProps) : Element[] { Show usages  ↳ Mredosz
  return props.links.map((link : BaseLink , index : number ) : Element  => (
    |   <SidebarItem key={link.name} link={link} {...props} index={index} />
  )));
}
```

Rysunek 7.132: Fragment kodu komponentu SidebarList.

SidebarLabel

Komponent odpowiada za wyświetlenie etykiety pozycji, gdy pasek jest rozwinięty. Dla płynnej animacji zastosowano `AnimatePresence` oraz `motion.p`. W przypadku submenu z niepustą listą dzieci prezentowana jest dodatkowo ikona strzałki (rys. 7.133).

```

interface SidebarLabelProps { Show usages ↵ Mredosz
  link: BaseLink;
  isOpen?: boolean;
  isSidebarOpen: boolean;
}

export default function SidebarLabel({ Show usages ↵ Mredosz
  link,
  isOpen,
  isSidebarOpen,
}: SidebarLabelProps): Element {
  return (
    <AnimatePresence initial={false}>
      {isSidebarOpen && (
        <div className="relative overflow-hidden">
          <motion.p
            key="sidebar-label"
            initial={{ opacity: 0, x: -8 }}
            animate={{ opacity: 1, x: 0 }}
            exit={{ opacity: 0, x: -8 }}
            transition={{ duration: 0.2 }}
            className={`ml-2 flex min-w-[10rem] items-center text-start font-semibold capitalize`}
          >
            {link.name}
            {isSidebarSubmenu(link) &&
              link.children?.length > 0 &&
              (isOpen ? (
                <IoIosArrowUp className="ml-2" />
              ) : (
                <IoIosArrowDown className="ml-2" />
              ))}
          </motion.p>
        </div>
      )}
    </AnimatePresence>
  );
}

```

Rysunek 7.133: Fragment kodu komponentu `SidebarLabel`.

SidebarIcon

Komponent odpowiada za prezentację ikony danej pozycji. W przypadku zwiniętego paska i pozycji typu submenu ikona stanowi `NavLink` (umożliwia przejście do strony nadrzędnej), natomiast dla `link` i `action` zwracana jest sama ikona (rys. 7.134).

```
interface SidebarIconProps { Show usages  ↗ Mredosz
  link: BaseLink;
  isSidebarOpen: boolean;
}

export default function SidebarIcon({ link, isSidebarOpen }: SidebarIconProps): Element | null {
  if (isSidebarSubmenu(link) && !isSidebarOpen) {
    return (
      <NavLink
        to={link.to}
        className="flex h-10 w-10 items-center justify-center"
      >
        <span className="z-50 flex h-full w-full items-center justify-center">
          {link.icon}
        </span>
      </NavLink>
    );
  }

  if (isSidebarLink(link) || isSidebarAction(link)) {
    return link.icon;
  }
  return null;
}
```

Rysunek 7.134: Fragment kodu komponentu `SidebarIcon`.

SidebarItemSubmenuLink

Komponent renderuje elementy podzielone submenu jako `NavLink`. Przy każdym kliknięciu wywoływana jest akcja `closeSidebar`, która zwija pasek wyłącznie na ekranach o szerokości mniejszej niż 1280px (rys. 7.135).

```

interface SidebarItemSubmenuProps { Show usages ▾ Mredosz
  link: SidebarSubmenuLink;
}

export default function SidebarItemSubmenuLink({ Show usages ▾ Mredosz *
  link,
}: SidebarItemSubmenuProps) : Element {
  const dispatch : ThunkDispatch<{account: AccountSliceProp...} = useDispatchTyped();

  const handleClose : () => void = () : void => dispatch(closeSidebar());
  Show usages ▾ Mredosz

  return (
    <NavLink
      to={link.to}
      end
      onClick={handleClose}
      className={({ isActive } : NavLinkRenderProps) : string =>
        ` hover:bg-violetLight mx-2 flex items-center space-x-1 rounded-md
          text-gray-300 ${isActive ? " bg-violetLight" : ""}`
      }
    >
      <p className="ml-12 min-w-[10rem] py-1.5 text-start font-semibold capitalize">
        {link.name}
      </p>
    </NavLink>
  );
}

```

Rysunek 7.135: Fragment kodu komponentu SidebarItemSubmenuLink.

SidebarItemSubmenu

Komponent odpowiada za obsługę pozycji typu submenu (rys. 7.136–7.138). Zastosowano:

- `useToggleState` do przełączania stanu rozwinięcia,
- `useState` do przechowywania nazwy aktualnie otwartego submenu (w celu utrzymania zasady „jedno submenu otwarcie naraz”),
- `useLocation` oraz `useEffect` do automatycznego rozwinięcia submenu, gdy aktywna jest jedna z podstron potomnych.

Pozycja submenu renderowana jest jako przycisk zawierający SidebarItemContent, a lista elementów potomnych pojawia się animacyjnie (przez AnimatePresence i motion.div) tylko wtedy, gdy pasek jest rozwinięty.

```
interface SidebarItemSubmenuProps { Show usages ▾ Mredosz
  link: SidebarSubmenu;
  isSidebarOpen: boolean;
  index: number;
  showTooltip: () => void;
  hideTooltip: () => void;
  isTooltipShown: boolean;
}

export default function SidebarItemSubmenu({ Show usages ▾ Mredosz +1
  index,
  isSidebarOpen,
  link,
  showTooltip,
  hideTooltip,
  isTooltipShown,
}: SidebarItemSubmenuProps): Element {
  const [is_submenuOpen, setIs_submenuOpen, toggle_submenu] =
    useState(false);
  const [opened_submenuName, setOpened_submenuName] = useState<string | null>();
  const [isActive, setIsActive] = useState(false);
  const location: Location<any> = useLocation();

  useEffect(() : void => {
    if (opened_submenuName !== link.name && is_submenuOpen) {
      toggle_submenu();
    }
  }, [opened_submenuName]);
```

Rysunek 7.136: Fragment kodu komponentu SidebarItemSubmenu (1).

```
useEffect(() : void => {
  const hasActiveChild : boolean =
    link.children?.some(child : SidebarSubmenuLink) : boolean =>
    location.pathname.startsWith(child.to),
  ) ?? false;

  setIsActive(hasActiveChild);

  if (hasActiveChild) {
    setOpenedSubMenuName(link.name);
    setIsSubMenuOpen(true);
  } else if (openedSubMenuName === link.name) {
    setOpenedSubMenuName(null);
    setIsSubMenuOpen(false);
  }
}, [location.pathname]);

const handleOpenSubMenu : () => void = () : void => { Show usages ⚡ Mredosz
  setOpenedSubMenuName(isSubMenuOpen ? null : link.name);
  toggleSubMenu();
};
```

Rysunek 7.137: Fragment kodu komponentu SidebarItemSubmenu (2).

```

return (
  <div
    className={`mx-2 ${index === 0 && "mt-2"} ${!isSidebarOpen && "hover:mr-0"}`}
  >
  <button
    type="button"
    onClick={handleOpenSubmenu}
    onMouseEnter={showTooltip}
    onMouseLeave={hideTooltip}
    className={`${!isSidebarOpen ? "hover:rounded-r-none" : ""} ${isActive ? "bg-violetLight" : ""}`}
  >
    <SidebarItemContent
      isSidebarOpen={isSidebarOpen}
      link={link}
      isOpen={is_submenuOpen}
      isTooltipShown={isTooltipShown}
    />
  </button>
  <AnimatePresence initial={false}>
    {is_submenuOpen && isSidebarOpen && (
      <motion.div
        key="submenu"
        initial={{ opacity: 0, height: 0, x: -8 }}
        animate={{ opacity: 1, height: "auto", x: 0 }}
        exit={{ opacity: 0, height: 0, x: -8 }}
        transition={{ duration: 0.3 }}
        className="mt-2 space-y-1 overflow-hidden"
      >
        {link.children?.map((link : SidebarSubmenuLink) : Element => (
          <SidebarItemSubmenuLink
            key={link.name}
            link={link}
          />
        ))}
      </motion.div>
    )}
  </AnimatePresence>
</div>
);
}

```

Rysunek 7.138: Fragment kodu komponentu `SidebarItemSubmenu` (3).

SidebarItemLink

Komponent renderuje pojedynczą pozycję typu `link` jako `NavLink` zawierający `SidebarItemContent`. Dodatkowo, po kliknięciu wywoływana jest akcja `closeSidebar` (działająca tylko dla szerokości poniżej 1280px) (rys. 7.139–7.140).

```
interface SidebarItemLinkProps { Show usages ▾ Mredosz
  link: SidebarLink;
  isSidebarOpen: boolean;
  showTooltip: () => void;
  hideTooltip: () => void;
  isTooltipShown: boolean;
}

export default function SidebarItemLink({ Show usages ▾ Mredosz *
  link,
  showTooltip,
  hideTooltip,
  isSidebarOpen,
  isTooltipShown,
}: SidebarItemLinkProps) : Element {
  const dispatch : ThunkDispatch<{ account: AccountSliceProp... } = useDispatchTyped();

  const handleClose :()=> void = () :void => dispatch(closeSidebar());
  Show usages ▾ Mredosz

  const isHome :boolean = link.to === "/";
  const isHomeActive :boolean =
    isHome &&
    (location.pathname === "/" ||
     location.pathname.startsWith("/advanced"));
}
```

Rysunek 7.139: Fragment kodu komponentu SidebarItemLink (1).

```
return (
  <NavLink
    to={link.to}
    id={`sidebar-link-${link.name}`}
    end
    onClick={handleClose}
    onMouseEnter={showTooltip}
    onMouseLeave={hideTooltip}
    className={({ isActive }: NavLinkRenderProps) : string =>
      ` hover:bg-violetLight mx-2 flex items-center space-x-3 rounded-md pl-2
      transition-all ${!isSidebarOpen} ? "transition-none" : ""
      ${!isSidebarOpen && isTooltipShown} ? "hover:mr-0 hover:rounded-r-none" : ""
      ${isActive || isHomeActive} ? " bg-violetLight" : ""
    }
  >
  <SidebarItemContent
    isSidebarOpen={isSidebarOpen}
    link={link}
    isTooltipShown={isTooltipShown}
  />
</NavLink>
);
}
```

Rysunek 7.140: Fragment kodu komponentu `SidebarItemLink` (2).

`SidebarItemContent`

Komponent stanowi wspólne „wnętrze” elementów `SidebarItem`. Zawiera `SidebarIcon`, warunkowo `Tooltip` (gdy pasek jest zwinięty i aktywny jest stan tooltipa) oraz `SidebarLabel` (rys. 7.141).

```

interface SidebarItemContentProps { Show usages ▾ Mredosz
  link: BaseLink;
  isSidebarOpen: boolean;
  isTooltipShown: boolean;
  isOpen?: boolean;
}

export default function SidebarItemContent({ Show usages ▾ Mredosz +1
  link,
  isSidebarOpen,
  isOpen,
  isTooltipShown,
}: SidebarItemContentProps): Element {
  return (
    <div className="flex items-center transition-all">
      <div className="relative flex h-10 w-10 shrink-0 items-center justify-center text-3xl">
        <SidebarIcon link={link} isSidebarOpen={isSidebarOpen} />
        {!isSidebarOpen && isTooltipShown && (
          <Tooltip
            name={link.name}
            links={isSidebarSubmenu(link) ? link.children : []}
          />
        )}
      </div>
      <SidebarLabel
        link={link}
        isOpen={isOpen}
        isSidebarOpen={isSidebarOpen}
      />
    </div>
  );
}

```

Rysunek 7.141: Fragment kodu komponentu SidebarItemContent.

SidebarItemAction

Komponent obsługuje pozycje typu `action`, które nie prowadzą do `routingu`, lecz uruchamiają funkcje. Zaimplementowano akcję wylogowania (wywołanie `logout` oraz aktualizacja stanu konta) oraz przełączanie motywów poprzez `onChangeTheme`. Renderowany jest przycisk zawierający `SidebarItemContent` (rys. 7.142–7.143).

```

interface SidebarItemActionProps { Show usages ▾ Mredosz
  link: SidebarAction;
  isSidebarOpen: boolean;
  showTooltip: () => void;
  hideTooltip: () => void;
  onChangeTheme: () => void;
  isTooltipShown: boolean;
}

export default function SidebarItemAction({ Show usages ▾ Mredosz
  link,
  showTooltip,
  hideTooltip,
  onChangeTheme,
  isSidebarOpen,
  isTooltipShown,
}: SidebarItemActionProps) : Element {
  const dispatch : ThunkDispatch<{account: AccountSliceProp...} = useDispatchTyped();
  const navigate : NavigateFunction = useNavigate();

  const handleSignOut :()=> Promise<void> = async () :Promise<void> => { Show usages ▾ Mredosz
    try {
      await logout();
      dispatch(
        notificationAction.addSuccess({
          message: "You have been successfully logged out",
        }),
      );
      dispatch(accountAction.signOut());
    } catch (error: any) {
      dispatch(
        notificationAction.addError({
          message: error?.message || "Logout failed!",
        }),
      );
    }
    navigate("/");
  };
}

```

Rysunek 7.142: Fragment kodu komponentu `SidebarItemAction` (1).

```

let clickHandler: (() => void) | undefined;
switch (link.actionType) {
  case "login":
    clickHandler = handleSignOut;
    break;
  case "changeMode":
    clickHandler = onChangeTheme;
    break;
  default:
    clickHandler = () : void => {};
}

return (
  <button
    type="button"
    onClick={clickHandler}
    onMouseEnter={showTooltip}
    onMouseLeave={hideTooltip}
    className={`&#x2022; hover:bg-violetLight mx-2 flex cursor-pointer
    items-center space-x-3 rounded-md pl-2
    ${!isSidebarOpen ? "mr-0 rounded-r-none transition-none" : ""}`}
  >
  <SidebarItemContent
    isSidebarOpen={isSidebarOpen}
    link={link}
    isTooltipShown={isTooltipShown}
  />
</button>
);
}

```

Rysunek 7.143: Fragment kodu komponentu `SidebarItemAction` (2).

`SidebarItem`

Komponent wybiera właściwy podtyp elementu na podstawie pola `type` (rys. 7.144–7.145). Wykorzystywany jest `hook useBoolean` do sterowania widocznością tooltipa oraz `useLocation` i `useEffect` do ukrycia tooltipa po zmianie ścieżki. Do-

bór implementacji realizowany jest przez instrukcję `switch: submenu → SidebarItemSubmenu`, `action → SidebarItemAction`, `link → SidebarItemLink`.

```
interface SidebarItemProps { Show usages  ↳ Mredosz
  link: BaseLink;
  isSidebarOpen: boolean;
  onChangeTheme?: () => void;
  index: number;
}

export default function SidebarItem({ Show usages  ↳ Mredosz
  link,
  isSidebarOpen,
  onChangeTheme,
  index,
}: SidebarItemProps): Element {
  const [isTooltipShown, showTooltip, hideTooltip] = useBoolean(false);
  const location: Location<any> = useLocation();

  useEffect(() : void => {
    hideTooltip();
  }, [location.pathname]);
```

Rysunek 7.144: Fragment kodu komponentu `SidebarItem` (1).

```
switch (link.type) {
  case "submenu":
    return (
      <SidebarItemSubmenu
        isSidebarOpen={isSidebarOpen}
        link={link as SidebarSubmenu}
        index={index}
        showTooltip={showTooltip}
        hideTooltip={hideTooltip}
        isTooltipShown={isTooltipShown}
      />
    );
  case "action":
    return (
      <SidebarItemAction
        link={link as SidebarAction}
        isSidebarOpen={isSidebarOpen}
        onChangeTheme={onChangeTheme!}
        showTooltip={showTooltip}
        hideTooltip={hideTooltip}
        isTooltipShown={isTooltipShown}
      />
    );
  case "link":
  default:
    return (
      <SidebarItemLink
        link={link as SidebarLink}
        isSidebarOpen={isSidebarOpen}
        showTooltip={showTooltip}
        hideTooltip={hideTooltip}
        isTooltipShown={isTooltipShown}
      />
    );
}
```

Rysunek 7.145: Fragment kodu komponentu `SidebarItem` (2).

7.3.11.2 Pliki pomocnicze

functions

Plik zawiera funkcje typu *type guard* umożliwiające rozróżnienie wariantów linków: `isSidebarSubmenu`, `isSidebarAction` oraz `isSidebarLink` (rys. 7.146).

```
export function isSidebarSubmenu(link: BaseLink): link is SidebarSubmenu { Show usages ⚙ Mredosz
  return (
    "children" in link &&
    Array.isArray(link.children) &&
    "to" in link &&
    typeof link.to === "string"
  );
}

export function isSidebarAction(link: BaseLink): link is SidebarAction { Show usages ⚙ Mredosz
  return "actionType" in link && typeof link.actionType === "string";
}

export function isSidebarLink(link: BaseLink): link is SidebarLink { Show usages ⚙ Mredosz
  return "to" in link && typeof link.to === "string";
}
```

Rysunek 7.146: Fragment kodu pliku `functions` (funkcje typu *guard*).

sidebarLinks

Plik definiuje strukturę nawigacji w postaci obiektów: `staticLinks`, `userLoggedLinks` oraz `getOptionsLinks` (rys. 7.147–7.149). Zależnie od stanu zalogowania (`isLoggedIn`) generowane są różne zestawy pozycji:

- dla użytkownika niezalogowanego: strona główna, mapa oraz forum (strona główna forum, regulamin),
- dla użytkownika zalogowanego: dodatkowo lista obserwowanych postów na forum, czat oraz panel użytkownika
- (submenu „account” z podstronami profilu, ustawień itd.).

Sekcja opcji (`getOptionsLinks`) zawiera:

- pozycję logowania lub wylogowania (zależnie od `isLoggedIn`),

- przełączenie motywu jasny/ciemny (zależnie od `isDark`).

```

export const staticLinks : (isLoggedIn: boolean) => SidebarItemType[] = (isLoggedIn: boolean): SidebarItemType[] => [
  {
    to: "/",
    icon: <BiHome aria-label="home" />,
    name: "home",
    type: "link",
  },
  {
    to: "/map",
    icon: <FaRegMap aria-label="map" />,
    name: "map",
    type: "link",
  },
  {
    to: "/forum",
    icon: <MdOutlineForum aria-label="forum" />,
    name: "forum",
    type: "submenu",
    children: [
      {
        to: "/forum",
        name: "home page",
        type: "link",
      },
      {
        to: "/forum/guidelines",
        name: "guidelines",
        type: "link",
      },
      ...(isLoggedIn
        ? [
          {
            to: "/forum/followed",
            name: "followed posts",
            type: "link",
          } as const,
        ]
        : []),
    ],
  },
];

```

Rysunek 7.147: Fragment kodu pliku `sidebarLinks` (linki statyczne).

```
export const userLoggedLinks: SidebarItemType[] = [ Show usages & Mredosz +1
{
  to: "/chat",
  icon: <BiMessageRounded aria-label="chat" />,
  name: "chat",
  type: "link",
},
{
  icon: <FaRegUser aria-label="account" />,
  name: "account",
  type: "submenu",
  to: "/account/profile",
  children: [
    {
      to: "/account/profile",
      name: "profile",
      type: "link",
    },
    {name: "spots" ...},
    {name: "photos" ...},
    {name: "movies" ...},
    {name: "social" ...},
    {name: "add spot" ...},
    {name: "comments" ...},
    {name: "settings" ...},
  ],
},
];

```

Rysunek 7.148: Fragment kodu pliku sidebarLinks (linki dostępne dla użytkownika zalogowanego).

```
export const getOptionsLinks : (isLoggedIn: boolean, isDark: boolean) => SidebarItemType[] = ( isLoggedIn: boolean,
  isDark: boolean,
): SidebarItemType[] => {
  return [
    isLoggedIn
      ? {
          icon: <TbLogout2 aria-label="login" />,
          name: "sign out",
          actionType: "login",
          type: "action",
        }
      : {
          icon: <TbLogin2 aria-label="login" />,
          to: "/login",
          name: "login",
          type: "link",
        },
    isDark
      ? {
          icon: <LuSun aria-label="changeMode" />,
          name: "light mode",
          actionType: "changeMode",
          type: "action",
        }
      : {
          icon: <LuMoon aria-label="changeMode" />,
          name: "dark mode",
          actionType: "changeMode",
          type: "action",
        },
  ];
};
```

Rysunek 7.149: Fragment kodu pliku sidebarLinks (linki opcji: logowanie/wylogowanie oraz zmiana motywów).

link

Plik z typami definiuje LinkType oraz interfejsy: BaseLink, SidebarLink, SidebarSubmenuLink, SidebarSubmenu, SidebarAction (rys. 7.150).

```
type LinkType = "link" | "submenu" | "action";

export interface BaseLink { 4 inheritors Show usages ▾ Mredosz
  name: string;
  type: LinkType;
}

export interface SidebarLink extends BaseLink { Show usages ▾ Mredosz
  type: "link";
  to: string;
  icon: ReactElement;
}

export interface SidebarSubmenuLink extends BaseLink { Show usages ▾ Mredosz
  type: "link";
  to: string;
}

export interface SidebarSubmenu extends BaseLink { Show usages ▾ Mredosz
  type: "submenu";
  to: string;
  icon: ReactElement;
  children: SidebarSubmenuLink[];
}

export interface SidebarAction extends BaseLink { Show usages ▾ Mredosz
  type: "action";
  actionType: string;
  icon: ReactElement;
}

export type SidebarItemType = SidebarLink | SidebarSubmenu | SidebarAction;
```

Rysunek 7.150: Fragment kodu pliku link.

7.3.11.3 Redux

Stan paska bocznego utrzymywany jest w module `Redux` w pliku `sidebar.ts`. Przechowywana jest wyłącznie flaga `isOpen` informująca o stanie rozwinięcia. Zdefiniowano:

- reducer `setIsSidebarOpen` (jawne ustawienie wartości),
- reducer `toggleSidebar` (przełączenie stanu),
- thunk `closeSidebar`, który zwija pasek wyłącznie dla szerokości mniejszej niż 1280px.

Rozwiązanie to umożliwia automatyczne zwijanie paska po kliknięciu w link na urządzeniach mobilnych, bez wpływu na zachowanie na dużych ekranach (rys. 7.151).

```
type sidebarInitialState = {
  isOpen: boolean;
};

const initialState: sidebarInitialState = {
  isOpen: false,
};

/** Thunk to close the sidebar on smaller screens. ...*/
export const closeSidebar : () => (dispatch: ThunkDispatch<{account:...}> = () : (dispatch: ThunkDispatch<{account: Account}> => (dispatch: AppDispatch) : void) => {
  if (window.innerWidth < 1280) {
    dispatch(sideBarAction.setIsSidebarOpen(false));
  }
};

export const sidebarSlice : Slice<sidebarInitialState, {setIsSidebarOpen: (...args: any[]) => void, toggleSidebar: (...args: any[]) => void}> = createSlice({
  name: "sidebar",
  initialState,
  reducers: {
    /** Sets the sidebar open/closed explicitly. ...*/
    setIsSidebarOpen(state : WritableDraft<sidebarInitialState>, action : {payload: any; type: string}) : void {
      state.isOpen = action.payload;
    },
    /** Toggles the current open/close state of the sidebar. ...*/
    toggleSidebar(state : WritableDraft<sidebarInitialState>) : void {
      state.isOpen = !state.isOpen;
    },
  },
});

export const sidebarAction : CaseReducerActions<{setIsSidebarOpen: (...args: any[]) => void, toggleSidebar: (...args: any[]) => void}> = sidebarSlice.actions;
```

Rysunek 7.151: Slice `Redux` odpowiadający za stan paska bocznego.

7.3.11.4 Użycie w układzie aplikacji

Layout

Stanowi szkielet widoków aplikacji: renderuje `Sidebar` oraz część główną (`main`) zawierającą `MobileBar`, listę powiadomień oraz `Outlet` (miejsce wstrzyknięcia aktualnej podstrony przez router). Dla strony mapy ustawiany jest układ `relative`, a dla pozostałych widoków układ elastyczny (`flex`). Dodatkowo, w `useEffect` wykrywana jest nawigacja do ścieżek panelu konta; na ekranach o szerokości co najmniej 1280px pasek boczny jest wówczas automatycznie rozwijany poprzez `dispatch(sidebarAction.setIsSidebarOpen(true))` (rys. 7.152).

```
export default function Layout() { Show usages ▾ Adam Langmesser +2
  const location = useLocation();
  const isMapPage = location.pathname === "/map";
  const dispatch = useDispatchTyped();

  useEffect(() => {
    if (
      location.pathname.includes("account") &&
      window.innerWidth >= 1280
    ) {
      dispatch(sidebarAction.setIsSidebarOpen(true));
    }
  }, [location]);

  return (
    <div className={`${isMapPage ? "relative" : "flex"} min-h-screen`}>
      <Sidebar />
      <main className="relative flex w-full flex-col items-center justify-center">
        <MobileBar />
        <NotificationList title="test" message="message" />
        <Outlet />
      </main>
    </div>
  );
}
```

Rysunek 7.152: Fragment kodu komponentu `Layout`.

`MobileBar`

Wyświetlany jest wyłącznie poniżej progu xl (1280px) (klasa `xl:hidden`). Stanowi górny pasek nawigacyjny na urządzeniach mobilnych: zawiera przycisk z ikoną menu, który przełącza stan paska bocznego (`toggleSidebar`), oraz tytuł aplikacji „Merkury”. Dzięki umieszczeniu w części głównej (`main`) komponent pozostaje wiadoczny nad zawartością strony, jednocześnie nie wpływając na układ desktopowy (rys. 7.153).

```
export default function MobileBar(): Element { Show usages ↗ Mredosz +1 *  
  const dispatch: ThunkDispatch<{ account: AccountSliceProp... }> = useDispatchTyped();  
  
  const handleToggle: () => { payload: undefined; type: "sidebar/togg... " } = () =>  
    dispatch(sideBarAction.toggleSidebar());  
  
  return (  
    <div className="bg-violetDark text-darkText absolute top-0 left-0 z-20 flex w-full  
      items-center justify-between p-2 xl:hidden">  
      <button  
        onClick={handleToggle}  
        className="ml-2 w-fit cursor-pointer"  
      >  
        <IoMenu size={40} />  
      </button>  
      <span className="mr-2 font-semibold">Merkury</span>  
    </div>  
  );  
}
```

Rysunek 7.153: Fragment kodu komponentu `MobileBar`.

7.4 Implementacja CI/CD

W projekcie zastosowano mechanizmy [CI/CD](#) z wykorzystaniem [GitHub Actions](#), ponieważ narzędzie to jest zintegrowane z repozytorium i umożliwia automatyczne uruchamianie procesu budowania oraz testowania po każdej zmianie w kodzie. W ramach repozytorium przygotowano dwa niezależne [workflow](#): dla [backendu](#) (Java CI) oraz dla [frontendu](#) (React CI). Rozdzielenie [pipeline'ów](#) pozwoliło ograniczyć liczbę uruchomień tylko do przypadków, gdy modyfikacje dotyczą danej części systemu.

7.4.1 Pipeline backendu

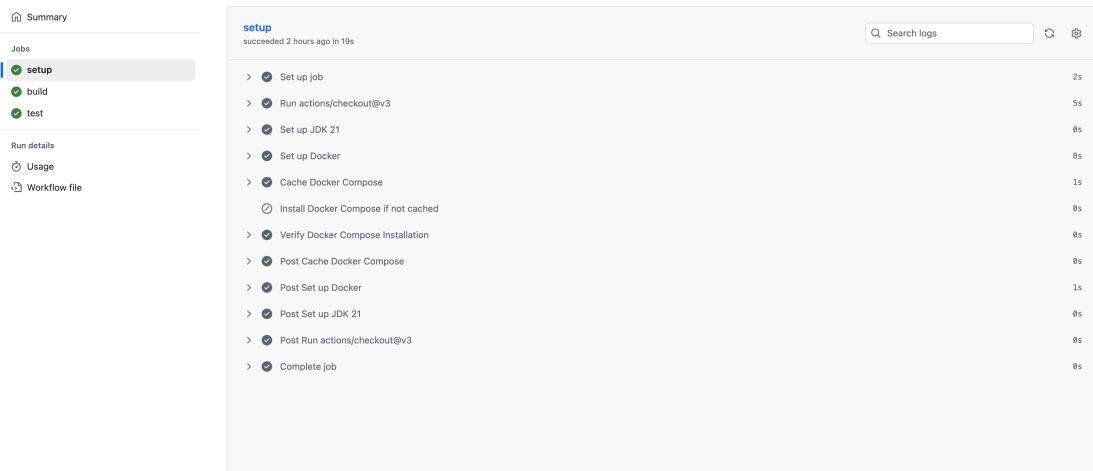
Workflow backendu zdefiniowano w pliku `java-ci.yml`. Jest on uruchamiany:

- przy zdarzeniu `push` dla zmian w katalogu `vulcanus/**`,
- przy otwarciu `pull request` do gałęzi `master` lub `develop`,
- w trybie `merge_group` (kolejka scalania) dla gałęzi `master` lub `develop`,

z jednoczesnym pomijaniem zmian w katalogu `book/**`. Fragment konfiguracji wyzwalaczy przedstawiono (rys. 7.154).

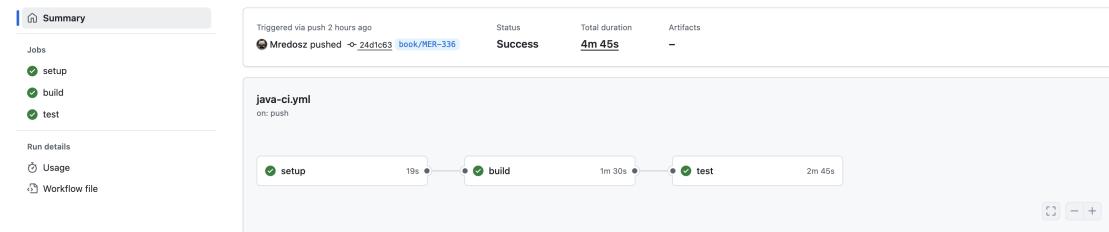
Wyzwalacze `workflow` (on:).

W plikach konfiguracyjnych [GitHub Actions](#) zastosowano wyzwalacze (*triggery*), które określają moment uruchomienia `workflow`. W przypadku zdarzenia `push` pipeline uruchamia się po wypchnięciu commitów do repozytorium, przy czym użyto filtra `paths`, aby wykonywać `workflow` wyłącznie dla zmian w odpowiednich katalogach (`vulcanus/**` lub `venus/**`). Zdarzenie `pull_request` powoduje uruchomienie `workflow` przy otwarciu `pull requesta` do gałęzi `master` lub `develop`, co umożliwia weryfikację jakości zmian przed ich scaleniem. Dodatkowo wykorzystano `paths-ignore`, aby pomijać uruchomienia związane wyłącznie ze zmianami w katalogu `book/**`. Zdarzenie `merge_group` odpowiada za uruchamianie `workflow` w kontekście kolejki scalania (*merge queue*) dla gałęzi `master` oraz `develop`, co pozwala testować zmiany w warunkach zbliżonych do rzeczywistego scalenia.



Rysunek 7.154: Fragment pliku `java-ci.yml`: konfiguracja wyzwalacych workflow (`on:`).

Workflow składa się z trzech jobów uruchamianych sekwencyjnie: `setup`, `build` oraz `test`. Zależności pomiędzy **jobami** (kolejność wykonania) oraz przykładowe czasy trwania przedstawia (rys. 7.155).



Rysunek 7.155: Podsumowanie wykonania workflow backendu (`java-ci.yml`) w GitHub Actions.

7.4.1.1 Job setup

Job setup odpowiada za przygotowanie środowiska uruchomieniowego na runnerze: pobiera repozytorium, ustawia wersję JDK 21, inicjalizuje środowisko **Docker** oraz obsługuje **cache** narzędzia **Docker Compose** (instalacja wykonywana jest wyłącznie w przypadku braku trafienia w `cache`). Konfigurację tego joba przedstawia

(rys. 7.156), a przykładowy log wykonania (rys. 7.157).

```
jobs:
  setup:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK 21
        uses: actions/setup-java@v3
        with:
          java-version: '21'
          distribution: 'temurin'

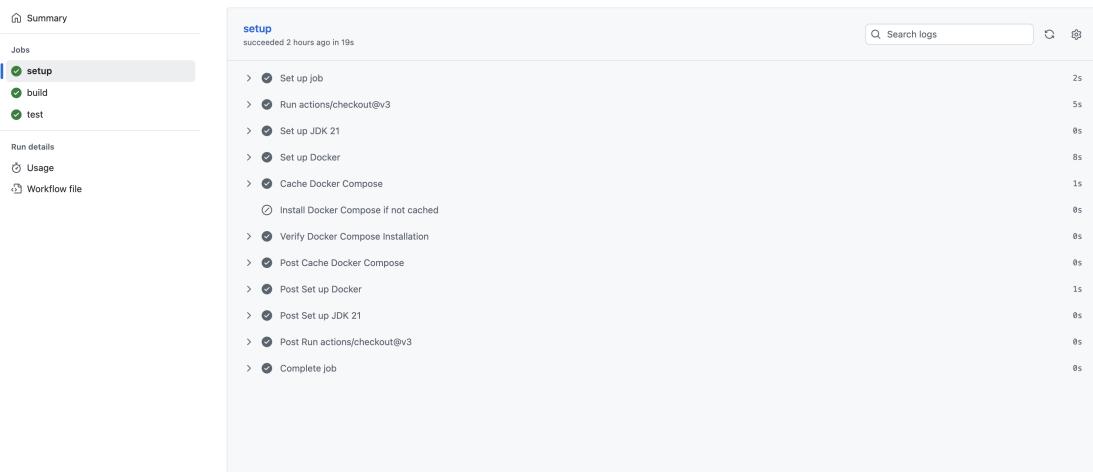
      - name: Set up Docker
        uses: docker/setup-buildx-action@v2

      - name: Cache Docker Compose
        id: docker-compose-cache
        uses: actions/cache@v3
        with:
          path: /usr/local/bin/docker-compose
          key: docker-compose-v2.20.2-${{ runner.os }}-${{ runner.arch }}
          restore-keys: |
            docker-compose

      - name: Install Docker Compose if not cached
        if: steps.docker-compose-cache.outputs.cache-hit != 'true'
        run: |
          sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
          sudo chmod +x /usr/local/bin/docker-compose

      - name: Verify Docker Compose Installation
        run: docker-compose --version
```

Rysunek 7.156: Fragment pliku `java-ci.yml`: definicja joba `setup`.



Rysunek 7.157: Przykładowy log wykonania joba `setup` w GitHub Actions.

7.4.1.2 Job build

Job build jest uruchamiany po poprawnym zakończeniu setup. W jego ramach:

- ustawiana jest wersja JDK 21,
- przywracany jest cache repozytorium Mavena (przyspieszenie pobierania zależności),
- uruchamiane są wymagane usługi poprzez docker compose,
- wykonywany jest build aplikacji z pominięciem testów (-DskipTests).

Konfigurację joba build przedstawia (rys. 7.158), natomiast przykładowy przebieg wykonania w GitHub Actions (rys. 7.159).

```

build:
  runs-on: ubuntu-latest
  needs: setup

steps:
  - uses: actions/checkout@v3

  - name: Set up JDK 21
    uses: actions/setup-java@v3
    with:
      java-version: '21'
      distribution: 'temurin'

  - name: Cache Maven repository
    uses: actions/cache@v4
    with:
      path: |
        ~/.m2/repository
        ~/.m2/wrapper
      key: maven-${{ runner.os }}-${{ hashFiles('vulcanus/pom.xml') }}
      restore-keys: |
        maven-${{ runner.os }}-

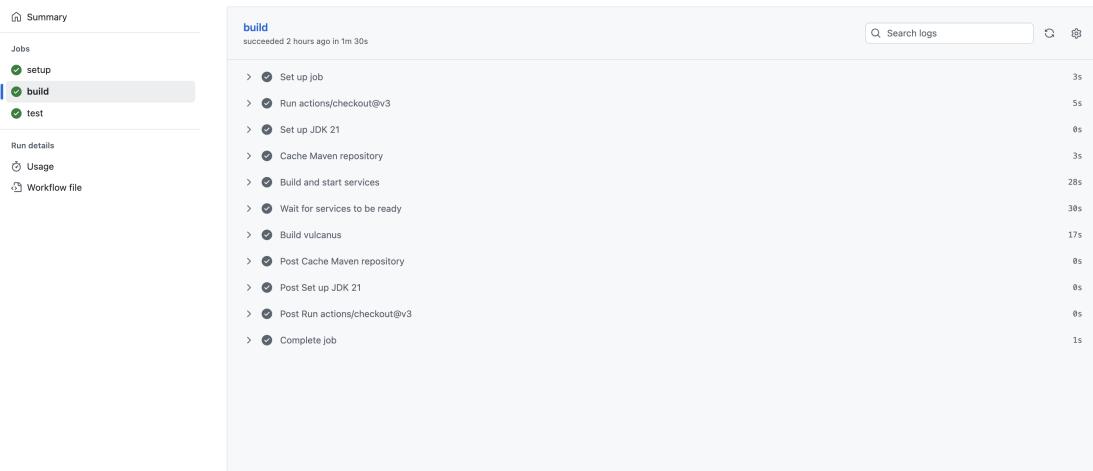
  - name: Build and start services
    run: docker compose -f vulcanus/docker/merkury/docker-compose.yml up -d

  - name: Wait for services to be ready
    run: sleep 30

  - name: Build vulcanus
    run: mvn -B package -DskipTests --file vulcanus/pom.xml

```

Rysunek 7.158: Fragment pliku `java-ci.yml`: definicja joba `build`.



Rysunek 7.159: Przykładowy log wykonania joba `build` w GitHub Actions.

7.4.1.3 Job test

Job `test` jest uruchamiany po zakończeniu `build`. W jego ramach przywracany jest [cache Mavena](#), a następnie uruchamiane są testy [backendu](#). W [jobie](#) wykorzystywane są również zmienne środowiskowe przekazywane z `secrets` repozytorium (dane dostępowe do usług zewnętrznych), co pozwala na bezpieczne wykonywanie testów bez umieszczania wrażliwych danych w kodzie. Konfigurację joba przedstawia (rys. 7.160), a przykładowy log wykonania (rys. 7.161).

```

test:
  runs-on: ubuntu-latest
  needs: build

  steps:
    - uses: actions/checkout@v3

    - name: Set up JDK 21
      uses: actions/setup-java@v3
      with:
        java-version: '21'
        distribution: 'temurin'

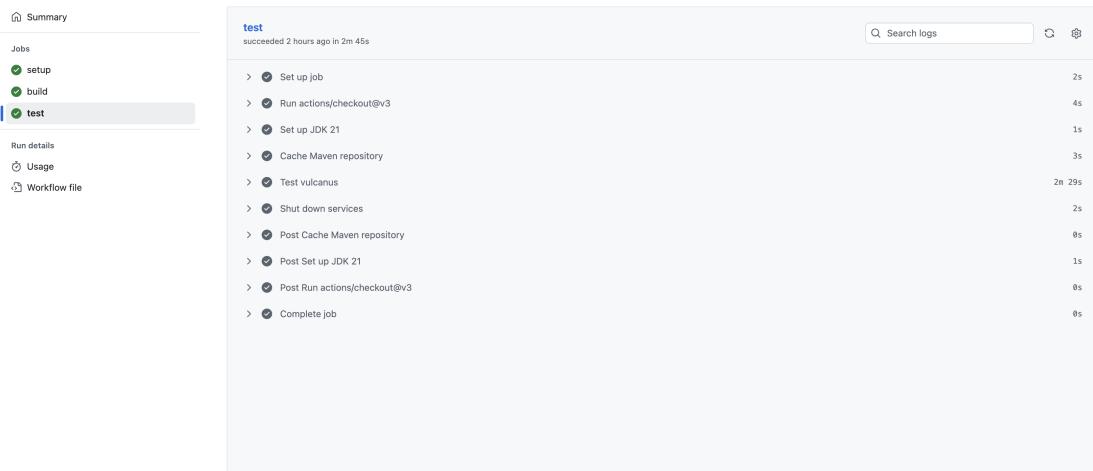
    - name: Cache Maven repository
      uses: actions/cache@v4
      with:
        path: |
          ~/.m2/repository
          ~/.m2/wrapper
        key: maven-${{ runner.os }}-${{ hashFiles('vulcanus/pom.xml') }}
        restore-keys: |
          maven-${{ runner.os }}-

    - name: Test vulcanus
      env:
        AZURE_STORAGE_CONNECTION_STRING: ${{ secrets.AZURE_STORAGE_CONNECTION_STRING }}
        MERKURY_LOCATIONQ_PROVIDER_API_KEY: ${{ secrets.MERKURY_LOCATIONQ_PROVIDER_API_KEY }}
      run: mvn -B test --file vulcanus/pom.xml

    - name: Shut down services
      run: docker compose -f vulcanus/docker/merkury/docker-compose.yml down

```

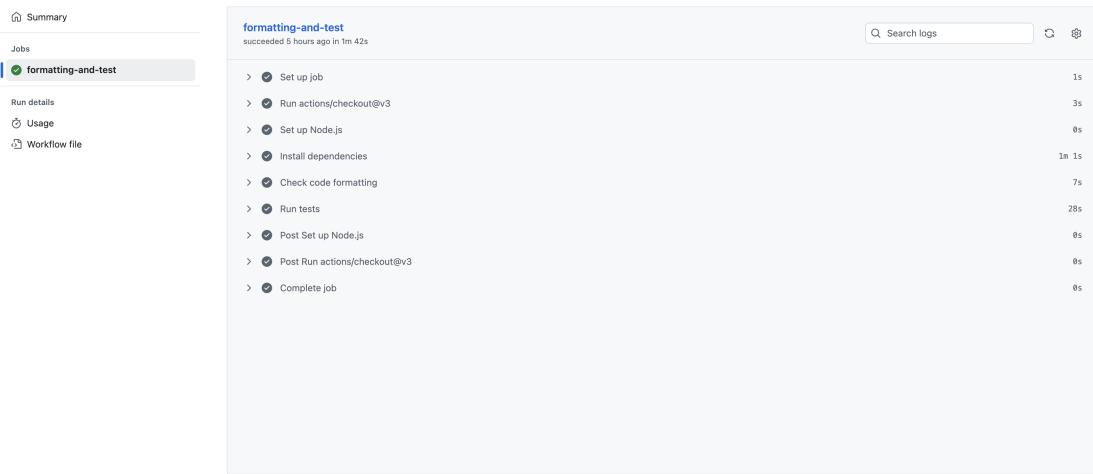
Rysunek 7.160: Fragment pliku `java-ci.yml`: definicja joba `test` (wraz z użyciem `secrets`).



Rysunek 7.161: Przykładowy log wykonania joba `test` w GitHub Actions.

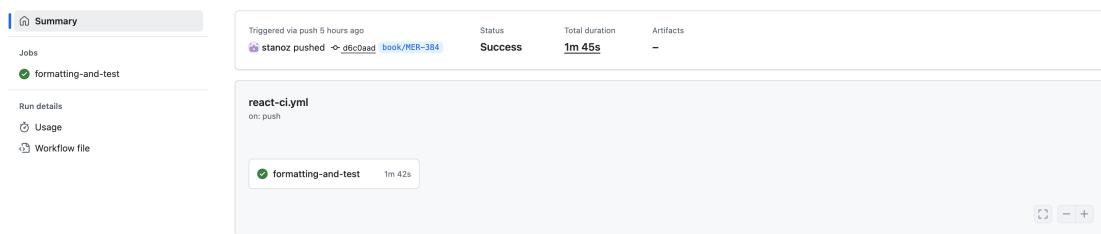
7.4.2 Pipeline frontendu

Workflow `frontendu` zdefiniowano w pliku `react-ci.yml`. Analogicznie do backendu uruchamiany jest dla zmian w katalogu `venus/**`, dla otwieranych *pull request* do gałęzi `master` i `develop` oraz w trybie `merge_group`, z pominięciem zmian w `book/**`. Fragment konfiguracji (wyzwalacze oraz `job`) przedstawia (rys. 7.162).

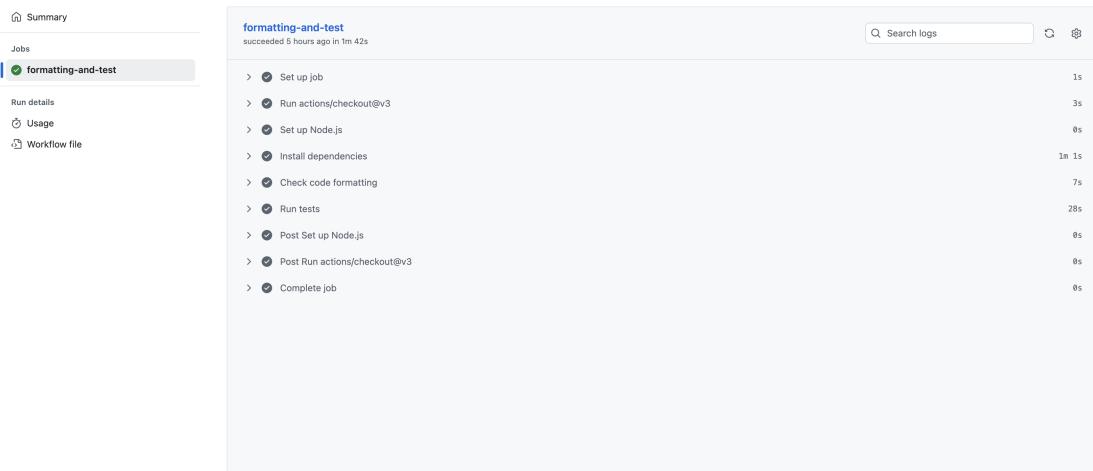


Rysunek 7.162: Fragment pliku `react-ci.yml`: konfiguracja workflow oraz joba `formatting-and-test`.

Workflow frontendu zawiera pojedynczy job `formatting-and-test`. Najpierw pobierany jest kod i ustawiana jest odpowiednia wersja [Node.js](#), następnie instalowane są zależności. Kolejnym krokiem jest weryfikacja formatowania kodu ([prettier](#)), a na końcu uruchamiane są testy jednostkowe oraz integracyjne. Podsumowanie przebiegu `workflow` przedstawia (rys. 7.163), natomiast przykładowy log wykonania joba (rys. 7.164).



Rysunek 7.163: Podsumowanie wykonania workflow frontendu (`react-ci.yml`) w GitHub Actions.



Rysunek 7.164: Przykładowy log wykonania joba `formatting-and-test` w GitHub Actions.

7.5 Implementacja WebSocket

Niniejszy rozdział prezentuje implementację protokołu [WebSocket](#). Rozdział przygotowano w oparciu o podane źródła:

- Specyfikacja protokołu WebSocket (RFC6455) [21].
- Dokumentacja Spring dotycząca obsługi WebSocket [22].
- Dokumentacja Spring dotycząca integracji STOMP [23].
- Dokumentacja biblioteki klienckiej SockJS [24].

7.5.1 Charakterystyka protokołu WebSocket

Na podstawie specyfikacji [RFC6455](#), opracowanej przez [IETF](#), protokół [WebSocket](#) można opisać jako mechanizm komunikacji pomiędzy aplikacją działającą w przeglądarce a [serwerem](#), który umożliwia dwukierunkową wymianę danych w czasie zbliżonym do rzeczywistego. Przeglądarka utrzymuje stałe połączenie z [serwerem](#), dzięki czemu może na bieżąco wysyłać dane, a [serwer](#) w dowolnym momencie przesyłać odpowiedzi, bez konieczności ciągłego odświeżania strony ani inicjowania

wielu osobnych zapytań [HTTP](#). Tego typu komunikacja jest możliwa wyłącznie z [serwerami](#), które świadomie ją dopuszczają, co zwiększa kontrolę nad bezpieczeństwem. Technologia [WebSocket](#) została zaprojektowana z myślą o aplikacjach webowych, takich jak czaty, powiadomienia w czasie rzeczywistym, aplikacje giełdowe czy gry sieciowe, które wymagają stałego kanału komunikacji z [serwerem](#), zamiast opierać się na wielu tradycyjnych połączeniach [HTTP](#).

Warto podkreślić, że sam protokół [WebSocket](#) definiuje przede wszystkim mechanizm utrzymywania stałego połączenia i przesyłania [ramek](#) danych. Nie narzuca natomiast formatu treści przesyłanych wiadomości, sposobu ich adresowania, routingu ani modelu [subskrypcji](#). Oznacza to, że po ustanowieniu połączenia [WebSocket](#) aplikacja musi samodzielnie ustalić, jak wygląda format wiadomości oraz w jaki sposób realizowane jest publikowanie i odbieranie zdarzeń.

Z tego powodu często stosuje się protokoły warstwy aplikacyjnej działające nad [WebSocket](#), np. [STOMP](#) (*Simple Text Oriented Messaging Protocol*). Zapewnia on ustandaryzowaną semantykę komunikacji w modelu wiadomościowym, w szczególności:

- **adresowanie wiadomości** do tzw. [destynacji](#),
- **mechanizm subskrypcji** ([SUBSCRIBE](#)) oraz dystrybucję wiadomości w stylu [publish–subscribe](#),
- **nagłówki i metadane** wiadomości (identyfikatory, typy zdarzeń, dodatkowy kontekst),
- **potwierdzanie odbioru** ([ACK](#)) i lepszą kontrolę nad niezawodnością dostarczenia,
- **obsługę błędów** na poziomie protokołu (ramki [ERROR](#)) oraz opcjonalne **heartbeat** do wykrywania zerwanych połączeń.

7.5.2 Zastosowanie [WebSocket](#) w naszym projekcie

Główną metodą komunikacji między [backendiem](#) a [frontendem](#) w naszej aplikacji jest [REST API](#) oparte na protokole [HTTP](#). Tego typu interfejs bardzo dobrze

sprawdza się w typowych scenariuszach, w których **klient** (**frontend**) inicjuje żądanie, a **serwer** (**backend**) zwraca odpowiedź, np. podczas logowania użytkownika, pobierania listy **spotów** czy dodawania komentarza na forum. Każda operacja ma wówczas postać pojedynczego, niezależnego wywołania **HTTP**, po którym połączenie jest zamkane.

Problem pojawia się jednak wtedy, gdy konieczna jest ciągła wymiana danych w czasie zbliżonym do rzeczywistego, nie tylko na żądanie **klienta**, lecz także z inicjatywy **serwera**. W naszej aplikacji taka potrzeba wystąpiła przy implementacji modułu czatu. Teoretycznie czat można zrealizować wyłącznie w oparciu o **REST API**, na przykład poprzez **polling**: **klient** cyklicznie wysyłałby zapytanie o nowe wiadomości, a **serwer** zwracałby aktualny stan konwersacji. Można również stosować **long polling**, w którym żądanie jest utrzymywane dłużej, aż pojawi się nowa wiadomość.

Takie podejście ma jednak istotne wady. Częste odpytywanie **serwera** generuje dużą liczbę żądań **HTTP**, obciążając zarówno **serwer**, jak i sieć, mimo że w wielu przypadkach odpowiedzi nie zawierają żadnych nowych danych. Dodatkowo wprowadza to opóźnienia: użytkownik zobaczy nową wiadomość dopiero przy kolejnym odpytywaniu, a nie w momencie jej wysłania. Rozwiązania oparte na **long pollingu** są z kolei bardziej złożone w implementacji i wciąż nie zapewniają tak naturalnego, dwukierunkowego kanału komunikacji, jakiego oczekuje się od wspólnego czatu.

Z tego powodu wszędzie tam, gdzie potrzebna jest stała, dwukierunkowa komunikacja w czasie rzeczywistym, stosuje się protokół **WebSocket**. W naszej aplikacji został on wykorzystany właśnie w module czatu, co pozwala na utrzymanie jednego trwałego połączenia pomiędzy **klientem** a **serwerem** i natychmiastowe dostarczanie wiadomości do wszystkich uczestników konwersacji, bez konieczności ciągłego odpytywania **serwera** za pomocą **REST API**.

7.5.3 Implementacja na backendzie

Spring Framework udostępnia dwa popularne podejścia do obsługi komunikacji **WebSocket**. Pierwsze to praca na „surowych” **ramkach WebSocket** co daje pełną

kontrolę, ale wymusza konieczność zaprojektowania własnego formatu komunikatów. Drugie wykorzystuje protokół **STOMP** jako warstwę aplikacyjną nad **WebSocket**. W niniejszym projekcie wybrano wariant z **STOMP**, ponieważ upraszcza on implementację czatu: pozwala korzystać z gotowych **destynacji**, **subskrypcji** oraz mechanizmu publikowania wiadomości.

7.5.3.1 Konfiguracja WebSocket

Na rysunku 7.165 przedstawiono konfigurację **WebSocket**. Definiuje ona endpoint do ustanowienia połączenia oraz dwa prefiksy destynacji: dla wiadomości przychodzących do aplikacji oraz dla wiadomości rozsyłanych do **klientów**.

```

@Configuration  ♫ Adam Langmesser
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override 1 usage ♫ Adam Langmesser
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker( ...destinationPrefixes: "/subscribe");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override 1 usage ♫ Adam Langmesser
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint( ...paths: "/connect")
            .setAllowedOriginPatterns("http://localhost:*")
            .withSockJS();
    }

    @Override 1 usage ♫ Adam Langmesser
    public void configureClientInboundChannel(ChannelRegistration registration) {
        registration.interceptors(securityContextChannelInterceptor());
    }

    @Bean ♫ Adam Langmesser
    public SecurityContextChannelInterceptor securityContextChannelInterceptor() {
        return new SecurityContextChannelInterceptor();
    }
}

```

Rysunek 7.165: Konfiguracja WebSocket po stronie backendu.

Najważniejsze elementy konfiguracji:

- `/connect` – endpoint ustanowienia połączenia (tzw. `handshake`); `withSockJS()` zapewnia mechanizm `fallback` (emulację WebSocket po HTTP), gdy `WebSocket` jest blokowany.
- `/app/*` – prefiks destynacji dla wiadomości wysyłanych od `klienta` do `serwera`.
- `/subscribe/*` – prefiks destynacji dla wiadomości wysyłanych od `serwera` do `klientów` (`klient` subskrybuje te kanały, a `serwer` publikuje tam komunikaty).

- **SecurityContextChannelInterceptor** – zapewnia dostęp do kontekstu bezpieczeństwa dla wiadomości przychodzących (**Backend** rozpoznaje, kto wysłał wiadomość).

7.5.3.2 Kontroler STOMP

Rysunek 7.166 pokazuje kontroler **STOMP**, który jest punktem wejścia dla nowych wiadomości wysyłanych przez użytkownika.

```
@Slf4j & Adam Langmesser
@Controller
@RequiredArgsConstructor
public class ChatStompCommunicationController {

    private final ChatStompCommunicationService chatStompCommunicationService;
    private final ChatService chatService;

    @MessageMapping("/send/{chatId}/message") & Adam Langmesser
    public void sendChatMessage(@DestinationVariable String chatId, @Payload IncomingChatMessageDto message) {
        log.debug("Received message for chat: {}, message: {}", chatId, message);
        var chatMessageDtoToBroadcast = chatService.saveChatMessage(message);
        chatStompCommunicationService.broadcastChatMessageToAllChatParticipants(chatMessageDtoToBroadcast);
        chatStompCommunicationService.broadcastACKVersionToSender(chatMessageDtoToBroadcast, message.optimisticMessageUUID());
    }
}
```

Rysunek 7.166: Kontroler STOMP odbierający wiadomości czatu.

Kontroler udostępnia wejściową **destynację**: `/app/send/{chatId}/message`.
Jej rola jest następująca:

1. odebrać wiadomość od **klienta**,
2. zlecić zapis wiadomości do **bazy danych** za co odpowiada serwis domenowy czatu,
3. uruchomić rozesłanie wiadomości,
4. oddelegować wysłanie potwierdzenia (**ACK**) do nadawcy wiadomości.

7.5.3.3 Serwis dystrybucji wiadomości

Rysunek 7.167 przedstawia serwis, który odpowiada za publikowanie komunikatów na kanały **subskrypcji** `/subscribe/*`. Serwis ten jest wykorzystywany zarówno

przez kontroler STOMP, jak i przez inne elementy logiki modułu czatu.

```
@Slf4j 3 usages  Adam Langmesser *
@Service
@RequiredArgsConstructor
public class ChatStompCommunicationService {

    private final SimpMessagingTemplate messagingTemplate;
    private final ChatRepository chatRepository;
    private final CustomUserDetailsService customUserDetailsService;

    Broadcasts a chat message to all participants of the chat.
    The STOMP endpoint for this method is dynamically created based on the username of each
    chat participant. This allows each participant to receive messages sent to the chat they are part
    of.

    Each User should listen on its own channel /subscribe/chats{username}.
    Each User has one universal channel to receive new messages in real time for all chats they are
    part of.

    Params: chatMessageDto – the chat message to be broadcasted

    @Transactional 2 usages  Adam Langmesser *
    public void broadcastChatMessageToAllChatParticipants(ChatMessageDto chatMessageDto) {
        var chatParticipants = chatRepository.findById(chatMessageDto.chatId())
            .orElseThrow(() -> new IllegalArgumentException("Chat not found"))
            .getParticipants();

        log.info("Broadcasting chat message to {} participants: {}", (long) chatParticipants.size(), chatMessageDto);
        chatParticipants.forEach( participant -> messagingTemplate.convertAndSend( destination: "/subscribe/chats/"
            + participant.getUser().getUsername(), chatMessageDto);
    }

    @Transactional 1 usage  Adam Langmesser *
    public void broadcastACKVersionToSender(ChatMessageDto chatMessageDto, String optimisticMessageUUID) {
        var currentUserUsername = customUserDetailsService.loadUserDetailsFromSecurityContext().getUsername();

        log.info("Broadcasting ack for message with id: {} for chat id: {} to {}", chatMessageDto.id(),
            chatMessageDto.chatId(), currentUserUsername);
        var ackChatMessageDto = new ChatMessageAckDto(
            chatMessageDto, optimisticMessageUUID
        );
        messagingTemplate.convertAndSend( destination: "/subscribe/chats/ack/" + currentUserUsername, ackChatMessageDto);
    }
}
```

Rysunek 7.167: Serwis odpowiedzialny za komunikację za pomocą WebSocket.

Serwis realizuje dwa typy wysyłki:

- **Broadcast wiadomości do uczestników czatu** – po zapisaniu wiadomości serwer pobiera listę uczestników czatu i wysyła wiadomość do każdego z nich.
- **ACK do nadawcy** – serwer wysyła potwierdzenie na osobny kanał nadawcy,

aby **frontend** mógł powiązać wiadomość zapisaną w **bazie danych** z odpowiadającą jej wersją **optimistyczną**.

7.5.3.4 Zestawienie wykorzystywanych destynacji

Poniżej zestawiono wszystkie **destynacje** używane w komunikacji **WebSocket**:

- **/connect – endpoint** ustanowienia połączenia **WebSocket** (tzw. **handshake**).
- **/app/send/{chatId}/message** – wejście do backendu: **klient** wysyła nową wiadomość do czatu o identyfikatorze **chatId**.
- **/subscribe/chats/{username}** – kanał użytkownika: **klient subskrybuje** i odbiera nowe wiadomości ze wszystkich czatów, w których uczestniczy.
- **/subscribe/chats/ack/{username}** – kanał potwierdzeń **ACK**: **klient** odbiera potwierdzenie zapisania wiadomości.

Dodatkowo stosowane są dwa **prefiksy** porządkujące komunikację:

- **/app/*** – wiadomości od **klienta** do aplikacji,
- **/subscribe/*** – wiadomości od aplikacji do **klientów** (kanały **subskrypcji**).

7.5.4 Implementacja na frontendzie

Na **frontendzie** zastosowano podejście oparte na jednej, współdzielonej instancji serwisu komunikacji, udostępnianej poprzez **kontekst React'a**. Takie podejście wynikało z chęci zapewnienia, aby w przyszłości również inne moduły aplikacji (poza czatem) mogły w prosty sposób korzystać z już istniejącego połączenia **WebSocket**, bez konieczności implementowania odrębnej logiki połączenia, subskrypcji i obsługi zdarzeń. Rozwiążanie składa się z:

- **WebSocketService** – utrzymuje pojedyncze połączenie **STOMP** działające nad **WebSocket**, obsługuje **reconnect**, zarządza **subskrypcjami** oraz umożliwia wysyłanie komunikatów do wskazanych **destynacji**,

- `hook useWebSocket` – zapewnia warstwę użycia w komponentach: automatycznie rejestruje i usuwa `subskrypcje` zgodnie z `cyklem życia komponentu` oraz udostępnia funkcję wysyłania wiadomości i status połączenia,
- `WebSocketProvider` – udostępnia instancję serwisu przez `kontekst React'a` i steruje cyklem życia połączenia na poziomie aplikacji (łączy po zalogowaniu i rozłącza po wylogowaniu),
- `createChatSubscription` – definiuje `destynacje` oraz logikę obsługi odbieranych komunikatów (`callback`), aktualizując `stan` w `Redux`. Rozwiążanie ma charakter rozszerzalny: kolejne moduły mogą dostarczać własne fabryki, które następnie są dodawane w `WebSocketProvider` do wspólnej listy subskrypcji, bez modyfikowania logiki połączenia.

7.5.4.1 Serwis komunikacji WebSocket

```
Zarządza jednym połączeniem STOMP/WebSocket:  
• inicjalizacja i automatyczne reconnect  
• jednoczesne subskrypcje (wznowione po reconnect)  
• publikacja wiadomości  
  
✓ export class WebSocketService { Show usages ▾ Adam Langmesser  
    private client: Client;  
    private subscriptions: Map<string, StompSubscription> = new Map<string, StompSubscription>();  
  
    Przechowuje subskrypcje oczekujące na moment, kiedy połączenie WebSocket/STOMP stanie  
    się aktywne. Prosty mechanizm buforowania  
  
    private pendingSubs: Map<string, (msg: IMessage) => void> = new Map<string, (msg: IMessage) => void>();  
    private isConnected: boolean = false;  
  
Zarządza jednym połączeniem STOMP/WebSocket:  
• inicjalizacja i automatyczne reconnect  
• jednoczesne subskrypcje (wznowione po reconnect)  
• publikacja wiadomości  
  
Params: brokerURL – adres SockJS endpointu, np. "http://localhost:8080/ws ↴"  
  
✓ constructor(private brokerURL: string) { Show usages ▾ Adam Langmesser  
    this.client = new Client({  
        webSocketFactory: () : WebSocket => new SockJS(this.brokerURL),  
        reconnectDelay: 5000,  
        debug: (msg: string) : void => console.debug(message: "[STOMP]", msg),  
        onStompError: this.onError.bind(this),  
        onWebSocketClose: this.onClose.bind(this),  
    });  
  
    this.client.onConnect = this.onConnect.bind(this);  
}
```

Rysunek 7.168: Serwis komunikacji WebSocket po stronie frontendu (część 1/4).

```

Jeśli połączenie jeszcze nie zostało nawiązane, to wywołanie connect() je utworzy.

Jeśli połączenie już istnieje (czyli metoda została już wcześniej wywołana i klient jest „aktywny”), to kolejne wywołanie connect() nie spowoduje błędu ani podwójnego połączenia — po prostu nic się nie zmieni.

connect(): void { Show usages  Adam Langmesser
    if (!this.client.active) {
        this.client.activate();
    }
}

Rozłącza i czyści wszystkie subskrypcje.

disconnect(): void { Show usages  Adam Langmesser
    this.client.deactivate();
    this.cleanupAll();
    this.isConnected = false;
}

Internal: Gdy STOMP się połączy – wznowienie pendingSubs.

private onConnect(): void { Show usages  Adam Langmesser
    this.isConnected = true;
    this.pendingSubs.forEach((cb : (msg: IMessage) => void , dest : string ) : void  => {
        const sub : StompSubscription  = this.client.subscribe(dest, cb);
        this.subscriptions.set(dest, sub);
    });
    this.pendingSubs.clear();
}

Internal: Obsługa niespodziewanego zamknięcia socketu.

private onClose(evt: CloseEvent): void { Show usages  Adam Langmesser
    console.warn( message: "WebSocket closed:", evt.reason);
    this.isConnected = false;
    // stomp.js sam spróbuje reconnect
}

```

Rysunek 7.169: Serwis komunikacji WebSocket po stronie frontendu (część 2/4).

```

Internal: Obsługa błędów STOMP frame z brokerem.

private onError(frame: Frame): void { Show usages & Adam Langmesser
    console.error( message: "STOMP error:", frame.headers["message"], frame.body);
}

Subskrybuje dany topic.

Params: destination – np. "/topic/chat"
        callback – wywoływany na każdy komunikat

subscribe(destination: string, callback: (msg: IMessage) => void): void { Show usages
    // odsubskrybij poprzednią, jeśli była
    if (this.subscriptions.has(destination)) {
        this.subscriptions.get(destination)!.unsubscribe();
        this.subscriptions.delete(destination);
    }
    if (this.isConnected) {
        const sub :StompSubscription = this.client.subscribe(destination, callback);
        this.subscriptions.set(destination, sub);
    } else {
        // zapisz do wznowienia po connect
        this.pendingSubs.set(destination, callback);
    }
}

Usuwa subskrypcję z danego topicu.

Params: destination – identyfikator subskrypcji

unsubscribe(destination: string): void { Show usages & Adam Langmesser
    if (this.subscriptions.has(destination)) {
        this.subscriptions.get(destination)!.unsubscribe();
        this.subscriptions.delete(destination);
    }
    this.pendingSubs.delete(destination);
}

```

Rysunek 7.170: Serwis komunikacji WebSocket po stronie frontendu (część 3/4).

```

Publikuje wiadomość JSON-ujac payload.

Params: destination – np. "/app/chat.send"
        payload – dowolny obiekt
        headers – dodatkowe nagłówki STOMP

publish( Show usages  Adam Langmesser
    destination: string,
    payload: any,
    headers: Record<string, string> = {},
): void {
    if (!this.isConnected) {
        console.warn(
            message: `STOMP not connected, cannot publish to ${destination}`,
        );
        return;
    }
    this.client.publish({
        destination,
        body: JSON.stringify(payload),
        headers,
    });
}

Czy klient jest aktualnie połączony?

get connected(): boolean { Show usages  Adam Langmesser
    return this.isConnected;
}

Internal: Pełne sprzątanie przy disconnect.

private cleanupAll(): void { Show usages  Adam Langmesser
    this.subscriptions.forEach((sub: StompSubscription) : void => sub.unsubscribe());
    this.subscriptions.clear();
    this.pendingSubs.clear();
}
}

```

Rysunek 7.171: Serwis komunikacji WebSocket po stronie frontendu (część 4/4).

Klasa `WebSocketService` stanowi warstwę po stronie `frontendu`, która zarządza jednym połączeniem `STOMP` działającym nad `WebSocket`. Do ustanowienia transportu wykorzystano `SockJS`, a właściwy protokół wiadomościowy realizuje `klient`

STOMP z biblioteki [StompJS](#).

Główne odpowiedzialności klasy:

- inicjalizacja i utrzymanie połączenia ([handshake](#) oraz automatyczny [reconnect](#)),
- rejestrowanie i usuwanie [subskrypcji STOMP](#),
- buforowanie subskrypcji, które zostały zadeklarowane przed nawiązaniem połączenia,
- publikowanie wiadomości do wskazanej [destynacji](#) oraz dołączanie [nagłówków](#).

Pola klasy:

- `client` – instancja [klient STOMP](#), która realizuje połączenie i wymianę wiadomości.
- `subscriptions` – mapa aktywnych subskrypcji (klucz: [destynacja](#), wartość: uchwyt subskrypcji).
- `pendingSubs` – bufor oczekujących subskrypcji: `destynacja` → [callback](#).
- `isConnected` – flaga stanu połączenia.

Konstruktor `constructor(brokerURL)`:

- konfiguruje `WebSocketFactory` tak, aby transport był realizowany przez [SockJS](#),
- ustawia opóźnienie automatycznego [reconnect](#) (w kodzie: `reconnectDelay = 5000 ms`),
- podpina obsługę zdarzeń: poprawne połączenie, błąd protokołu oraz zamknięcie połączenia.

Metody klasy:

`connect()`: uruchamia połaczenie wywołując `activate()` na kliencie STOMP. Jeżeli klient jest już aktywny, metoda nie powoduje utworzenia drugiego połączenia.

`disconnect()`: rozłącza klienta (`deactivate()`), usuwa wszystkie subskrypcje (`cleanup`) oraz ustawia `isConnected = false`. Zapobiega to pozostawieniu nieużywanych kanałów po stronie `frontendu`.

`onConnect()`: jest wywoływana po udanym połączeniu i odpowiada za „odtworzenie” subskrypcji: zawartość `pendingSubs` zostaje zamieniona na aktywne subskrypcje w `subscriptions`, po czym bufor jest czyszczony.

`subscribe(destination, callback)`: zakłada subskrypcję na danej `destynacji`. Jeżeli subskrypcja o tej samej destynacji już istnieje, jest usuwana (uniknięcie wielokrotnego odbioru). Gdy połączenie nie jest aktywne, subskrypcja trafia do bufora `pendingSubs` i zostanie zarejestrowana po połączeniu.

`publish(destination, payload, headers)`: publikuje wiadomość tylko wtedy, gdy połączenie jest aktywne. Dane `payload` są poddawane `serializacji` do formatu `JSON`, a następnie wysyłane do `backendu`. Opcjonalnie dołączane są `nagłówki` protokołu STOMP.

7.5.4.2 Mechanizm rejestracji subskrypcji

```
Definicja jednej subskrypcji.

export interface SubscriptionDef { Show usages & Adam Langmesser
    Adres STOMP tematu, np. "/topic/chat/123"
    destination: string;
    Callback na każdy otrzymany komunikat
    callback: (msg: IMessage) => void;
}

Opcje inicjalizacji hooka.

• subscriptions: tablica kanałów do zalożenia od razu

export interface UseWebSocketOptions { Show usages & Adam Langmesser
    subscriptions?: SubscriptionDef[];
}

Hook do obsługi WebSocketów z protokołem STOMP:

• umożliwia przekazanie opcjonalnej listy subskrypcji - można za pomocą tego hooka zasubskrybować
się do różnych kanałów ale sugeruję subskrybowanie się do kanałów w komponencie
WebSocketContext.tsx tak jak robię to dla chatów
• automatycznie zakłada i usuwa subskrypcje podczas montowania/odmontowywania komponentu
• zwraca funkcję publish() do wysyłania wiadomości oraz flagę connected informującą o statusie
połączenia

Params: options – konfiguracja hooka

export function useWebSocket(options: UseWebSocketOptions = {}) :{ publish: (destination: string, payload:... ) : void; connected: boolean } { Show usages & Adam Langmesser
    const { subscriptions = [] } = options;
    const ws : WebSocketService = useWebSocketService();

    // Lifecycle subskrypcji: mount → subscribe, unmount → unsubscribe
    useEffect(() : void => {
        const cleanups : () => void[] = subscriptions.map(({ destination, callback } : SubscriptionDef ) : () => void => {
            ws.subscribe(destination, callback);
            // zwrócić funkcję do unsubscribe
            return () : void => ws.unsubscribe(destination);
        });
        return () : void => cleanups.forEach((fn : () => void ) : void => fn());
        // każda zmiana 'destination' powoduje teardown i re-sub
    }, [ws, ...subscriptions.map((s : SubscriptionDef ) : string => s.destination)]);
}
```

Rysunek 7.172: Mechanizm rejestracji subskrypcji w cyklu życia komponentu (część 1/2).

```

    /**
     * Wysyła wiadomość do brokeru.
     */
    function publish( Show usages  Adam Langmesser
      destination: string,
      payload: any,
      headers?: Record<string, string>,
    ) : void {
      ws.publish(destination, payload, headers);
    }

    /** Status połączenia */
    const connected :boolean = ws.connected;

    return { publish, connected };
}

```

Rysunek 7.173: Mechanizm rejestrowania subskrypcji w cyklu życia komponentu (część 2/2).

Hook `useWebSocket` upraszcza korzystanie z komunikacji `WebSocket` w `React`. Zapewnia automatyczne zakładanie i zdejmowanie `subskrypcji` zgodnie z `cyklem życia` komponentu oraz udostępnia funkcję do wysyłania wiadomości.

Typ `SubscriptionDef`:

- `destination` – `destynacja` protokołu `STOMP`,
- `callback` – `callback` uruchamiany dla każdej odebranej wiadomości.

Działanie `useWebSocket(options)`:

- pobiera instancję serwisu z `useWebSocketService()` (z `kontekstu`),
- w `useEffect` zakłada subskrypcje podczas `montowania` komponentu oraz usuwa je podczas `odmontowania` (lub przy zmianie zależności).

7.5.4.3 Kontekst i zarządzanie połączeniem

```
const WS_URL :string = process.env.REACT_APP_WS_URL || "http://localhost:8080/connect";
const wsService = new WebSocketService(WS_URL);
const WebSocketContext :Context<WebSocketService> = createContext<WebSocketService>(wsService);
export const WebSocketProvider: React.FC<{ children: React.ReactNode }> = ({ children, ...props }) => {
  const { isLoggedIn, username } = useSelectorTyped<RootState>((state) => state.account);
  const selectedChatId :number | null = useSelectorTyped<(state: AccountSliceProps) => state.chats.selectedChatId;
  const dispatch :ThunkDispatch<AccountSliceProps> = useDispatchTyped();
  const selectedRef :MutableRefObject<number | null> = useRef<number | null>(initialValue: null);
  useEffect(() :void => {
    selectedRef.current = selectedChatId;
  }, [selectedChatId]);
  useEffect(() :void => {
    if (isLoggedIn) wsService.connect();
    else wsService.disconnect();
  }, [isLoggedIn]);
  useEffect(() :()=> void | undefined => {
    if (!isLoggedIn || !username) return;
    const getSelectedChatId :()=> number | null = () :number | null => store.getState().chats.selectedChatId;
    const allSubs: SubscriptionDef[] = [
      createChatSubscription(username, dispatch, getSelectedChatId),
    ];
    const cleanups :()=> void[] = allSubs.map((sub :SubscriptionDef) :()=> void => {
      wsService.subscribe(sub.destination, sub.callback);
      return () :void => wsService.unsubscribe(sub.destination);
    });
    return () :void => cleanups.forEach((fn :()=> void) :void => fn());
  }, [isLoggedIn, username, dispatch]);
  return (
    <WebSocketContext.Provider value={wsService}>
      {children}
    </WebSocketContext.Provider>
  );
};

export const useWebSocketService :()=> WebSocketService = () :WebSocketService => {
  const context = useContext(WebSocketContext);
  if (!context) throw new Error("useWebSocketService must be used within a WebSocketProvider");
  return context;
}
```

Rysunek 7.174: Udostępnienie serwisu komunikacji oraz zarządzanie cyklem życia połączenia po stronie frontendu.

Plik definiuje [kontekst React'a](#), który udostępnia w całej aplikacji jedną wspólną instancję `WebSocketService`. Dzięki temu wiele komponentów może korzystać z jednego połączenia [WebSocket](#), zamiast tworzyć osobne połączenia niezależnie.

Adres endpointu: `WS_URL` jest pobierany ze [zmiennej środowiskowej](#) `REACT_APP_WS_URL`, a w razie braku ustawienia przyjmuje wartość domyślną `http://localhost:8080/connect`.

Współdzielona instancja serwisu: `wsService` jest tworzony raz (wzorzec [Singleton](#)) poza komponentem providera. Zapewnia to, że aplikacja nie tworzy nowego połączenia przy każdym renderowaniu.

Zarządzanie połączeniem w WebSocketProvider:

- po zmianie `isLoggedIn` provider łączy się (`connect`) lub rozłącza (`disconnect`),
- po zalogowaniu rejestrowane są [subskrypcje](#) zależne od użytkownika (np. kanał czatów),
- podczas sprzątania wykonywany jest [cleanup subskrypcji](#).

Dodatkowo używany jest `dispatch`, aby aktualizować [stan](#) aplikacji w [Redux'sie](#), w reakcji na wiadomości przychodzące w czasie rzeczywistym.

7.5.4.4 Definicja subskrypcji czatu

```
export function createChatSubscription( Show usages Adam Langmesser
  username: string,
  dispatch: AppDispatch,
  getSelectedChatId: () => number | null,
): SubscriptionDef {
  return {
    destination: `/subscribe/chats/${username}`,
    callback: (msg: IMessage) : void => {
      const payload = JSON.parse(msg.body) as ChatMessageDto;

      dispatch(
        chatActions.setLastMessage({
          chatId: payload.chatId,
          message: payload,
        }),
      );

      const selected :number | null = getSelectedChatId();
      if (selected !== payload.chatId) {
        dispatch(chatActions.markNew(payload.chatId));
      } else {
        dispatch(chatActions.clearNew(payload.chatId));
      }
    },
  };
}
```

Rysunek 7.175: Definicja subskrypcji czatu i aktualizacja stanu w Redux.

Plik definiuje funkcję fabrykującą obiekt typu `SubscriptionDef`, czyli pojedynczą subskrypcję STOMP dla kanału wiadomości czatu danego użytkownika. Jej celem jest odbieranie nowych wiadomości w czasie rzeczywistym oraz aktualizacja stanu aplikacji w Redux'sie.

Funkcja `createChatSubscription(username, dispatch, getSelectedChatId)`:

- buduje destynację subskrypcji: `/subscribe/chats/{username}`,

- w `callbacku` wykonuje `deserializację` wiadomości z `JSON` (`JSON.parse`),
- wysyła `akcję` do `Redux` w celu aktualizacji ostatniej wiadomości w danym czacie,
- porównuje `payload.chatId` z aktualnie wybranym czatem i odpowiednio oznacza czat jako „nowy” lub czyści to oznaczenie.

Zastosowanie funkcji `getSelectedChatId` (zamiast wartości przechowywanej w komponentach) wynika z asynchronicznego charakteru odbioru wiadomości: `callback` wykonuje się w chwili nadejścia komunikatu, dlatego odczyt bieżącego `stanu` bezpośrednio ze `store` minimalizuje ryzyko użycia nieaktualnych danych.

7.5.5 Przebieg komunikacji

Rozdział 8

Testy

W niniejszym rozdziale opisano proces testowania aplikacji [frontendowej](#) oraz [backendowej](#), obejmujący testy jednostkowe, integracyjne i end-to-end (E2E), a także przedstawiono uzyskane wyniki oraz wnioski.

8.1 Testy jednostkowe

Testy jednostkowe (ang. *unit tests*) to automatyczne testy weryfikujące poprawność działania najmniejszych fragmentów kodu (funkcji, metod lub pojedynczych komponentów) w możliwie pełnej izolacji od reszty systemu. Ich celem jest szybkie wykrywanie błędów logicznych, walidacja zachowania dla danych typowych i brzegowych oraz ułatwienie refaktoryzacji poprzez zapewnienie, że wprowadzone zmiany nie psują wcześniej działających elementów.

W praktyce izolacja ta jest uzyskiwana przez zastępowanie zależności zewnętrznych (bazy danych, systemu plików czy usług zewnętrznych [API](#)) atrapami lub obiektami typu *mock*. Dzięki temu testy jednostkowe mogą być uruchamiane często i szybko, stanowiąc podstawowy element kontroli jakości w procesie tworzenia oprogramowania.

Do automatycznego uruchamiania testów jednostkowych wykorzystano [GitHub Actions](#), co umożliwiło ich cykliczne wykonywanie w ramach procesu [CI/CD](#) (przy każdym *push* lub *pull request*). Łącznie przygotowano 273 testy jednostkowe, w tym 211 dla [frontenu](#) oraz 62 dla [backendu](#). Wszystkie przygotowane testy zakończyły się powodzeniem (rys. 8.1 oraz rys. 8.2–8.3).

W warstwie **frontendu** utworzono łącznie 211 testów jednostkowych (w 23 plikach testowych). Testy opracowano dla następujących modułów i komponentów:

- AddedSpots,
- Comments,
- FavoriteSpots,
- Movies,
- Photos (w tym DateChooser oraz SortDropdown),
- Profile (ProfileForViewer, UserOwnProfile),
- Settings,
- Social (SocialCard, SocialForViewer, UserOwnSocial),
- Login,
- Register,
- CurrentViewSpotsList,
- SearchedSpotsList,
- SearchedSpotsSortingForm,
- Sidebar,
- SpotDetails,
- SpotsNameSearchBar,
- UserLocationPanel,
- ZoomPanelControl.

W testach weryfikowano poprawność renderowania komponentów, obecność i treść kluczowych elementów interfejsu. Przykładowy wynik uruchomienia testów jednostkowych frontendu przedstawiono (rys. 8.1).

```
Test Files 23 passed (23)
Tests 211 passed (211)
Start at 17:13:47
Duration 5.88s (transform 1.33s, setup 1.17s, collect 20.17s, tests 5.57s, environment 8.13s, prepare 1.08s)
```

Rysunek 8.1: Wynik uruchomienia testów jednostkowych warstwy frontendowej

Dla backendu przygotowano łącznie 62 testy jednostkowe, napisane dla serwisów:

- FollowersService,
- CommentsService,
- MediaService,
- AddSpotService,
- SettingsService,
- ProfileService,
- FavoriteSpotService,
- FriendsService,
- RegisterService.

Testy te służyły do potwierdzenia poprawnego działania metod serwisowych, w tym obsługi przypadków brzegowych oraz walidacji danych wejściowych. Dodatkowo sprawdzano poprawność współpracy z wybranymi zewnętrznymi interfejsami API, przy zachowaniu izolacji logiki aplikacyjnej (poprzez zastępowanie zależności atrapami). Zestaw uruchomionych testów backendu pokazano (rys. 8.2), natomiast podsumowanie ich wykonania przedstawiono (rys. 8.3).

✓ account (com.merkury.vulcanus.features)	1 sec 182 ms
> ✓ FollowersServiceTest	595 ms
> ✓ CommentsServiceTest	34 ms
> ✓ MediaServiceTest	33 ms
> ✓ AddSpotServiceTest	364 ms
> ✓ SettingsServiceTest	60 ms
> ✓ ProfileServiceTest	8 ms
> ✓ FavoriteSpotServiceTest	19 ms
> ✓ FriendsServiceTest	38 ms
> ✓ RegisterServiceTest	31 ms

Rysunek 8.2: Zestaw testów jednostkowych uruchomionych dla warstwy backenedowej

✓ 62 tests passed 62 tests total, 1 sec 182 ms

Rysunek 8.3: Podsumowanie uruchomienia testów jednostkowych warstwy backenedowej

8.2 Testy integracyjne

Testy integracyjne (ang. *integration tests*) to testy automatyczne służące do weryfikacji poprawnej współpracy kilku modułów lub warstw aplikacji, które w testach jednostkowych były badane oddzielnie. Ich celem jest potwierdzenie, że komponenty poprawnie komunikują się ze sobą, przekazują dane, obsługują zależności oraz że cały fragment funkcjonalności działa spójnie w warunkach zbliżonych do rzeczywistych.

W przeciwnieństwie do testów jednostkowych, testy integracyjne zwykle obejmują większy zakres systemu i mogą wykorzystywać rzeczywiste implementacje wybranych zależności (warstwę dostępu do danych, konfigurację kontenera zależności lub uruchomienie kontekstu aplikacji), ewentualnie częściowo zastępowane atrapami w celu kontroli środowiska testowego. Dzięki temu umożliwiają wykrycie problemów wynikających z integracji (błędnej konfiguracji, niezgodnych kontraktów lub niepoprawnych interakcji pomiędzy komponentami).

Do automatycznego uruchamiania testów integracyjnych wykorzystano [GitHub Actions](#), co umożliwiło ich cykliczne wykonywanie w ramach procesu [CI/CD](#) (przy każdym *push* lub *pull request*). Łącznie przygotowano 102 testy integracyjne, w tym 52 dla [frontendu](#) oraz 50 dla [backendu](#). Wszystkie przygotowane testy zakończyły się powodzeniem (rys. 8.4 oraz rys. 8.5–8.6).

W warstwie [frontendu](#) utworzono łącznie 52 testy integracyjne (w 11 plikach testowych). Testy te pozwalały zweryfikować poprawną współpracę wybranych komponentów w ramach większych fragmentów interfejsu oraz spójność przepływu danych pomiędzy nimi. Dodatkowo sprawdzano poprawność zmian stanu aplikacji w odpowiedzi na interakcje użytkownika (kliknięcia) oraz działanie mechanizmu [infinite scroll](#), w tym poprawne dociąganie i prezentację kolejnych elementów listy.

Przykładowy wynik uruchomienia testów integracyjnych [frontendu](#) przedstawiono (rys. 8.4).

```
Test Files 11 passed (11)
Tests 52 passed (52)
Start at 17:17:18
Duration 4.39s (transform 942ms, setup 681ms, collect 10.80s, tests 4.24s, environment 4.29s, prepare 552ms)
```

Rysunek 8.4: Wynik uruchomienia testów integracyjnych warstwy frontendowej

Dla [backendu](#) przygotowano łącznie 50 testów integracyjnych. Testy te służyły do potwierdzenia poprawnej współpracy kluczowych warstw aplikacji, w tym poprawnego uruchomienia kontekstu aplikacji. Zestaw uruchomionych testów [backendu](#) pokazano (rys. 8.5), natomiast podsumowanie ich wykonania przedstawiono (rys. 8.6).

▼	✓ <default package>	14 sec 877 ms
>	✓ TestControllerWithApplicationContextStartupTest	307 ms
>	✓ AccountControllerOAuth2WithServerStartupTest	34 ms
>	✓ SpotControllerWithServerStartupTest	449 ms
>	✓ UserDashboardControllerWithServerStartu	12 sec 962 ms
>	✓ JwtGeneratorTest	8 ms
>	✓ AccountControllerWithServerStartupTest	949 ms
>	✓ RegisterServiceTest	168 ms

Rysunek 8.5: Zestaw testów integracyjnych uruchomionych dla warstwy backenedowej

✓ 50 tests passed	50 tests total, 14 sec 877 ms
-------------------	-------------------------------

Rysunek 8.6: Podsumowanie uruchomienia testów integracyjnych warstwy backenedowej

8.3 Testy end-to-end (E2E)

Testy end-to-end (E2E) (ang. *end-to-end tests*) to testy automatyczne weryfikujące działanie aplikacji jako całości z perspektywy użytkownika. Obejmują one pełny przepływ realizacji funkcjonalności, od interakcji w interfejsie (nawigacja, kliknięcia, wypełnianie formularzy) aż po komunikację z warstwą serwerową i przetwarzanie danych, dzięki czemu pozwalają potwierdzić poprawność działania kluczowych scenariuszy biznesowych w warunkach zbliżonych do rzeczywistego użycia systemu.

W odróżnieniu od testów jednostkowych i integracyjnych, testy E2E uruchamiane są na działającej aplikacji i wykorzystują rzeczywistą przeglądarkę, co zwiększa ich realistyczność i dokładność.

sza wiarygodność weryfikacji, ale jednocześnie zwykle wiąże się z dłuższym czasem wykonania oraz większą wrażliwością na zmiany w interfejsie.

Testy end-to-end (E2E) zrealizowano z wykorzystaniem narzędzia Cypress. W przeciwieństwie do testów jednostkowych i integracyjnych, testy E2E nie były uruchamiane w ramach [GitHub Actions](#) (procesu [CI/CD](#)), lecz wykonywano je lokalnie w środowisku deweloperskim. Testy uruchamiano na działającej aplikacji, symuluując rzeczywiste działania użytkownika w przeglądarce, co pozwoliło zweryfikować pełny przepływ od interfejsu [frontendowego](#) do warstwy [backendowej](#).

Łącznie przygotowano 40 testów E2E (w 9 plikach), a wszystkie zakończyły się powodzeniem (rys. 8.7). Zaimplementowane testy stanowią bezpośrednią realizację scenariuszy testowych opisanych w sekcji 8.4, obejmujących zarówno przypadki z użyciem danych mockowanych (poprzez przechwytywanie żądań HTTP), jak i scenariusze wykonywane na rzeczywistym backendzie.

Testy E2E opracowano dla kluczowych obszarów aplikacji:

- account (logowanie, rejestracja),
- user-dashboard/add-spot (lista, *infinite scroll*, dodawanie miejsca),
- user-dashboard/comments (lista, sortowanie),
- user-dashboard/favorite-spots (listy, przełączanie typów, *infinite scroll*),
- user-dashboard/movies (lista, sortowanie, *infinite scroll*),
- user-dashboard/photos (lista, sortowanie, *infinite scroll*),
- user-dashboard/profile (widok profilu, nawigacja, akcje społecznościowe),
- user-dashboard/settings (edytowanie danych konta oraz ograniczenia dla kont OAuth),
- user-dashboard/social (listy: friends/followed/followers, zaproszenia, *infinite scroll*).

W testach weryfikowano poprawność realizacji scenariuszy użytkownika, w tym nawigację pomiędzy widokami, wykonywanie operacji w interfejsie (kliknięcie i wypełnianie formularzy) oraz poprawną aktualizację stanu aplikacji po wykonanych akcjach. Dodatkowo sprawdzano działanie mechanizmu przewijania z dynamicznym doładowywaniem danych (*infinite scroll*) w warunkach zbliżonych do rzeczywistego użycia aplikacji.

Spec	Tests	Passing	Failing	Pending	Skipped
✓ account.cy.js	00:04	2	2	-	-
✓ user-dashboard/add-spot.cy.js	00:10	5	5	-	-
✓ user-dashboard/comments.cy.js	00:04	3	3	-	-
✓ user-dashboard/favorite-spots.cy.js	00:04	4	4	-	-
✓ user-dashboard/movies.cy.js	00:04	4	4	-	-
✓ user-dashboard/photos.cy.js	00:04	4	4	-	-
✓ user-dashboard/profile.cy.js	00:06	7	7	-	-
✓ user-dashboard/settings.cy.js	00:12	7	7	-	-
✓ user-dashboard/social.cy.js	00:03	4	4	-	-
✓ All specs passed!	00:55	40	40	-	-

Rysunek 8.7: Podsumowanie uruchomienia testów end-to-end (E2E) w Cypress

8.4 Scenariusze testów end-to-end (E2E)

Poniżej przedstawiono scenariusze testowe zrealizowane w ramach testów end-to-end (E2E). W opisach rozróżniono przypadki z użyciem danych mockowanych (poprzez przechwytywanie żądań HTTP) oraz przypadki wykonywane na rzeczywistym backendzie.

8.4.1 Account (logowanie i rejestracja)

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ACC-01
Cel:	Potwierdzenie możliwości zalogowania użytkownika.
Typ:	Rzeczywisty backend
Warunki wstępne:	Konto istnieje w systemie.
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie strony głównej aplikacji. 2. Przejście do widoku logowania. 3. Wprowadzenie nazwy użytkownika oraz hasła. 4. Wysłanie formularza.
Oczekiwany rezultat:	Zakończenie procesu logowania bez błędów oraz przejście do aplikacji.

Tabela 8.1: Scenariusz E2E: Logowanie użytkownika

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ACC-02
Cel:	Potwierdzenie możliwości założenia konta.
Typ:	Rzeczywisty backend
Warunki wstępne:	brak
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie strony głównej aplikacji. 2. Przejście do widoku logowania, a następnie do widoku rejestracji. 3. Wprowadzenie danych rejestracyjnych (nazwa użytkownika, e-mail, hasło, potwierdzenie hasła). 4. Wysłanie formularza.

Oczekiwany rezultat:	Zakończenie rejestracji bez błędów.
-----------------------------	-------------------------------------

Tabela 8.2: Scenariusz E2E: Rejestracja użytkownika

8.4.2 User dashboard – Add spot

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ADD-01
Cel:	Weryfikacja poprawnej obsługi pustej listy dodanych spotów.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> Otwarcie widoku <i>Add spot</i>. Oczekивание na odpowiedź endpointu z listą spotów (pusta lista).
Oczekiwany rezultat:	Wyświetlenie komunikatu o braku dodanych spotów.

Tabela 8.3: Scenariusz E2E: Wyświetlenie stanu pustego dla dodanych spotów

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ADD-02
Cel:	Weryfikacja działania mechanizmu <i>infinite scroll</i> .
Typ:	Mockowane API

Warunki wstępne:	Użytkownik jest zalogowany i znajduje się na widoku listy z włączonym nieskończonym przewijaniem; w źródle danych istnieją kolejne elementy do załadowania.
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku <i>Add spot</i>. 2. Weryfikacja wyświetlenia pierwszej porcji danych. 3. Doprowadzenie do uruchomienia mechanizmu <i>infinite scroll</i> i pobranie kolejnej strony.
Oczekiwany rezultat:	Dołączenie kolejnych elementów do listy oraz ich poprawna prezentacja.

Tabela 8.4: Scenariusz E2E: Ładowanie kolejnych elementów listy przy przewijaniu (*infinite scroll*)

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ADD-03
Cel:	Weryfikacja dostępności modala dodawania spota w trybie desktop.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Ustawienie rozdzielczości okna na tryb desktop. 2. Otwarcie widoku <i>Add spot</i>. 3. Wybranie przycisku <i>Add spot</i>.
Oczekiwany rezultat:	Wyświetlenie modala dodawania spota wraz z sekcjami (<i>Basic Information, Upload Media</i>).

Tabela 8.5: Scenariusz E2E: Otwarcie modala dodawania spota na widoku desktop

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ADD-04
Cel:	Weryfikacja blokady dodawania spota dla małych rozdzielczości.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Ustawienie rozdzielczości okna na tryb mobilny. 2. Otwarcie widoku <i>Add spot</i>. 3. Próba otwarcia modala dodawania spota.
Oczekiwany rezultat:	Wyświetlenie komunikatu o wymaganym większym ekranie oraz brak wyświetlenia formularza dodawania.

Tabela 8.6: Scenariusz E2E: Zablokowanie dodawania spota na małym ekranie

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-ADD-05
Cel:	Weryfikacja pełnego procesu dodania spota (UI → backend).
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i możliwość zalogowania.

Kroki:	<ol style="list-style-type: none"> 1. Wykonanie logowania użytkownika. 2. Przejście do widoku <i>Add spot</i>. 3. Otwarcie modala dodawania spota. 4. Uzupełnienie danych podstawowych. 5. Dodanie pliku graficznego w sekcji multimediiów. 6. Wyznaczenie wielokąta na mapie oraz zakończenie rysowania. 7. Zatwierdzenie dodania spota.
Oczekiwany rezultat:	Zapis spota, komunikat o powodzeniu oraz obecność nowo dodanego spota na liście.

Tabela 8.7: Scenariusz E2E: Dodanie nowego spota z użyciem rzeczywistego backendu

8.4.3 User dashboard – Comments

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-COM-01
Cel:	Weryfikacja obsługi pustej listy komentarzy.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku komentarzy. 2. Oczekiwanie na odpowiedź z pustą listą komentarzy.
Oczekiwany rezultat:	Wyświetlenie komunikatu o braku komentarzy oraz brak elementów listy.

Tabela 8.8: Scenariusz E2E: Wyświetlenie stanu pustego dla komentarzy

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-COM-02
Cel:	Weryfikacja ponownego pobrania i prezentacji danych po zmianie sortowania.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku komentarzy. 2. Weryfikacja początkowego układu listy. 3. Zmiana sortowania. 4. Oczekiwanie na ponowne pobranie danych.
Oczekiwany rezultat:	Aktualizacja listy zgodnie z wybranym sortowaniem.

Tabela 8.9: Scenariusz E2E: Sortowanie komentarzy po zmianie typu sortowania

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-COM-03
Cel:	Weryfikacja poprawnego pobrania danych komentarzy z backendu po logowaniu.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i może się na nie zalogować.
Kroki:	<ol style="list-style-type: none"> 1. Wykonanie logowania użytkownika. 2. Przejście do widoku komentarzy. 3. Weryfikacja poprawnej odpowiedzi API (np. struktura <code>items</code> i <code>hasNext</code>).

Oczekiwany rezultat:	Poprawne załadowanie widoku komentarzy.
-----------------------------	---

Tabela 8.10: Scenariusz E2E: Załadowanie widoku komentarzy po Rzeczywistym logowaniu

8.4.4 User dashboard – Favorite spots

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-FAV-01
Cel:	Weryfikacja obsługi pustej listy ulubionych spotów.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku ulubionych spotów. 2. Pobranie pustej listy.
Oczekiwany rezultat:	Komunikat o braku spotów w liście.

Tabela 8.11: Scenariusz E2E: Wyświetlenie stanu pustego dla list ulubionych spotów

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-FAV-02
Cel:	Weryfikacja przełączania zakładek i pobrania danych dla wybranego typu listy.
Typ:	Mockowane API

Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku ulubionych spotów. 2. Weryfikacja elementów w domyślnej liście. 3. Przełączenie zakładki listy (na <i>Favorites</i>).
Oczekiwany rezultat:	Pobranie i wyświetlenie danych odpowiadających wybranemu typowi listy.

Tabela 8.12: Scenariusz E2E: Zmiana typu listy

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-FAV-03
Cel:	Weryfikacja infinite scroll w liście ulubionych spotów.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Przewinięcie listy do dołu w celu pobrania kolejnej strony.
Oczekiwany rezultat:	Zwiększenie liczby elementów listy o kolejne pozycje.

Tabela 8.13: Scenariusz E2E: Dociąganie kolejnej strony przy przewijaniu

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-FAV-04

Cel:	Weryfikacja pobrania danych ulubionych spotów z backendu po logowaniu.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i może się na nie zalogować.
Kroki:	<ol style="list-style-type: none"> Logowanie użytkownika. Przejście do widoku ulubionych spotów. Weryfikacja poprawnej odpowiedzi API.
Oczekiwany rezultat:	Poprawne wyświetlenie widoku oraz elementów (jeśli istnieją).

Tabela 8.14: Scenariusz E2E: Załadowanie danych z rzeczywistego backendu po logowaniu

8.4.5 User dashboard – Movies

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-MOV-01
Cel:	Weryfikacja obsługi pustej listy filmów.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> Otwarcie widoku filmów. Pobranie pustej listy.
Oczekiwany rezultat:	Brak elementów listy oraz poprawne wyświetlenie nagłówka widoku.

Tabela 8.15: Scenariusz E2E: Stan pusty dla filmów

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-MOV-02
Cel:	Weryfikacja odświeżenia listy po zmianie sortowania.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Przełączenie sortowania. 2. Oczekiwanie na ponowne pobranie danych.
Oczekiwany rezultat:	Odświeżenie listy zgodnie z wybranym sortowaniem.

Tabela 8.16: Scenariusz E2E: Zmiana sortowania filmów

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-MOV-03
Cel:	Weryfikacja dociągania kolejnych stron danych.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Przewinięcie listy do dołu w celu pobrania kolejnej strony.
Oczekiwany rezultat:	Dołączenie nowych elementów do listy.

Tabela 8.17: Scenariusz E2E: *Infinite scroll* w widoku filmów

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-MOV-04
Cel:	Weryfikacja pobrania filmów z backendu po logowaniu.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i może się na nie zalogować.
Kroki:	<ol style="list-style-type: none"> 1. Logowanie użytkownika. 2. Przejście do widoku filmów. 3. Weryfikacja poprawnej odpowiedzi API.
Oczekiwany rezultat:	Poprawne wyświetlenie widoku filmów.

Tabela 8.18: Scenariusz E2E: Załadowanie filmów po logowaniu na Rzeczywistym backendzie

8.4.6 User dashboard – Photos

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PHO-01
Cel:	Weryfikacja obsługi pustej listy zdjęć.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku zdjęć. 2. Pobranie pustej listy.
Oczekiwany rezultat:	Brak elementów listy zdjęć.

Tabela 8.19: Scenariusz E2E: Stan pusty dla zdjęć

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PHO-02
Cel:	Weryfikacja odświeżenia listy po zmianie sortowania.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Przełączenie sortowania. 2. Oczekiwanie na ponowne pobranie danych.
Oczekiwany rezultat:	Aktualizacja listy zdjęć.

Tabela 8.20: Scenariusz E2E: Zmiana sortowania zdjęć

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PHO-03
Cel:	Weryfikacja dociągania kolejnych stron danych w liście zdjęć.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Przewinięcie listy w celu pobrania kolejnej strony.
Oczekiwany rezultat:	Zwiększenie liczby elementów na liście zdjęć.

Tabela 8.21: Scenariusz E2E: *Infinite scroll* w widoku zdjęć

KARTA SCENARIUSZA E2E	
-----------------------	--

Identyfikator:	E2E-PHO-04
Cel:	Weryfikacja pobrania zdjęć z backendu po logowaniu.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i może się na nie zalogować.
Kroki:	<ol style="list-style-type: none"> 1. Logowanie użytkownika. 2. Przejście do widoku zdjęć. 3. Weryfikacja obecności danych.
Oczekiwany rezultat:	Poprawne wyświetlenie widoku oraz zdjęć użytkownika.

Tabela 8.22: Scenariusz E2E: Widok zdjęć po logowaniu na Rzeczywistym backendzie

8.4.7 User dashboard – Profile

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-01
Cel:	Weryfikacja poprawnej prezentacji profilu użytkownika.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie profilu własnego. 2. Weryfikacja statystyk oraz wyświetlenia najpopularniejszych zdjęć.
Oczekiwany rezultat:	Obecność zdjęcia profilowego, statystyk oraz sekcji najpopularniejszych zdjęć.

Tabela 8.23: Scenariusz E2E: Wyświetlenie profilu użytkownika (statystyki i popularne zdjęcia)

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-02
Cel:	Weryfikacja obsługi pustej listy zdjęć w profilu.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	1. Otwarcie profilu własnego z pustą listą zdjęć.
Oczekiwany rezultat:	Wyświetlenie komunikatu o braku dodanych zdjęć.

Tabela 8.24: Scenariusz E2E: Komunikat o braku zdjęć w profilu

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-03
Cel:	Weryfikacja nawigacji do widoku znajomych.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	1. Kliknięcie statystyki <i>Friends</i> . 2. Oczekiwanie na załadowanie widoku znajomych.
Oczekiwany rezultat:	Przejście do <i>/account/friends</i> .

Tabela 8.25: Scenariusz E2E: Nawigacja do listy znajomych z poziomu profilu

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-04
Cel:	Weryfikacja nawigacji do widoku zdjęć.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Kliknięcie statystyki <i>Photos</i>. 2. Oczekiwanie na załadowanie widoku zdjęć.
Oczekiwany rezultat:	Przejście do <i>/account/photos</i> .

Tabela 8.26: Scenariusz E2E: Nawigacja do zdjęć z poziomu profilu

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-05
Cel:	Weryfikacja widoczności akcji społecznościowych na profilu innego użytkownika.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie profilu innego użytkownika. 2. Weryfikacja dostępnych akcji.
Oczekiwany rezultat:	Widoczność przycisków <i>follow</i> oraz <i>add to friends</i> .

Tabela 8.27: Scenariusz E2E: Wyświetlenie profilu innego użytkownika (akcje *follow/friends*)

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-06
Cel:	Weryfikacja poprawnych żądań HTTP dla akcji follow/friends.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Kliknięcie <i>follow</i>. 2. Kliknięcie <i>add to friends</i>. 3. Weryfikacja żądań HTTP.
Oczekiwany rezultat:	Wysłanie żądań PATCH z poprawnymi parametrami.

Tabela 8.28: Scenariusz E2E: Wysłanie żądania follow oraz zaproszenia do znajomych

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-PRO-07
Cel:	Weryfikacja poprawnego wyświetlenia profilu po logowaniu.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i może się na nie zalogować.
Kroki:	<ol style="list-style-type: none"> 1. Logowanie. 2. Przejście do profilu własnego.
Oczekiwany rezultat:	Poprawne wyświetlenie profilu oraz elementów interfejsu.

Tabela 8.29: Scenariusz E2E: Załadowanie profilu własnego po logowaniu na Rzeczywistym backendzie

8.4.8 User dashboard – Settings

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-01
Cel:	Weryfikacja obecności danych konta i możliwości edycji nazwy użytkownika.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie ustawień. 2. Weryfikacja pól. 3. Wybranie opcji <i>Edit</i> dla nazwy użytkownika.
Oczekiwany rezultat:	Wyświetlenie formularza zmiany nazwy użytkownika.

Tabela 8.30: Scenariusz E2E: Wyświetlenie danych konta i otwarcie formularza zmiany nazwy użytkownika

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-02
Cel:	Weryfikacja wysłania poprawnego żądania zmiany nazwy użytkownika.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Wprowadzenie nowej nazwy. 2. Zapisanie zmian.

Oczekiwany rezultat:	Wysłanie żądania PATCH oraz komunikat o powodzeniu.
-----------------------------	---

Tabela 8.31: Scenariusz E2E: Zmiana nazwy użytkownika

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-03
Cel:	Weryfikacja wysłania poprawnego żądania zmiany e-mail.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Wprowadzenie nowego adresu e-mail. 2. Zapisanie zmian.
Oczekiwany rezultat:	Wysłanie żądania PATCH oraz komunikat o powodzeniu.

Tabela 8.32: Scenariusz E2E: Zmiana adresu e-mail

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-04
Cel:	Weryfikacja procesu zmiany hasła.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Wprowadzenie starego i nowego hasła (z potwierdzeniem). 2. Zapis zmian.

Oczekiwany rezultat:	Wysłanie żądania PATCH oraz komunikat o powodzeniu.
-----------------------------	---

Tabela 8.33: Scenariusz E2E: Zmiana hasła

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-05
Cel:	Weryfikacja ograniczeń edycji danych dla kont OAuth.
Typ:	Mockowane API
Warunki wstępne:	Użytkownik jest zalogowany kontem z providerem OAuth.
Kroki:	1. Otwarcie ustawień dla użytkownika z providerem OAuth.
Oczekiwany rezultat:	Brak formularzy edycji danych oraz wyświetlenie informacji o providerze.

Tabela 8.34: Scenariusz E2E: Ograniczone ustawienia dla konta OAuth

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-06
Cel:	Weryfikacja poprawnego wyświetlenia ustawień po logowaniu.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto i może się na nie zalogować.
Kroki:	1. Logowanie. 2. Przejście do ustawień.

Oczekiwany rezultat:	Poprawne wyświetlenie danych konta.
-----------------------------	-------------------------------------

Tabela 8.35: Scenariusz E2E: Wyświetlenie ustawień po logowaniu na Rzeczywistym backendzie

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SET-07
Cel:	Weryfikacja zmiany hasła i ponownego logowania.
Typ:	Rzeczywisty backend
Warunki wstępne:	Użytkownik posiada konto lokalne (nie OAuth).
Kroki:	<ol style="list-style-type: none"> Wykonanie logowania użytkownika. Przejście do ustawień i zmiana hasła. Wyczyszczenie danych sesyjnych po stronie przeglądarki. Ponowne logowanie z nowym hasłem.
Oczekiwany rezultat:	Możliwość zalogowania się nowym hasłem oraz poprawne załadowanie widoku ustawień.

Tabela 8.36: Scenariusz E2E: Zmiana hasła na Rzeczywistym backendzie oraz ponowne logowanie

8.4.9 User dashboard – Social (friends/followed/followers)

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SOC-01
Cel:	Weryfikacja obsługi pustej listy znajomych.

Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku listy społeczności. 2. Pobranie pustej listy znajomych.
Oczekiwany rezultat:	Wyświetlenie komunikatu o braku znajomych.

Tabela 8.37: Scenariusz E2E: Stan pusty listy znajomych

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SOC-02
Cel:	Weryfikacja przełączania zakładek i poprawnego wyświetlania danych.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	<ol style="list-style-type: none"> 1. Otwarcie widoku społeczności. 2. Weryfikacja listy znajomych. 3. Przełączenie zakładki na <i>followed</i>, a następnie <i>followers</i>.
Oczekiwany rezultat:	Wyświetlenie list odpowiednich dla wybranych zakładek.

Tabela 8.38: Scenariusz E2E: Przełączanie zakładek friends/followed/followers

KARTA SCENARIUSZA E2E	
-----------------------	--

Identyfikator:	E2E-SOC-03
Cel:	Weryfikacja dociągania kolejnych stron danych w liście społeczności.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	1. Przewinięcie listy do dołu w celu pobrania kolejnej strony danych. 2. Akceptacja kolejnego zaproszenia.
Oczekiwany rezultat:	Dołączenie kolejnych kart użytkowników do listy.

Tabela 8.39: Scenariusz E2E: *Infinite scroll* na liście znajomych

KARTA SCENARIUSZA E2E	
Identyfikator:	E2E-SOC-04
Cel:	Weryfikacja akceptowania zaproszeń i wysyłania poprawnych żądań.
Typ:	Mockowane API
Warunki wstępne:	Zasymulowanie zalogowania poprzez wpisy w <i>localStorage</i> .
Kroki:	1. Otwarcie widoku społeczności. 2. Przejście do widoku zaproszeń (<i>See friend invites</i>). 3. Akceptacja wybranego zaproszenia.
Oczekiwany rezultat:	Wysłanie żądania zmiany statusu zaproszenia oraz brak błędów w interfejsie.

Tabela 8.40: Scenariusz E2E: Obsługa zaproszeń do znajomych

8.5 Wyniki testów i wnioski

Przeprowadzone testy jednostkowe, integracyjne oraz end-to-end potwierdziły poprawność działania kluczowych funkcjonalności aplikacji zarówno po stronie **frontendu**, jak i **backendu**. Uzyskane wyniki wskazują, że zaimplementowana logika biznesowa, warstwa prezentacji oraz komunikacja z **API** działają zgodnie z założeniami projektowymi.

Testy jednostkowe pozwoliły zweryfikować poprawność działania pojedynczych komponentów i usług, w tym obsługę przypadków brzegowych oraz walidację danych wejściowych. Testy integracyjne umożliwiły potwierdzenie współpracy większych fragmentów systemu, w szczególności przepływu danych pomiędzy komponentami oraz reakcji aplikacji na działania użytkownika (zmianę stanu po interakcji). Z kolei testy E2E, uruchamiane w środowisku zbliżonym do rzeczywistego, potwierdziły poprawną realizację pełnych scenariuszy użytkownika, obejmujących logowanie, edycję ustawień konta, przeglądanie treści oraz dynamiczne doładowywanie danych (*infinite scroll*).

Na podstawie wyników testów sformułowano następujące wnioski:

- Zapewniono wysoką stabilność kluczowych modułów aplikacji dzięki szerskiemu pokryciu testami jednostkowymi.
- Mechanizmy odpowiedzialne za paginację i doładowywanie danych (*infinite scroll*) działają poprawnie w testowanych widokach oraz nie powodują błędów w interfejsie.
- Integracja frontendu z backendem została potwierdzona zarówno w testach integracyjnych, jak i E2E, co minimalizuje ryzyko regresji w przypadku dalszego rozwoju aplikacji.
- Zastosowanie mockowania zapytań w testach ułatwiło deterministyczne odtwarzanie scenariuszy oraz testowanie stanów trudnych do uzyskania w środowisku rzeczywistym (puste listy, konkretne warianty sortowania).

Rozdział 9

Prezentacja systemu

W niniejszym rozdziale przedstawiono finalny wygląd aplikacji. Wcześniej opracowany projekt interfejsu zaprezentowany w rozdziale [5.3 Architektura interfejsu użytkownika](#) pełnił funkcję punktu wyjścia oraz wsparcia w procesie implementacji. Ostateczny kształt interfejsu został wypracowany iteracyjnie, na podstawie konsultacji między członkami zespołu oraz z promotorem, a także wniosków wynikających z testów manualnych. Z tego względu w wielu miejscach warstwa wizualna różni się od pierwotnych makiet.

9.1 Strona główna

9.2 Strona mapy

9.3 Strona czatu

9.3.1 Widok bazowy modułu czatu

Ekran 1. Widok główny rozmów

Rysunek X przedstawia główny widok modułu czatu. Interfejs zaprojektowano z wyraźnym podziałem na dwie strefy: panel listy rozmów po lewej stronie oraz obszar aktywnej konwersacji w części centralnej.

Bezpośrednio na prawo od paska nawigacyjnego znajduje się panel prezentujący listę dostępnych konwersacji. Każdy element listy zawiera awatar, nazwę

rozmowy, fragment ostatniej wiadomości oraz informację o czasie wysłania. Aktualnie wybrana rozmowa jest wyróżniona wizualnie, co ułatwia orientację w module i jednoznacznie wskazuje konwersację, której treść jest prezentowana w części centralnej.

Centralną część ekranu zajmuje okno aktywnej rozmowy. W górnym pasku widoczna jest nazwa konwersacji oraz zestaw akcji kontekstowych związanych z bieżącym czatem. W przypadku rozmowy prywatnej nazwę konwersacji stanowi nazwa użytkownika, z którym prowadzona jest wymiana wiadomości. Kliknięcie nazwy przenosi użytkownika do profilu tej osoby w module społecznościowym. W przypadku czatu grupowego nazwa rozmowy oraz jej awatar mogą zostać zmodyfikowane, a wejście w tryb edycji jest dostępne z poziomu nagłówka konwersacji. Dodatkowo, dla czatu prywatnego udostępniono akcję umożliwiającą utworzenie czatu grupowego z udziałem aktualnego rozmówcy.

Poniżej nagłówka znajduje się przewijalna historia wiadomości ułożona chronologicznie. Każda wiadomość prezentuje awatar nadawcy, nazwę użytkownika, znacznik czasu oraz treść, co pozwala jednoznacznie identyfikować autora wypowiedzi i kontekst czasowy rozmowy.

W dolnej części widoku umieszczono pasek wprowadzania wiadomości, składający się z pola tekstowego oraz przycisków akcji (dodawanie załączników, wysyłanie GIF-ów oraz emoji) i przycisku wysyłania. Taki układ wspiera typowy scenariusz użycia modułu: wybór rozmowy z listy po lewej stronie, przegląd historii w części centralnej oraz tworzenie i wysyłanie kolejnych wiadomości.

Ekran 2. Edycja czatu grupowego

Rysunek Y przedstawia okno edycji czatu grupowego, wyświetlane po kliknięciu w obszar nagłówka konwersacji (kafelek z awatarem i nazwą czatu). W ramach tego widoku użytkownik może zmienić awatar oraz nazwę czatu grupowego. Rozwiązanie to umożliwia dostosowanie identyfikacji rozmowy do potrzeb jej uczestników, bez konieczności opuszczania modułu czatu.

Ekran 3. Dodawanie załączników do wiadomości

Rysunek Z przedstawia proces dołączania plików do wiadomości. Po użyciu przycisku dodawania załączników w pasku wprowadzania treści wyświetlane jest sys-

temowe okno wyboru pliku, pozwalające wskazać zasób z lokalnego dysku. Po zatwierdzeniu wyboru plik zostaje dołączony do aktualnie tworzonej wiadomości i może zostać wysłany do pozostałych uczestników rozmowy jako załącznik.

Ekran 4. Panel wyszukiwania i wysyłania GIF-ów

Rysunek W prezentuje panel wyboru GIF-ów dostępny z poziomu paska edycji wiadomości. Panel otwierany jest w obrębie widoku konwersacji i nie wymaga opuszczania czatu. Użytkownik może przeszukiwać zasoby poprzez pole wyszukiwania, a wyniki prezentowane są w formie siatki miniatur. Wybranie elementu powoduje dodanie GIF-a do wiadomości i umożliwia jego wysłanie w ramach rozmowy.

Ekran 5. Panel wyboru emoji

Rysunek V przedstawia panel wyboru emoji, uruchamiany analogicznie jak panel GIF-ów. Emoji wybierane są z poziomu *picker-a*, który udostępnia kategorie predefiniowane (np. według typu emotikon) oraz wyszukiwanie tekstowe po frazie. Zaznaczenie emoji wstawia je do pola tekstowego wiadomości, dzięki czemu może ono zostać wykorzystane samodzielnie lub jako uzupełnienie treści.

Ekran 6. Tworzenie czatu grupowego z rozmowy prywatnej

Rysunek U przedstawia okno dialogowe tworzenia czatu grupowego uruchamiane z poziomu rozmowy prywatnej (przycisk akcji w nagłówku konwersacji). W oknie dostępne jest pole wyszukiwania użytkowników oraz lista wybranych osób prezentowana w formie etykiet. Domyślnie uwzględniony jest aktualny rozmówca oraz użytkownik tworzący czat; możliwe jest wskazanie dodatkowych uczestników (do zdefiniowanego limitu). Jeżeli nie zostaną dodane kolejne osoby, tworzony jest dwuosobowy czat grupowy, który zachowuje cechy rozmowy grupowej (m.in. możliwość późniejszej rozbudowy składu oraz edycji danych czatu).

Ekran 7. Widok czatu grupowego

Rysunek T przedstawia widok aktywnej konwersacji grupowej po jej utworzeniu. Interfejs zachowuje układ znany z rozmów prywatnych (lista czatów po lewej stronie oraz historia wiadomości w części centralnej), natomiast w nagłówku prezentowana jest nazwa czatu grupowego (lub skrócona lista uczestników, zależnie od konfiguracji). W prawym górnym rogu dostępne są dwa przyciski: pierwszy otwiera

listę uczestników czatu, natomiast drugi umożliwia dodawanie kolejnych osób do grupy.

Ekran 8. Lista uczestników czatu grupowego

Rysunek S przedstawia panel boczny z listą uczestników, wyświetlany po kliknięciu przycisku podglądu członków grupy. Panel pojawia się po prawej stronie widoku konwersacji i zawiera nagłówek z liczbą uczestników oraz listę profili (wraz z awatarami i nazwami użytkowników). Rozwiążanie to umożliwia szybkie zweryfikowanie składu czatu bez przerywania ciągłości rozmowy.

Ekran 9. Dodawanie nowych uczestników do czatu grupowego

Rysunek R przedstawia okno dialogowe dodawania użytkowników do istniejącej konwersacji grupowej. W oknie dostępne jest pole wyszukiwania oraz mechanizm wyboru użytkowników wraz z licznikiem określającym maksymalną liczbę osób możliwych do dodania w ramach jednej operacji. Po zatwierdzeniu akcją kontekstową nowi członkowie zostają dopisani do czatu i są widoczni na liście uczestników.

9.4 Strona forum

9.5 Panel logowania

9.6 Panel konta użytkownika

Rozdział 10

Nakład pracy

10.1 Ogólny nakład pracy

10.2 Indywidualne nakłady pracy

10.2.1 Adam Langmesser

10.2.2 Mateusz Redosz

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

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

- Propozycja wyglądu

3. Backend i Frontend

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

- Strona główna z prostymi filtrami
- Strona główna z zaawansowanymi filtrami
- [Sidebar](#)
- Strona profilu
- Ustawienia
- Listy spotów
- Lista zdjęć
- Lista filmów
- Lista znajomych
- Dodanie spotów
- Lista komentarzy
- Strona główna profilu
- Listy
- Poprawa [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 scrolllem
- Lista zdjęć innego usera
- Walidacja i responsywność w dodaniu spotów
- Dodanie sortowania i filtrów na zaawansowanej stronie
- Zmiana na infinity scrola

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

4. CI/CD

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

5. Praca dyplomowa

- Uzupełnienie informacji o zespole i podział na rozdziały

10.2.3 Stanisław Oziemczuk

10.2.4 Kacper Badek

Rozdział 11

Podsumowanie

W niniejszym rozdziale podsumowano rezultaty uzyskane podczas realizacji projektu, przedstawiono napotkane trudności oraz zarysowano dalsze plany dotyczące rozwijania rozwiązania.

Realizacja projektu umożliwiła zdobycie praktycznego doświadczenia w wytwarzaniu aplikacji internetowej o rozbudowanej funkcjonalności, obejmującej zarówno warstwę **frontendową**, jak i **backendową**. Pozyskana wiedza z zakresu projektowania, implementacji, testowania oraz integracji komponentów może stanowić wartościową podstawę w dalszym rozwoju zawodowym.

11.1 Osiągnięte rezultaty

W ramach projektu zrealizowano wszystkie wymagania główne, większość wymagań typu *should* oraz znaczną część wymagań typu *could*. Powstała pełnowymiarowa aplikacja internetowa, w dużym stopniu przygotowana do wdrożenia.

Zaimplementowano kluczowe moduły systemu, w tym: wyszukiwarkę **spotów** wraz z mapą wspierającą ich przeglądanie i lokalizowanie, czat umożliwiający komunikację między użytkownikami, forum do publikowania i komentowania treści oraz panel użytkownika wspierający zarządzanie kontem.

Opracowano również model **bazy danych** przechowujący informacje niezbędne do działania systemu. W celu ułatwienia uruchamiania środowiska zastosowano konteneryzację, uruchamiając dwie instancje **baz danych** w **kontenerach Docker**, co usprawniło konfigurację oraz powtarzalność środowiska.

11.2 Napotkane wyzwania

Jednym z pierwszych wyzwań był ograniczony poziom wcześniejszego doświadczenia w realizacji projektów o porównywalnej skali i złożoności, co wymagało dopracowania sposobu planowania prac oraz ujednolicenia podejścia do implementacji.

Istotnym obszarem trudności okazała się integracja zabezpieczeń po stronie [backendu](#) z wykorzystaniem [biblioteki](#) Spring Security, z którą wcześniej nie pracowano. W konsekwencji pojawiały się nieporozumienia interpretacyjne oraz konieczność wprowadzania poprawek w konfiguracji.

Kolejnym problemem była implementacja logowania z użyciem zewnętrznych dostawców (Google oraz GitHub). Wystąpiły trudności związane z konfiguracją [OAuth](#), przekierowaniami oraz dopasowaniem integracji do przyjętej architektury, przy jednoczesnym ograniczonym dostępem do materiałów opisujących analogiczny przypadek.

Dodatkowym wyzwaniem było zastosowanie w projekcie wielu nowych [bibliotek](#) zarówno po stronie [frontendowej](#), jak i [backendowej](#). Wymagało to analizy dokumentacji, weryfikacji kompatybilności wersji oraz doboru właściwych sposobów integracji.

Dodatkowym wyzwaniem była nauka języka [LATEX](#), który wykorzystano do przygotowania dokumentacji projektu, co wiązało się z koniecznością poznania składni oraz sposobów formatowania treści technicznej.

Trudności pojawiły się również podczas wdrażania usługi mapowej wykorzystywanej w aplikacji. Było to pierwsze uruchomienie komponentu na zewnętrznym serwerze, co ujawniło problemy konfiguracyjne.

11.3 Plany na przyszłość

Projekt w większości zrealizowano zgodnie z założeniami, jednak ze względów organizacyjnych i osobistych nie przewiduje się wdrożenia produkcyjnego ani dalszej rozbudowy w obecnym kształcie.

Rozdział 12

Słownik pojęć i skrótów

ACK

Skrót od *Acknowledgement*; komunikat potwierdzający odbiór lub przetworzenie wiadomości. W [STOMP](#) może służyć do potwierdzania wiadomości po stronie klienta.. [430](#), [434–436](#)

Adnotacja

To sposób dodania metadanych do kodu Java, nie zmieniają bezpośrednio działania programu, ale są wykorzystywane przez kompilator i [frameworki](#). Zaczynają się symbolem @. [192](#), [194](#), [264](#)

akcja Redux

Obiekt opisujący zdarzenie/operację zmiany stanu w Redux, obsługiwany przez reducery (np. `chatActions.setLastMessage(...)`). [449](#)

API

(ang. *application programming interface*); zbiór reguł i operacji do komunikacji z oprogramowaniem.. [2](#), [28](#), [30](#), [146](#), [147](#), [150](#), [151](#), [345](#), [361](#), [450](#), [452](#), [480](#), [496](#)

Azure Blob Storage

Usługa magazynu obiektowego w chmurze Microsoft Azure do przechowywania nieuszkodzonych danych (*blobs*) takich jak obrazy, wideo i pliki. Udostępnia kontenery, warstwy dostępu, wersjonowanie oraz tokeny SAS; często używana do hostowania multimediów w aplikacjach webowych.. [186](#)

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](#), [18](#), [26](#), [153](#), [156](#), [157](#), [159](#), [183](#), [189](#), [196](#), [197](#), [200](#), [209](#), [210](#), [259](#), [263](#), [275](#), [277](#), [293](#), [299](#), [300](#), [308](#), [309](#), [311](#), [314](#), [315](#), [319](#), [322](#), [324](#), [327](#), [328](#), [335](#), [342](#), [344](#), [345](#), [348](#), [350](#), [362](#), [363](#), [365](#), [366](#), [368](#), [376](#), [379](#), [382](#), [384](#), [387](#), [419](#), [420](#), [425](#), [430](#), [431](#), [434](#), [443](#), [450](#), [452](#), [454](#), [456](#), [480](#), [486](#), [489](#), [490](#), [495](#), [507](#)

Backlog

Lista zadań, które należy wykonać w ramach projektu, używana w metodykach zwinnych.. [27](#)

Baza danych

Zbiór uporządkowanych danych przechowywanych w sposób umożliwiający ich łatwe wyszukiwanie, modyfikowanie i analizowanie. W aplikacjach najczęściej wykorzystywane są relacyjne lub nierelacyjne bazy danych. [2](#), [156](#), [157](#), [159](#), [259](#), [262](#), [263](#), [434](#), [436](#), [489](#)

Bean

To obiekt w [frameworku Spring](#), który jest tworzony i zarządzany przez kontener Spring IoC. [192](#)

Biblioteka

Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. [21](#), [156](#), [183](#), [189–191](#), [259](#), [270](#), [272](#), [275](#), [277](#), [281](#), [286](#), [290](#), [293](#), [298–302](#), [304](#), [306](#), [308–311](#), [314](#), [315](#), [322](#), [328](#), [335](#), [342](#), [344](#), [355](#), [362](#), [365](#), [368](#), [372](#), [490](#)

BPMN

(ang. *Business Process Model and Notation*); standardowa notacja graficzna, która umożliwia szczegółowe przedstawienie i dokumentowanie procesów biznesowych..
[29](#)

Cache

Mechanizm przechowywania danych w celu przyspieszenia ich ponownego odczytu.
[25](#), [27](#), [156](#), [157](#), [184](#), [209](#), [262–265](#), [421](#), [423](#), [425](#), [509](#)

callback

Funkcja przekazywana jako parametr, wywoływaną automatycznie przy wystąpieniu zdarzenia (np. gdy nadaje się wiadomość z WebSocket).. [437](#), [442](#), [445](#), [449](#)

CI/CD

Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. [28](#), [184](#), [419](#), [450](#), [454](#), [456](#), [485](#), [488](#)

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. [276](#)

cleanup

„Sprzątanie” zasobów: usuwanie subskrypcji, rozłączanie połączeń i czyszczenie struktur pomocniczych, aby nie pozostawiać aktywnych uchwytów. [443](#), [447](#)

Convention Over Configuration

Zasada programowania polegająca na przyjmowaniu domyślnych, bazowych reguł, zamiast ręcznego implementowania konfiguracji. [18](#)

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.. [281](#)

cykl życia komponentu

Etapy działania komponentu (np. montowanie, aktualizacja, odmontowanie) determinujące momenty inicjalizacji i zwalniania zasobów. [437](#), [445](#)

Debounce

Technika programistyczna polegająca na ograniczaniu liczby wywołań funkcji po przez jej uruchomienie dopiero po upływie zadanego czasu od ostatniego wywołania.. [300](#), [361](#), [365](#)

deserializacja

Proces odtworzenia obiektu z formatu przenośnego (np. JSON) po stronie odbiorcy. [449](#)

Design

Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). [485](#)

destynacja

Adres logiczny (ścieżka) w komunikacji **STOMP**, na który wysyła się wiadomości lub który się **subskrybuje** (np. `/app/...` albo `/subscribe/...`). [430](#), [432–434](#), [436](#), [437](#), [442](#), [443](#), [445](#), [448](#), [509](#), [510](#)

Disciplined Agile Delivery - Lean Life Cycle

Disciplined Agile Delivery w wariantie 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 ciągłe doprecyzowywanie wymagań i backlogu, oparte na regularnym feedbacku udziałowców. Całość opiera się na praktykach Lean oraz lekkim governance: code review i regularnych przeglądach postępów. . [14](#), [183](#)

dispatch

Funkcja w Redux służąca do wysyłania akcji, które modyfikują stan przechowywany w store. [447](#)

Docker

Platforma do konteneryzacji aplikacji wykorzystująca wirtualizację na poziomie systemu operacyjnego. Umożliwia uruchamianie oprogramowania w lekkich, odizolowanych kontenerach wraz z wszystkimi zależnościami, co ułatwia przenoszenie i powtarzalne odtwarzanie środowisk uruchomieniowych. [184](#), [262](#), [421](#), [500](#)

Docker Compose

Narzędzie do definiowania i uruchamiania aplikacji składających się z wielu kontenerów Docker. Konfiguracja usług (m.in. obrazy, porty, wolumeny i sieci) opisywana jest w pliku YAML (np. `docker-compose.yml`) i umożliwia jednoczesne uruchamianie powiązanych usług (np. [Backend](#), baza danych, usługi pomocnicze) jednym poleceniem. [262](#), [263](#), [421](#), [423](#), [515](#)

Docker Hub

Publiczne repozytorium (rejestr) obrazów kontenerów Docker, umożliwiające przechowywanie, udostępnianie oraz dystrybucję obrazów. Użytkownicy mogą korzystać z oficjalnych obrazów przygotowanych przez społeczność i dostawców oprogramowania lub publikować własne obrazy. [262](#)

DOM

(ang. *Document Object Model*); interfejs reprezentacji stron internetowych jako węzły i obiekty. Dzięki temu skrypty, np. napisane w JavaScript, mogą wchodzić w interakcje ze stroną (np. modyfikować strukturę, treść lub styl). [21](#), [206](#), [365](#), [500](#)

Droniarz

Potoczne określenie osoby, która jest jednocześnie pilotem oraz operatorem drona. Zwykle entuzjasta dronów.. [12](#), [13](#), [521](#)

Droniarz foto/video

Pilot wykorzystujący drony fotograficzne/filmowe do rejestracji materiałów wizualnych (zdjęcia, wideo), zwykle z naciskiem na stabilizację i jakość obrazu.. [29](#)

emoji

Małe graficzne ikonki używane do wyrażania emocji lub pojęć w komunikacji cyfrowej (np. uśmiechnięta buźka, kciuk w górę, symbol serca).. [118](#), [186](#)

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).. [200](#), [209](#), [263](#), [319](#), [322](#), [327](#), [328](#), [344](#), [358](#), [432](#), [433](#), [436](#)

ESLint

Statyczny analizator kodu JavaScript/TypeScript. Umożliwia wykrywanie błędów, niespójności stylu oraz potencjalnych problemów poprzez zestaw reguł, które można dostosować do projektu. [184](#)

fallback

Mechanizm awaryjnego przełączania na alternatywny sposób działania, gdy preferowana metoda jest niedostępna. W kontekście WebSocket/SockJS oznacza automatyczne przejście z WebSocket na techniki oparte o HTTP (np. streaming lub long polling), aby utrzymać komunikację.. [433](#)

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.. [209](#), [267](#)

Framework

Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. [2](#), [18](#), [157](#), [192](#), [209](#), [278](#), [491](#), [492](#), [498](#)

Frontend

Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. [2](#), [18](#), [21](#), [26](#), [156](#), [157](#), [159](#), [183](#), [189](#), [266](#), [267](#), [275](#), [289](#), [343](#), [375](#), [419](#), [427](#), [428](#), [430](#), [431](#), [436](#), [441](#), [443](#), [450](#), [451](#), [454](#), [456](#), [480](#), [486](#), [489](#), [490](#), [499](#)

GIF

Format graficzny *Graphics Interchange Format* obsługujący krótkie, zapętlone animacje. W aplikacjach czatowych wykorzystywany do wysyłania „reakcji” w postaci ruchomych obrazków. [114](#), [115](#), [150](#), [151](#), [186](#), [266](#)

GitHub

Platforma hostingu repozytoriów *Git* w chmurze, oferująca m.in. pull requesty, system zgłoszeń (issues), zarządzanie wersjami oraz integrację z narzędziami CI/CD. [184](#)

GitHub Actions

Platforma CI/CD zintegrowana z portalem GitHub, umożliwiająca automatyzację procesów takich jak budowanie, testowanie i wdrażanie aplikacji w odpowiedzi na zdarzenia w repozytorium.. [419](#), [420](#), [423](#), [450](#), [454](#), [456](#)

handshake

Etap początkowy zestawiania połączenia (np. [WebSocket](#)), w którym [klient](#) i [serwer](#) uzgadniają parametry komunikacji.. [433](#), [436](#), [442](#)

hook

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`).. [196](#), [197](#), [272](#), [273](#), [277](#), [290](#), [293](#), [298–301](#), [305](#), [308–310](#), [314](#), [315](#), [322](#), [324](#), [327](#), [328](#), [335](#), [342](#), [344](#), [345](#), [347](#), [348](#), [350](#), [354](#), [355](#), [361](#), [363](#), [365](#), [366](#), [368](#), [372](#), [376](#), [384](#), [393](#), [409](#), [437](#), [445](#), [500](#)

HTTP

Podstawowy protokół komunikacji w sieci WWW, wykorzystywany m.in. przez [REST API](#). Zwykle działa w modelu „żądanie–odpowiedź”, bez stałego połączenia.. [430](#), [431](#), [503](#)

IDE

(ang. *integrated development environment*); zintegrowane środowisko programistyczne, służące do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. [26](#)

IETF

Organizacja opracowująca i publikująca standardy internetowe, m.in. dokumenty typu [RFC](#).. [429](#), [507](#)

Infinite scroll

Wzorzec interfejsu użytkownika, w którym kolejne porcje treści są automatycznie doładowywane podczas przewijania strony w dół, zamiast być podzielone na odreębne, ręcznie przełączane strony. [88](#), [96](#), [106](#), [109](#), [123](#), [287](#), [299](#), [300](#), [310](#), [335](#), [348](#), [350](#), [359](#), [363](#), [366](#), [454](#), [456](#), [457](#), [480](#)

IoC

Inversion of Control (tłum. *Odwrocenie kontroli*); paradymat programowania, w którym kontrola nad zarządzaniem obiektami i zależnościami przekazywana jest do zewnętrznego kontenera lub [frameworka](#). [492](#)

Job (GitHub Actions)

Jednostka wykonawcza w GitHub Actions, składająca się z sekwencji kroków (*steps*) uruchamianych na tym samym runnerze. Job może zależeć od innych jobów i być

wykonywany równolegle lub sekwencyjnie w ramach workflow.. [421](#), [423](#), [425](#), [427](#), [428](#)

JSON

Format tekstowy do wymiany danych (JavaScript Object Notation), często używany jako reprezentacja komunikatów API i wiadomości w aplikacjach webowych. [443](#), [449](#)

JSX

JavaScript XML; składnia pozwalająca pisać kod wyglądający jak HTML w plikach JavaScript. Umożliwia między innymi wyświetlanie zawartości zmiennych poprzez umieszczenie ich w {}. [499](#)

JVM

(ang. *Java Virtual Machine*); maszyna wirtualna oraz środowisko do wykonywania kodu bajtowego Javy. [18](#)

JWT

Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. [147](#), [276](#), [485](#)

klient

Część systemu (zwykle po stronie [frontendu](#)), która korzysta z usług serwera i prezentuje dane użytkownikowi.. [431–434](#), [436](#), [491](#), [497](#), [501](#), [503](#), [507](#), [512](#)

klient STOMP

Biblioteka/komponent po stronie klienta realizujący protokół STOMP (subskrypcje, publikacja, ramki, nagłówki) nad transportem WebSocket/SockJS. [441](#), [442](#)

Komponent React

Podstawowy blok budujący aplikację React, który zwraca [JSX](#). Cykl życia komponentu ma trzy stany: montowanie (inicjalizacja i renderowanie), aktualizacja

(reagowanie na zmiany), demontowanie (usunięcie komponentu z drzewa DOM), a ich nazwy muszą zaczynać się wielką literą. Powinien być tworzony w sposób umożliwiający jego wielokrotne użycie w projekcie. Komponenty dzielą się na dwa rodzaje:

- **Komponenty Klasowe** — klasy JavaScript dziedziczące po wbudowanej klasie Component. Pozwalają na zarządzanie stanem poprzez obiekt *state*, zawierający się w każdym komponencie klasowym. Zarządzanie cyklem życia komponentu odbywa się za pomocą wbudowanych metod. Po wprowadzeniu komponentów funkcyjnych nie zaleca się stosowania klasowych.
- **Komponenty Funkcyjne** — definiowane jak funkcje JavaScript. **Propsy** są przyjmowane poprzez argumenty funkcji. Zarządzanie stanem oraz cyklem życia komponentu odbywa się za pomocą hook'ów takich jak *useState*.
. 196, 197, 200, 204, 206, 289, 290, 293, 294, 298–302, 304–306, 308–311, 314, 315, 319, 321, 322, 324, 326–328, 332, 334, 335, 341–343

kontekst React'a

Mechanizm w React pozwalający udostępnić wspólny obiekt wielu komponentom bez przekazywania go przez **propsy**. 436, 437, 445, 446

kontener

Lekka, odizolowana jednostka uruchomieniowa, w której umieszczana jest aplikacja wraz z jej zależnościami (bibliotekami, konfiguracją itp.). Kontenery współdzielą jądro systemu operacyjnego, ale posiadają własną przestrzeń procesów, systemu plików i sieci, co zapewnia powtarzalne i przenośne środowisko uruchomieniowe, najczęściej zarządzane przez platformę taką jak **docker**. 262, 263, 489

LaTeX

System składu tekstu wykorzystywany do przygotowywania profesjonalnych dokumentów technicznych i naukowych. Umożliwia precyzyjne formatowanie, zarządzanie odwołaniami, bibliografią i wzorami matematycznymi. 184

long polling

Odmiana [polling'u](#): [klient](#) wysyła żądanie, a [serwer](#) wstrzymuje odpowiedź do momentu pojawienia się nowych danych (lub upłynięcia limitu czasu).. [431](#)

Maven

Narzędzie do automatyzacji budowania projektów w języku Java, zarządzania zależnościami oraz uruchamiania testów. Konfiguracja procesu budowania i zależności jest definiowana w pliku `pom.xml`.. [423](#), [425](#)

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*).. [281](#), [505](#)

Merge

Proces scalania zmian z jednej gałęzi do innej. Scalenie integruje historię commitów oraz może wymagać rozwiązymania konfliktów, jeśli zmiany dotyczą tych samych fragmentów kodu.. [420](#), [427](#)

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. [308–311](#), [342](#), [345](#), [350](#), [362](#), [365](#)

montowanie

Moment „podpięcia” komponentu do drzewa UI.. [445](#)

Mutacja

W TanStack Query: operacja wysyłająca żądanie zmiany (POST/PUT/PATCH/DELETE) i zarządzająca jego stanem (pending/success/error) oraz reakcją aplikacji, aktualizacją lub unieważnieniem cache.. [293](#), [308](#), [309](#), [311](#), [315](#), [350](#), [361](#), [362](#), [365](#), [368](#)

nagłówek

Metadane dołączane do wiadomości lub żądania (np. dodatkowe pola opisujące typ komunikatu, identyfikator, autoryzację).. [442](#), [443](#)

Node.js

Środowisko uruchomieniowe oparte na silniku V8, umożliwiające wykonywanie kodu JavaScript poza przeglądarką. W projektach frontendowych wykorzystywane do instalacji zależności (npm) oraz uruchamiania narzędzi budowania i testów..
[428](#)

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. [184](#), [490](#)

odmontowanie

Usunięcie komponentu z drzewa UI.. [445](#)

Optimistic UI

Technika w aplikacjach klienckich polegająca na natychmiastowym zaktualizowaniu interfejsu jeszcze przed otrzymaniem potwierdzenia z serwera. Poprawia wrażenie responsywności, a w przypadku błędu po stronie serwera zmiana jest cofana lub oznaczana jako nieudana.. [436](#)

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.. [187](#), [335](#)

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. . [51](#), [52](#), [522](#)

Parametry zapytania (query params)

Pary **klucz=wartość** przekazywane w części adresu URL po znaku zapytania ?, służące m.in. do filtrowania, sortowania, paginacji wyników lub przekazywania dodatkowych opcji żądania do serwera, np. ?param1=val1¶m2=val2. [213](#), [219–229](#), [231](#), [233](#), [235–238](#), [240–243](#), [245–253](#), [255–259](#), [309](#)

Pipeline

Uporządkowany ciąg etapów procesu CI/CD (budowanie, testowanie, wdrażanie), uruchamiany automatycznie po wystąpieniu określonych zdarzeń. W GitHub Actions pipeline jest realizowany jako workflow składający się z jednego lub wielu jobów.. [419](#)

polling

Sposób komunikacji, w którym **klient** cyklicznie „odpytuje” **serwer** (np. co kilka sekund) o nowe dane, zwykle poprzez **HTTP**.. [431](#), [501](#)

prefiks

Ustalony początek ścieżki (np. /app lub /subscribe), który porządkuje i rozróżnia typy komunikacji w aplikacji.. [432](#), [433](#), [436](#)

Prettier

Narzędzie do automatycznego formatowania kodu (np. JavaScript, TypeScript, CSS, HTML). Narzuca spójny styl formatowania, zastępując ręczne ustawianie wcięć i łamań linii. [184](#), [428](#)

Props

Właściwości przekazywane do komponentu React przez komponent nadziedny; służą do konfiguracji i przekazywania danych. Powinny być traktowane jako tylko do odczytu (read-only) wewnątrz komponentu potomnego.. [272](#), [301](#), [304](#), [309](#), [500](#)

Protected route

Trasa w aplikacji, do której dostęp jest ograniczony, zwykle tylko dla zalogowanych użytkowników lub użytkowników z odpowiednimi uprawnieniami. Jeżeli użytkow-

nik nie spełnia warunków, jest przekierowywany (np. na stronę główną). [271](#)

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.. [160](#)

publish–subscribe

Model komunikacji, w którym nadawca publikuje zdarzenia na kanał, a odbiorcy, którzy go [subskrybują](#), automatycznie otrzymują wiadomości.. [430](#), [509](#)

Pull request

Mechanizm służący do zgłoszania propozycji zmian w kodzie i ich scalania do wybranej gałęzi. Umożliwia przegląd kodu (code review), uruchamianie testów oraz dyskusję nad zmianami przed wykonaniem scalenia.. [420](#), [427](#)

Push

Operacja w systemie kontroli wersji polegająca na wysłaniu lokalnych zmian (commitów) do zdalnego repozytorium. [420](#)

ramka

Podstawowa jednostka danych przesyłana w ramach połączenia [WebSocket](#).. [430](#), [431](#)

React

Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. [2](#), [21](#), [156](#), [196](#), [272](#), [445](#)

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. [185](#)

reconnect

Automatyczne ponowne nawiązanie połączenia po jego zerwaniu (np. po chwilowej utracie internetu).. [436](#), [442](#)

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. [26](#), [156](#), [157](#), [159](#), [262](#), [263](#)

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.. [183](#), [272](#), [273](#), [290](#), [293](#), [308](#), [315](#), [324](#), [335](#), [341](#), [342](#), [356](#), [359](#), [361](#), [375](#), [376](#), [379](#), [395](#), [437](#), [447–449](#), [520](#)

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.. [278](#), [281](#), [304](#)

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 zwrocone 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 serwrem, 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.

[2](#), [27](#), [157](#), [209](#), [213](#), [430](#), [431](#), [498](#)

Review kodu

Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. [27](#), [28](#), [183](#), [485](#)

RFC

Request for Comments to numerowane publikacje opisujące ustalenia dotyczące działania Internetu — m.in. protokoły, formaty danych i dobre praktyki. Są wy-

dawane w ramach społeczności IETF.. [429](#), [498](#)

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. [186](#)

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.. [183](#), [407](#)

serializacja

Proces zamiany obiektu (np. w TypeScript/Java) na format przenośny, np. JSON, w celu przesłania lub zapisu. [443](#)

serwer

Część systemu (zwykle po stronie [backendu](#)), która przetwarza żądania i udostępnia dane lub funkcje [klientom](#).. [429–431](#), [433](#), [435](#), [497](#), [499](#), [501](#), [503](#), [510](#), [512](#)

Sidebar

Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. [145](#), [392](#), [485–487](#)

Single Responsibility Principle

(tłum. *Zasada Pojedynczej Odpowiedzialności*); zasada należąca do wytycznych [SOLID](#). Zgodnie z nią klasa lub funkcja powinna mieć dokładnie jeden powód do zmiany, czyli być odpowiedzialną za jedną rzecz. [192](#)

Singleton

Wzorzec projektowy gwarantujący istnienie jednej, współdzielonej instancji obiektu w aplikacji. [447](#)

SockJS

Mechanizm kompatybilności dla [WebSocket](#): gdy przeglądarka lub sieć nie obsługuje WebSocket, SockJS może użyć alternatywnego sposobu komunikacji, aby zachować podobne zachowanie.. [441](#), [442](#)

SOLID

Mnemonik zdefiniowany przez Roberta C. Martina, zawierający 5 zasad projektowania, które mają poprawić elastyczność i prostotę utrzymania oprogramowania.

- **S** – Single Responsibility Principle (tłum. *Zasada Pojedynczej Odpowiedzialności*)
- **O** – Open/Closed Principle (tłum. *Zasada Otwarte/Zamknięte*)
- **L** – Liskov Substitution Principle (tłum. *Zasada Podstawienia Liskov*)
- **I** – Interface Segregation Principle (tłum. *Zasada Segregacji Interfejsów*)
- **D** – Dependency Inversion Principle (tłum. *Zasada Odwrócenia Zależności*)

Definicja została napisana na podstawie treści książki [\[14\]](#) . [507](#)

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.. [22](#), [507](#)

Spot

Potencjalne miejsce do latania dronem, zaznaczone na mapie.. [2](#), [27](#), [79](#), [81–112](#), [152](#), [153](#), [162](#), [163](#), [166](#), [172](#), [173](#), [175](#), [177–180](#), [185](#), [187](#), [188](#), [190](#), [191](#), [284–290](#), [293](#), [298–302](#), [304](#), [306](#), [308–311](#), [315](#), [319](#), [324](#), [334](#), [343](#), [347–350](#), [362–366](#), [431](#), [489](#), [513](#), [514](#), [516](#), [523–525](#), [528](#)

Spring Boot

Framework w ekosystemie Spring dla języka Java, ułatwiający tworzenie aplikacji backendowych dzięki automatycznej konfiguracji, wbudowanemu serwerowi aplikacyjnemu oraz zestawowi gotowych starterów. [2](#), [264](#)

Spring Cache

Mechanizm [cache'owania](#) w ekosystemie Spring, który pozwala przechowywać wyniki wywołań metod (przez adnotacje `@Cacheable`, `@CachePut`, `@CacheEvict`) w celu przyspieszenia działania aplikacji i zmniejszenia obciążenia zasobów (bazy danych).. [264](#)

Spring Framework

Framework, który służy do tworzenia aplikacji w Javie. Zajmuje się między innymi ustawianiem konfiguracji oraz zarządzaniem zależnościami, np. wstrzykiwaniem (ang. *Dependency Injection*), odwróceniem kontroli (ang. *Inversion of Control*). Oferuje wiele modułów wspierających, które ułatwiają rozwój projektów. [18](#), [431](#), [492](#)

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. [183](#)

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).. [196](#), [272](#), [355](#), [363](#), [365](#), [372](#), [375](#), [417](#), [419](#), [437](#), [447–449](#), [520](#)

STOMP

Protokół wiadomościowy działający „nad” [WebSocket](#). Wprowadza pojęcia [destynacji](#), [subskrypcji](#) i wysyłania komunikatów w modelu [publish–subscribe](#).. [430](#), [432](#), [434–436](#), [441–443](#), [445](#), [448](#), [491](#), [494](#)

StompJS

Biblioteka JavaScript/TypeScript (np. `@stomp/stompjs`) implementująca klienta protokołu STOMP. [442](#)

store

Centralne miejsce przechowywania stanu aplikacji w Redux. Pozwala odczytywać aktualny stan oraz wysyłać akcje zmieniające ten stan.. [359](#), [449](#)

subskrypcja

Mechanizm „zapisania się” na kanał ([destynację](#)) w celu automatycznego odbierania wiadomości publikowanych przez serwer.. [430](#), [432–434](#), [436](#), [437](#), [442](#), [445](#), [447](#), [448](#), [494](#), [504](#), [509](#)

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”.. [27](#)

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. [2](#), [183](#), [332](#), [370](#), [387](#)

TanStack Query

Biblioteka do obsługi zapytań do serwera i cachowania danych w aplikacjach front-endowych (m.in. React). Ułatwia zarządzanie stanem danych z backendu: pobieranie, odświeżanie i obsługę błędów. [183](#), [322](#), [344](#)

Testy E2E

Testy *end-to-end*, które sprawdzają działanie systemu od strony użytkownika, przehodząc przez wszystkie warstwy aplikacji (frontend, backend, baza danych) i symulując rzeczywiste scenariusze użycia. [184](#)

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. [184](#)

testy jednostkowe

Testy sprawdzające poprawność działania pojedynczych, małych fragmentów kodu (np. funkcji, metod, klas) w izolacji od reszty systemu.. [184](#)

TTL

(*Time To Live*) Czas życia wpisu w cache lub w innym systemie przechowywania danych — po jego upływie element wygasza i jest usuwany albo uznawany za nieaktualny, co wymusza ponowne pobranie/wyliczenie danych.. [264](#), [266](#)

TypeScript

Rozszerzenie do języka JavaScript dodający statyczne typowanie, interfejsy i narzędzia do większych projektów. Kompiluje się do czystego JavaScript, ułatwiając wykrywanie błędów w czasie komplikacji i refaktoryzacji.. [2](#), [156](#), [272](#)

UI

Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. [21](#), [28](#), [117](#), [164](#), [184](#), [197](#), [206](#), [272](#), [315](#), [342](#)

UML

(ang. *Unified Modeling Language*); graficzny język wizualizacji, specyfikowania oraz dokumentowania składników systemów informatycznych.. [29](#)

URL

Adres zasobu w internecie (ang. *Uniform Resource Locator*), np. adres strony, widoku w aplikacji webowej lub konkretnego posta na forum. [290](#), [345](#), [384](#), [496](#)

useEffect

Hook React służący do uruchamiania efektów ubocznych po renderowaniu (oraz do definiowania funkcji sprzątającej wykonywanej przy odmontowaniu lub zmianie zależności). [445](#)

WebSocket

Protokół komunikacyjny umożliwiający utrzymanie stałego połączenia pomiędzy klientem a serwerem oraz dwukierunkową wymianę danych w czasie zbliżonym do rzeczywistego.. [186](#), [429–433](#), [436](#), [441](#), [445](#), [446](#), [497](#), [504](#), [508](#), [509](#)

Wolumen

Zarządzane przez Dockera magazyny danych dla kontenerów, które są odizolowane od maszyny hosta.. [27](#)

Workflow (GitHub Actions)

Zdefiniowany w pliku YAML proces automatyzacji w GitHub Actions, zawierający wyzwalacze uruchomienia (`on:`) oraz zestaw jobów realizujących zadania CI/CD (budowanie i testowanie aplikacji).. [419–421](#), [427](#), [428](#)

Wzorzec

Powtarzalne, sprawdzone rozwiązanie typowego problemu projektowego lub architektonicznego. Wzorzec opisuje *jak* coś organizować lub implementować, żeby było czytelne, skalowalne i łatwe w utrzymaniu. [189](#), [191–194](#), [196](#), [197](#), [204](#), [206](#)

zmienna środowiskowa

Parametr konfiguracyjny przekazywany aplikacji z zewnątrz (np. plik `.env` lub konfiguracja CI/CD), pozwalający zmieniać zachowanie bez modyfikacji kodu. [446](#)

Spis rysunków

4.1	Wysokopoziomowy diagram przypadków użycia	35
4.2	Ogólny diagram przypadków użycia	36
4.3	Diagram przypadków użycia wyszukiwarki spotów oraz mapy	37
4.4	Diagram przypadków użycia forum	37
4.5	Diagram przypadków użycia czatu	38
4.6	Diagram przypadków użycia profilu użytkownika	39
5.1	Diagram architektury	158
5.2	Projekt prostej strony głównej	161
5.3	Projekt zaawansowanej strony głównej (1)	162
5.4	Projekt zaawansowanej strony głównej (2)	163
5.5	Projekt strony logowania	165
5.6	Projekt strony rejestracji	165
5.7	Widok prywatny profilu	168
5.8	Widok publiczny profilu	168
5.9	Lista obserwujących	169
5.10	Lista obserwowanych	169
5.11	Lista znajomych	170
5.12	Lista zdjęć użytkownika	170
5.13	Zaproszenie wysłane	171
5.14	Możliwość usunięcia z listy znajomych	171
5.15	Możliwość zakończenia obserwowania	172
5.16	Projekt strony z listą ulubionych spotów	173
5.17	Projekt strony z listą dodanych zdjęć	174

5.18 Projekt strony z listą dodanych zdjęć po najechaniu kursorem	175
5.19 Projekt strony z listą dodanych filmów	176
5.20 Projekt strony z listą znajomych	177
5.21 Projekt strony z listą dodanych spotów	179
5.22 Projekt formularza do dodawania nowego spota	179
5.23 Projekt strony z listą dodanych komentarzy	180
5.24 Projekt strony ustawień (1)	181
5.25 Projekt strony ustawień (2)	182
 7.1 Diagram klas wzorca projektowego Fasada	190
7.2 Implementacja wzorca Fasada	191
7.3 Przykładowe użycie klasy PolygonAreaCalculator	191
7.4 Diagram klas wzorca projektowego Singleton	192
7.5 Adnotacja frameworka Spring Boot tworząca Singleton	193
7.6 Użycie Singletona jako zależności	193
7.7 Diagram klas wzorca projektowego Builder	194
7.8 Implementacja wzorca projektowego Builder	195
7.9 Przykładowe użycie wzorca projektowego Builder	195
7.10 Przykładowe użycie hook'a useState	196
7.11 Przykładowe użycie hook'a useEffect	196
7.12 Implementacja hook'a useBoolean	197
7.13 Przykładowe użycie hook'a useBoolean	197
7.14 Implementacja komponentu ChatBottomBar (1)	198
7.15 Implementacja komponentu ChatBottomBar (2)	199
7.16 Implementacja komponentu ChatBottomBar (3)	200
7.17 Implementacja komponentu ChatMessagingWindow (1)	201
7.18 Implementacja komponentu ChatMessagingWindow (2)	202
7.19 Implementacja komponentu ChatMessagingWindow (3)	203
7.20 Implementacja komponentu ChatMessagingWindow (4)	204
7.21 Implementacja komponentu ProtectedRoute	205
7.22 Przykładowe użycie komponentu ProtectedRoute	205
7.23 Implementacja komponentu Modal (1)	207

7.24	Implementacja komponentu Modal (2)	208
7.25	Przykładowe użycie komponentu Modal	209
7.26	Struktura katalogów (1)	211
7.27	Struktura katalogów (2)	212
7.28	Plik konfiguracyjny Docker Compose	263
7.29	Konfiguracja <code>RedisCacheManager</code> wraz z przypisaniem TTL do wybranych przestrzeni cache.	264
7.30	Cachowanie danych z zewnętrznego źródła z wykorzystaniem <code>@Cacheable</code> oraz synchronizacji.	265
7.31	Cachowanie wyników wyszukiwania spotów z kontrolą warunków (<code>condition</code> , <code>unless</code>).	265
7.32	Cachowanie danych pogodowych z kluczem opartym o współrzędne geograficzne.	266
7.33	Struktura katalogów (1)	268
7.34	Struktura katalogów (2)	269
7.35	Implementacja routera (1)	270
7.36	Implementacja routera (2)	271
7.37	Implementacja komponentu bramki (<code>ProtectedRoute</code>)	272
7.38	Konfiguracja sklepu (<code>Redux store</code>)	274
7.39	Przykładowy slice odpowiedzialny za sprawdzenie czy użytkownik jest zalogowany	275
7.40	Implementacja modułu <code>account</code> (1)	276
7.41	Implementacja modułu <code>account</code> (2)	277
7.42	Wykorzystanie TanStack Query przy logowaniu użytkownika	278
7.43	Implementacja zmiennych kolorystycznych (1)	279
7.44	Implementacja zmiennych kolorystycznych (2)	280
7.45	Przykładowe użycie klas Tailwind (w tym prefiksów responsywności)	281
7.46	Przykładowe użycie klas Tailwind (w tym wariantu <code>dark:</code>)	281
7.47	Implementacja animacji z wykorzystaniem Motion	282
7.48	Implementacja prostej wersji wyszukiwarki	283
7.49	Implementacja zaawansowanej wersji wyszukiwarki	283

7.50 Implementacja komponentu do przełączania trybów	284
7.51 Implementacja karuzeli z najpopularniejszymi spotami	285
7.52 Implementacja prostej wyszukiwarki	287
7.53 Implementacja listy do wyświetlania spotów	288
7.54 Implementacja komponentu MapPage (1/3)	291
7.55 Implementacja komponentu MapPage (2/3)	292
7.56 Implementacja komponentu MapPage (3/3)	293
7.57 Implementacja komponentu Spots (1/5)	294
7.58 Implementacja komponentu Spots (2/5)	295
7.59 Implementacja komponentu Spots (3/5)	296
7.60 Implementacja komponentu Spots (4/5)	297
7.61 Implementacja komponentu Spots (5/5)	298
7.62 Implementacja komponentu SearchCurrentViewButton	299
7.63 Implementacja komponentu SpotDetails (1/3)	302
7.64 Implementacja komponentu SpotDetails (2/3)	303
7.65 Implementacja komponentu SpotDetails (3/3)	304
7.66 Implementacja komponentu SpotGeneralInfo (1/2)	305
7.67 Implementacja komponentu SpotGeneralInfo (2/2)	306
7.68 Implementacja komponentu SpotDetailsGallery (1/2)	307
7.69 Implementacja komponentu SpotDetailsGallery (2/2)	308
7.70 Implementacja komponentu AddSpotCommentModal (1/4)	311
7.71 Implementacja komponentu AddSpotCommentModal (2/4)	312
7.72 Implementacja komponentu AddSpotCommentModal (3/4)	313
7.73 Implementacja komponentu AddSpotCommentModal (4/4)	314
7.74 Implementacja komponentu SpotCommentVotesPanel (1/3)	316
7.75 Implementacja komponentu SpotCommentVotesPanel (2/3)	317
7.76 Implementacja komponentu SpotCommentVotesPanel (3/3)	318
7.77 Implementacja komponentu BasicSpotWeather (1/2)	320
7.78 Implementacja komponentu BasicSpotWeather (2/2)	321
7.79 Implementacja komponentu DetailedSpotWeather (1/2)	323
7.80 Implementacja komponentu DetailedSpotWeather (2/2)	324

7.81 Implementacja komponentu WeatherOverview (1/2)	325
7.82 Implementacja komponentu WeatherOverview (2/2)	326
7.83 Implementacja komponentu WindSpeeds (1/2)	327
7.84 Implementacja komponentu WindSpeeds (2/2)	328
7.85 Implementacja komponentu WeatherTimelinePlot (1/4)	329
7.86 Implementacja komponentu WeatherTimelinePlot (2/4)	330
7.87 Implementacja komponentu WeatherTimelinePlot (3/4)	331
7.88 Implementacja komponentu WeatherTimelinePlot (4/4)	332
7.89 Implementacja komponentu WeatherIcon (1/2)	333
7.90 Implementacja komponentu WeatherIcon (2/2)	334
7.91 Implementacja komponentu ExpandedSpotMediaGallery (1/8) . . .	336
7.92 Implementacja komponentu ExpandedSpotMediaGallery (2/8) . . .	336
7.93 Implementacja komponentu ExpandedSpotMediaGallery (3/8) . . .	337
7.94 Implementacja komponentu ExpandedSpotMediaGallery (4/8) . . .	338
7.95 Implementacja komponentu ExpandedSpotMediaGallery (5/8) . . .	338
7.96 Implementacja komponentu ExpandedSpotMediaGallery (6/8) . . .	339
7.97 Implementacja komponentu ExpandedSpotMediaGallery (7/8) . . .	340
7.98 Implementacja komponentu ExpandedSpotMediaGallery (8/8) . . .	341
7.99 Komponent <code>UserOwnProfile</code>	344
7.100 Komponent <code>ProfileForViewer</code>	346
7.101 Komponent <code>FavoriteSpots</code>	349
7.102 Komponent <code>Photos</code>	351
7.103 Komponent <code>Media</code>	353
7.104 Komponent <code>Movies</code>	355
7.105 Komponent <code>Social</code> (1).	357
7.106 Komponent <code>Social</code> (2).	358
7.107 Komponent <code>Social</code>	360
7.108 Komponent <code>AddedSpot</code>	364
7.109 Komponent <code>Comments</code>	367
7.110 Komponent <code>Settings</code>	369
7.111 Komponent <code>AccountWrapper</code>	371

7.112Komponent AccountTitle.	372
7.113Wybór zakresu dat w panelu filtrowania (DateChooser).	373
7.114Komponent SortAndDateFilters.	374
7.115Komponent DateBadge.	375
7.116Komponent Login (1).	377
7.117Komponent Login (2).	378
7.118Komponent Register (1).	380
7.119Komponent Register (2).	381
7.120Komponent ForgotPassword.	383
7.121Komponent NewPassword (1).	385
7.122Komponent NewPassword (2).	386
7.123Komponent FormContainer (1).	388
7.124Komponent FormContainer (2).	389
7.125Komponent FormInput (1).	390
7.126Komponent FormInput (2).	391
7.127Komponent SubmitFormButton.	392
7.128Fragment kodu komponentu Sidebar.	394
7.129Fragment kodu komponentu Tooltip.	395
7.130Fragment kodu komponentu SidebarToggleButton.	396
7.131Fragment kodu komponentu SidebarSection.	397
7.132Fragment kodu komponentu SidebarList.	398
7.133Fragment kodu komponentu SidebarLabel.	399
7.134Fragment kodu komponentu SidebarIcon.	400
7.135Fragment kodu komponentu SidebarItemSubmenuLink.	401
7.136Fragment kodu komponentu SidebarItemSubmenu (1).	402
7.137Fragment kodu komponentu SidebarItemSubmenu (2).	403
7.138Fragment kodu komponentu SidebarItemSubmenu (3).	404
7.139Fragment kodu komponentu SidebarItemLink (1).	405
7.140Fragment kodu komponentu SidebarItemLink (2).	406
7.141Fragment kodu komponentu SidebarItemContent.	407
7.142Fragment kodu komponentu SidebarItemAction (1).	408

7.143	Fragment kodu komponentu <code>SidebarItemAction</code> (2)	409
7.144	Fragment kodu komponentu <code>SidebarItem</code> (1)	410
7.145	Fragment kodu komponentu <code>SidebarItem</code> (2)	411
7.146	Fragment kodu pliku <code>functions</code> (funkcje typu <i>guard</i>)	412
7.147	Fragment kodu pliku <code>sidebarLinks</code> (linki statyczne)	413
7.148	Fragment kodu pliku <code>sidebarLinks</code> (linki dostępne dla użytkownika zalogowanego)	414
7.149	Fragment kodu pliku <code>sidebarLinks</code> (linki opcji: logowanie/wylogowanie oraz zmiana motywów)	415
7.150	Fragment kodu pliku <code>link</code>	416
7.151	Slice <code>Redux</code> odpowiadający za stan paska bocznego	417
7.152	Fragment kodu komponentu <code>Layout</code>	418
7.153	Fragment kodu komponentu <code>MobileBar</code>	419
7.154	Fragment pliku <code>java-ci.yml</code> : konfiguracja wyzwalaczy workflow (<code>on:</code>)	421
7.155	Podsumowanie wykonania workflow backendu (<code>java-ci.yml</code>) w GitHub Actions	421
7.156	Fragment pliku <code>java-ci.yml</code> : definicja joba <code>setup</code>	422
7.157	Przykładowy log wykonania joba <code>setup</code> w GitHub Actions	422
7.158	Fragment pliku <code>java-ci.yml</code> : definicja joba <code>build</code>	424
7.159	Przykładowy log wykonania joba <code>build</code> w GitHub Actions	425
7.160	Fragment pliku <code>java-ci.yml</code> : definicja joba <code>test</code> (wraz z użyciem <code>secrets</code>)	426
7.161	Przykładowy log wykonania joba <code>test</code> w GitHub Actions	427
7.162	Fragment pliku <code>react-ci.yml</code> : konfiguracja workflow oraz joba <code>formatting-and-test</code>	428
7.163	Podsumowanie wykonania workflow frontendu (<code>react-ci.yml</code>) w GitHub Actions	428
7.164	Przykładowy log wykonania joba <code>formatting-and-test</code> w GitHub Actions	429
7.165	Konfiguracja WebSocket po stronie backendu	433
7.166	Kontroler STOMP odbierający wiadomości czatu	434

7.167	Serwis odpowiedzialny za komunikację za pomocą WebSocket.	435
7.168	Serwis komunikacji WebSocket po stronie frontendu (część 1/4).	438
7.169	Serwis komunikacji WebSocket po stronie frontendu (część 2/4).	439
7.170	Serwis komunikacji WebSocket po stronie frontendu (część 3/4).	440
7.171	Serwis komunikacji WebSocket po stronie frontendu (część 4/4).	441
7.172	Mechanizm rejestracji subskrypcji w cyklu życia komponentu (część 1/2).	444
7.173	Mechanizm rejestracji subskrypcji w cyklu życia komponentu (część 2/2).	445
7.174	Udostępnienie serwisu komunikacji oraz zarządzanie cyklem życia połączenia po stronie frontendu.	446
7.175	Definicja subskrypcji czatu i aktualizacja stanu w Redux.	448
8.1	Wynik uruchomienia testów jednostkowych warstwy frontendowej .	452
8.2	Zestaw testów jednostkowych uruchomionych dla warstwy backen- dowej	453
8.3	Podsumowanie uruchomienia testów jednostkowych warstwy bac- kendowej	453
8.4	Wynik uruchomienia testów integracyjnych warstwy frontendowej .	454
8.5	Zestaw testów integracyjnych uruchomionych dla warstwy backen- dowej	455
8.6	Podsumowanie uruchomienia testów integracyjnych warstwy bac- kendowej	455
8.7	Podsumowanie uruchomienia testów end-to-end (E2E) w Cypress .	457

Spis tabel

Tabela 2.1: Karta udziałowca: Zespół projektowy	11
Tabela 2.2: Karta udziałowca: Promotor	12
Tabela 2.3: Karta udziałowca: Droniarze	13
Tabela 3.1: Zestawienie wszystkich bibliotek użytych na backendzie.	21
Tabela 3.2: Zestawienie wszystkich bibliotek i pluginów użytych na frontendzie.	25
Tabela 3.3: Usługa zewnętrzna: GitHub Actions (CI)	30
Tabela 3.4: Usługa zewnętrzna: Azure Blob Storage	30
Tabela 3.5: Usługa zewnętrzna: Mailtrap	30
Tabela 3.6: Usługa zewnętrzna: LocationIQ	31
Tabela 3.7: Usługa zewnętrzna: Google Maps (Maps URLs)	31
Tabela 3.8: Usługa zewnętrzna: OpenFreeMap	31
Tabela 3.9: Usługa zewnętrzna: Open-Meteo	32
Tabela 3.10: Usługa zewnętrzna: Tenor GIF API	32
Tabela 3.11: Usługa zewnętrzna: Where the ISS at?	32
Tabela 4.1: Scenariusz przypadku użycia: Rejestracja użytkownika	40
Tabela 4.2: Scenariusz przypadku użycia: Logowanie użytkownika	41
Tabela 4.3: Scenariusz przypadku użycia: Resetowanie hasła	42
Tabela 4.4: Scenariusz przypadku użycia: Zmiana hasła w ustawieniach konta	43
Tabela 4.5: Scenariusz przypadku użycia: Wylogowanie użytkownika	43
Tabela 4.6: Scenariusz przypadku użycia: Przeglądanie powiadomień . . .	44

Tabela 4.7: Scenariusz przypadku użycia: Wyszukiwanie spota w globalnej wyszukiwarce	45
Tabela 4.8: Scenariusz przypadku użycia: Przejście do spota na mapie z wyszukiwarki	46
Tabela 4.9: Scenariusz przypadku użycia: Przeglądanie mapy spotów	47
Tabela 4.10: Scenariusz przypadku użycia: Otwarcie szczegółów spota	48
Tabela 4.11: Scenariusz przypadku użycia: Przeglądanie komentarzy do spota	48
Tabela 4.12: Scenariusz przypadku użycia: Przeglądanie pogody na spocie	49
Tabela 4.13: Scenariusz przypadku użycia: Wyszukiwanie spota na mapie	50
Tabela 4.14: Scenariusz przypadku użycia: Zmiana typu mapy	51
Tabela 4.15: Scenariusz przypadku użycia: Przeglądanie stref PANSA	52
Tabela 4.16: Scenariusz przypadku użycia: Utworzenie prywatnego czatu	52
Tabela 4.17: Scenariusz przypadku użycia: Otworzenie czatu	53
Tabela 4.18: Scenariusz przypadku użycia: Utworzenie czatu grupowego .	54
Tabela 4.19: Scenariusz przypadku użycia: Przeglądanie listy czatów	55
Tabela 4.20: Scenariusz przypadku użycia: Wysyłanie wiadomości na czacie	56
Tabela 4.21: Scenariusz przypadku użycia: Wysyłanie GIF-a na czacie	57
Tabela 4.22: Scenariusz przypadku użycia: Wysyłanie pliku na czacie	58
Tabela 4.23: Scenariusz przypadku użycia: Edycja ustawień czatu	58
Tabela 4.24: Scenariusz przypadku użycia: Dodanie członka do czatu grupowego	59
Tabela 4.25: Scenariusz przypadku użycia: Przeszukiwanie historii czatu .	60
Tabela 4.26: Scenariusz przypadku użycia: Przeglądanie postów na forum	61
Tabela 4.27: Scenariusz przypadku użycia: Wyszukiwanie postów na forum	62
Tabela 4.28: Scenariusz przypadku użycia: Dodanie posta na forum	63
Tabela 4.29: Scenariusz przypadku użycia: Dodanie komentarza na forum	64
Tabela 4.30: Scenariusz przypadku użycia: Przeglądanie historii interakcji z postami	65
Tabela 4.31: Scenariusz przypadku użycia: Zarządzanie komentarzami na forum	66

Tabela 4.32: Scenariusz przypadku użycia: Zgłoszenie komentarza naruszającego regulamin	67
Tabela 4.33: Scenariusz przypadku użycia: Zgłoszenie posta na forum	68
Tabela 4.34: Scenariusz przypadku użycia: Przeglądanie komentarzy pod postem	69
Tabela 4.35: Scenariusz przypadku użycia: Dodanie spota w panelu użytkownika	70
Tabela 4.36: Scenariusz przypadku użycia: Przeglądanie profilu użytkownika	71
Tabela 4.37: Scenariusz przypadku użycia: Przeglądanie profilu innego użytkownika	72
Tabela 4.38: Scenariusz przypadku użycia: Dodanie użytkownika do znamionych	72
Tabela 4.39: Scenariusz przypadku użycia: Przeglądanie społeczności	73
Tabela 4.40: Scenariusz przypadku użycia: Przeglądanie dodanych spotów	74
Tabela 4.41: Scenariusz przypadku użycia: Edycja danych użytkownika	75
Tabela 4.42: Scenariusz przypadku użycia: Przeglądanie dodanych zdjęć do spotów	76
Tabela 4.43: Scenariusz przypadku użycia: Przeglądanie dodanych filmów do spotów	77
Tabela 4.44: Scenariusz przypadku użycia: Przeglądanie dodanych komentarzy do spotów	78
Tabela 4.45: Karta wymagania ogólnego dla czatu: Wysyłanie wiadomości na czacie	80
Tabela 4.46: Karta wymagania ogólnego dla czatu: Edycja czatu	80
Tabela 4.47: Karta wymagania ogólnego dla czatu: Przeglądanie historii czatu	80
Tabela 4.48: Karta wymagania ogólnego dla czatu: Tworzenie czatu	81
Tabela 4.49: Karta wymagania ogólnego dla mapy: Wyświetlanie spotów na mapie	81

Tabela 4.50: Karta wymagania ogólnego dla mapy: Wyświetlanie szczegółów spota	82
Tabela 4.51: Karta wymagania ogólnego dla mapy: Wyświetlanie informacji pogodowych spota	82
Tabela 4.52: Karta wymagania ogólnego dla mapy: Wyszukiwanie spotów	83
Tabela 4.53: Karta wymagania ogólnego dla mapy: Komentowanie spotów	83
Tabela 4.54: Karta wymagania ogólnego dla mapy: Dodawanie media do spotów	83
Tabela 4.55: Karta wymagania ogólnego dla mapy: Wyświetlenie lokalizacji użytkownika na mapie	84
Tabela 4.56: Karta wymagania ogólnego dla wyszukiwarki spotów: Wy szukiwanie na prostej wyszukiwarce	84
Tabela 4.57: Karta wymagania ogólnego dla wyszukiwarki spotów: Wy szukiwanie na zaawansowanej wyszukiwarce	85
Tabela 4.58: Wymaganie funkcjonalne dla mapy: Wyświetlanie spotów na mapie	86
Tabela 4.59: Wymaganie funkcjonalne dla mapy: Wyświetlanie szczegółów spota	87
Tabela 4.60: Wymaganie funkcjonalne dla mapy: Wyświetlanie interaktywnej galerii mediów spota	87
Tabela 4.61: Wymaganie funkcjonalne dla mapy: Wyświetlanie przewijalnej listy komentarzy spota	88
Tabela 4.62: Wymaganie funkcjonalne dla mapy: Ocena komentarza spota	89
Tabela 4.63: Wymaganie funkcjonalne dla mapy: Dodanie komentarza do spota	91
Tabela 4.64: Wymaganie funkcjonalne dla mapy: Dodanie media do spota	92
Tabela 4.65: Wymaganie funkcjonalne dla mapy: Dodanie spota do listy ulubionych	93
Tabela 4.66: Wymaganie funkcjonalne dla mapy: Udostępnienie spota	94
Tabela 4.67: Wymaganie funkcjonalne dla mapy: Nawigowanie do spota	95

Tabela 4.68: Wymaganie funkcjonalne dla mapy: Duża galeria mediów spota	96
Tabela 4.69: Wymaganie funkcjonalne dla mapy: Zarządzanie listą medii w dużej galerii mediów spota	97
Tabela 4.70: Wymaganie funkcjonalne dla mapy: Wyświetlanie powiększonego obecnie wybranego media w dużej galerii mediów spota	98
Tabela 4.71: Wymaganie funkcjonalne dla mapy: Udostępnienie obecnie wybranego media w dużej galerii mediów spota	99
Tabela 4.72: Wymaganie funkcjonalne dla mapy: Polubienie obecnie wybranego media w dużej galerii mediów spota	100
Tabela 4.73: Wymaganie funkcjonalne dla mapy: Powiększenie na cały ekran obecnie wybranego media w dużej galerii mediów spota	101
Tabela 4.74: Wymaganie funkcjonalne dla mapy: Pobranie obecnie wybranego media w dużej galerii mediów spota	102
Tabela 4.75: Wymaganie funkcjonalne dla mapy: Wyświetlanie informacji pogodowych spota	103
Tabela 4.76: Wymaganie funkcjonalne dla mapy: Wyświetlanie szczegółowych informacji pogodowych spota	104
Tabela 4.77: Wymaganie funkcjonalne dla mapy: Wyszukiwanie spotów po nazwie	107
Tabela 4.78: Wymaganie funkcjonalne dla mapy: Sortowanie wyników wyszukiwania spotów po nazwie	108
Tabela 4.79: Wymaganie funkcjonalne dla mapy: Sortowanie wyników wyszukiwania spotów po nazwie	109
Tabela 4.80: Wymaganie funkcjonalne dla mapy: Zarządzanie listą wyników wyszukiwania spotów w widocznym obszarze mapy	112
Tabela 4.81: Wymaganie funkcjonalne dla mapy: Przybliżanie mapy do lokalizacji spotów z listy z wynikami	112
Tabela 4.82: Wymaganie funkcjonalne dla mapy: Wyświetlanie na mapie lokalizacji użytkownika	113

Tabela 4.83: Wymaganie funkcjonalne dla mapy: Ustawianie przybliżenia mapy	114
Tabela 4.84: Wymaganie funkcjonalne dla czatu: Wysyłanie GIF'ów	115
Tabela 4.85: Wymaganie funkcjonalne dla czatu: Wysyłanie plików	116
Tabela 4.86: Wymaganie funkcjonalne dla czatu: Wysyłanie wiadomości prywatnych	116
Tabela 4.87: Wymaganie funkcjonalne dla czatu: Wysyłanie wiadomości do wielu osób jednocześnie	117
Tabela 4.88: Wymaganie funkcjonalne dla czatu: Rozpoczynanie nowego czatu	118
Tabela 4.89: Wymaganie funkcjonalne dla czatu: Wysyłanie emotikonów .	118
Tabela 4.90: Wymaganie funkcjonalne dla czatu: Dostępność czatu po utworzeniu	119
Tabela 4.91: Wymaganie funkcjonalne dla czatu: Edytowanie nazwy czatu grupowego	120
Tabela 4.92: Wymaganie funkcjonalne dla czatu: Edycja zdjęcia czatu grupowego	121
Tabela 4.93: Wymaganie funkcjonalne dla czatu: Edycja wysłanej wiadomości	121
Tabela 4.94: Wymaganie funkcjonalne dla czatu: Dodawanie użytkowników do istniejącego czatu	122
Tabela 4.95: Wymaganie funkcjonalne dla czatu: Wyświetlanie starszych wiadomości	123
4.96 Profil użytkownika	124
4.97 Lista dodanych spotów	125
4.98 Dodanie spota	126
4.99 Lista zdjęć	127
4.100 Lista filmów	127
4.101 Lista znajomych	128
4.102 Lista obserwujących	128
4.103 Lista obserwowanych	129

4.104	Lista polubionych/odwiedzonych/planowanych spotów	129
4.105	Lista komentarzy	130
4.106	Ustawienia profilu	131
4.107	Resetowanie hasła	132
4.108	Dodawanie do znajomych	133
4.109	Logowanie i rejestracja	134
Tabela 4.96:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Wyświe-	
	tlenie najpopularniejszych spotów w karuzeli	135
Tabela 4.97:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Wysz-	
	ukiwanie za pomocą państwa, regionu oraz miasta	136
Tabela 4.98:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Wyświe-	
	tlenie wyszukanych spotów	137
Tabela 4.99:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Podpo-	
	wiedzi wartości w polach lokalizacji (autocomplete)	138
Tabela 4.100:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Przy-	
	cisk do pokazania spota na mapie oraz zobaczenia jego szczegółów .	139
Tabela 4.101:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Wy-	
	szukiwanie za pomocą miasta oraz tagów	140
Tabela 4.102:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Sorto-	
	wanie po popularności oraz ocenie	142
Tabela 4.103:	Wymaganie funkcjonalne dla wyszukiwarki spotów: Filtro-	
	wanie po ocenie	142
4.118	Ustawienia motywu (ręczna zmiana)	143
4.119	Zapamiętanie preferencji motywu	144
4.120	Szybki przełącznik motywu w interfejsie	145
Tabela 4.104:	Wymaganie pozafunkcjonalne dla czatu: Dostęp do czatów	
	ograniczony do uczestników (autoryzacja)	146
Tabela 4.105:	Wymaganie pozafunkcjonalne dla czatu: Korzystanie z czatu	
	wymaga zalogowania (uwierzytelnienie)	147
Tabela 4.106:	Wymaganie pozafunkcjonalne dla czatu: Grupowanie wi-	
	domości według daty wysłania	148

Tabela 4.107: Wymaganie pozafunkcjonalne dla czatu: Wyraźne oznaczenie nadawcy i czasu wysłania	149
Tabela 4.108: Wymaganie pozafunkcjonalne dla czatu: Czas załadowania starszych wiadomości poniżej 10 sekund	149
Tabela 4.109: Wymaganie pozafunkcjonalne dla czatu: Natychmiastowe wysyłanie wiadomości	150
Tabela 4.110: Wymaganie pozafunkcjonalne dla czatu: Integracja z API Tenor	151
Tabela 4.111: Wymaganie pozafunkcjonalne dla mapy: Hostowanie mapy na niezależnym serwerze	152
Tabela 4.112: Wymaganie pozafunkcjonalne dla mapy: Czas ładowania mapy poniżej 10 sekund	152
Tabela 4.113: Wymaganie pozafunkcjonalne dla mapy: Czas pobierania spotów poniżej 10 sekund	153
Tabela 4.114: Wymaganie pozafunkcjonalne dla wyszukiwarki spotów: Czas ładowania wyszukanych spotów nie przekracza 10 sekund	154
Tabela 4.115: Wymaganie pozafunkcjonalne dla wyszukiwarki spotów: Podpowiedzi w polach lokalizacji (kraj/region/miasto)	155
 Tabela 7.1: Zestawienie endpointów: panelu użytkownika	214
Tabela 7.2: Zestawienie endpointów: modułu spotów	215
Tabela 7.3: Zestawienie endpointów: komentarzy do spotów	216
Tabela 7.4: Zestawienie endpointów: postów forum	216
Tabela 7.5: Zestawienie endpointów: komentarzy forum	217
Tabela 7.6: Zestawienie endpointów: konta użytkownika i autoryzacji	217
Tabela 7.7: Zestawienie endpointów: integracji GIF-ów	218
Tabela 7.8: Zestawienie endpointów: modułu czatu	218
Tabela 7.9: Karta endpointu: /public/user-dashboard/profile/{targetUsername}	219
Tabela 7.10: Karta endpointu: /user-dashboard/favorite-spots	220
Tabela 7.11: Karta endpointu: /user-dashboard/photos	221
Tabela 7.12: Karta endpointu: /user-dashboard/add-spot	222
Tabela 7.13: Karta endpointu: /user-dashboard/add-spot	223

Tabela 7.14: Karta endpointu: /public/spot/gallery	224
Tabela 7.15: Karta endpointu: /public/spot/current-view	225
Tabela 7.16: Karta endpointu: /public/spot/get-spot-basic-weather . . .	226
Tabela 7.17: Karta endpointu: /public/spot/get-spot-detailed-weather . .	227
Tabela 7.18: Karta endpointu: /public/spot/get-spot-wind-speeds	228
Tabela 7.19: Karta endpointu: /public/spot/most-popular	229
Tabela 7.20: Karta endpointu: /public/spot/search/home-page	230
Tabela 7.21: Karta endpointu: /public/spot/search/home-page/locations	231
Tabela 7.22: Karta endpointu: /public/spot/search/home-page/advance .	232
Tabela 7.23: Karta endpointu: /public/spot/{spotId}/comments	234
Tabela 7.24: Karta endpointu: /public/spot/{spotId}/comments/{commentId}	235
Tabela 7.25: Karta endpointu: /spot/{spotId}/comments	236
Tabela 7.26: Karta endpointu: /spot/comments/{commentId}/vote	237
Tabela 7.27: Karta endpointu: /spot/comments/vote-type	237
Tabela 7.28: Karta endpointu: /public/post/{postId}	239
Tabela 7.29: Karta endpointu: /post	240
Tabela 7.30: Karta endpointu: /post/{postId}	241
Tabela 7.31: Karta endpointu: /post/{postId}/vote	241
Tabela 7.32: Karta endpointu: /public/categories-tags	242
Tabela 7.33: Karta endpointu: /public/post/{postId}/comments	244
Tabela 7.34: Karta endpointu: /post/{postId}/comments	245
Tabela 7.35: Karta endpointu: /post/comments/{commentId}	246
Tabela 7.36: Karta endpointu: /post/comments/{commentId}/vote	246
Tabela 7.37: Karta endpointu: /comments/{commentId}/replies	247
Tabela 7.38: Karta endpointu: /public/account/register	248
Tabela 7.39: Karta endpointu: /public/account/login	248
Tabela 7.40: Karta endpointu: /public/account/forgot-password	249
Tabela 7.41: Karta endpointu: /public/account/set-new-password	250
Tabela 7.42: Karta endpointu: /account/check	250
Tabela 7.43: Karta endpointu: /account/login-success	251
Tabela 7.44: Karta endpointu: /gifs/trending	252

Tabela 7.45: Karta endpointu: /gifs/search	253
Tabela 7.46: Karta endpointu: /chats/{chatId}/messages	254
Tabela 7.47: Karta endpointu: /chats/get-or-create-private-chat	255
Tabela 7.48: Karta endpointu: /chats/{chatId}/send-files	256
Tabela 7.49: Karta endpointu: /chats/create/group	257
Tabela 7.50: Karta endpointu: /chats/{chatId}	258
Tabela 7.51: Karta endpointu: /chats/group-chat/add/search/{chatId} .	259
Tabela 8.1: Scenariusz E2E: Logowanie użytkownika	458
Tabela 8.2: Scenariusz E2E: Rejestracja użytkownika	459
Tabela 8.3: Scenariusz E2E: Wyświetlenie stanu pustego dla dodanych spotów	459
Tabela 8.4: Scenariusz E2E: Ładowanie kolejnych elementów listy przy przewijaniu (<i>infinite scroll</i>)	460
Tabela 8.5: Scenariusz E2E: Otwarcie modala dodawania spota na widoku desktop	460
Tabela 8.6: Scenariusz E2E: Zablokowanie dodawania spota na małym ekranie	461
Tabela 8.7: Scenariusz E2E: Dodanie nowego spota z użyciem rzeczywistego backendu	462
Tabela 8.8: Scenariusz E2E: Wyświetlenie stanu pustego dla komentarzy .	462
Tabela 8.9: Scenariusz E2E: Sortowanie komentarzy po zmianie typu sortowania	463
Tabela 8.10: Scenariusz E2E: Załadowanie widoku komentarzy po Rzeczywistym logowaniu	464
Tabela 8.11: Scenariusz E2E: Wyświetlenie stanu pustego dla list ulubionych spotów	464
Tabela 8.12: Scenariusz E2E: Zmiana typu listy	465
Tabela 8.13: Scenariusz E2E: Dociąganie kolejnej strony przy przewijaniu	465
Tabela 8.14: Scenariusz E2E: Załadowanie danych z rzeczywistego backendu po logowaniu	466
Tabela 8.15: Scenariusz E2E: Stan pusty dla filmów	466

Tabela 8.16: Scenariusz E2E: Zmiana sortowania filmów	467
Tabela 8.17: Scenariusz E2E: <i>Infinite scroll</i> w widoku filmów	467
Tabela 8.18: Scenariusz E2E: Załadowanie filmów po logowaniu na Rze- czywistym backendzie	468
Tabela 8.19: Scenariusz E2E: Stan pusty dla zdjęć	469
Tabela 8.20: Scenariusz E2E: Zmiana sortowania zdjęć	469
Tabela 8.21: Scenariusz E2E: <i>Infinite scroll</i> w widoku zdjęć	469
Tabela 8.22: Scenariusz E2E: Widok zdjęć po logowaniu na Rzeczywistym backendzie	470
Tabela 8.23: Scenariusz E2E: Wyświetlenie profilu użytkownika (staty- styki i popularne zdjęcia)	471
Tabela 8.24: Scenariusz E2E: Komunikat o braku zdjęć w profilu	471
Tabela 8.25: Scenariusz E2E: Nawigacja do listy znajomych z poziomu profilu	472
Tabela 8.26: Scenariusz E2E: Nawigacja do zdjęć z poziomu profilu . . .	472
Tabela 8.27: Scenariusz E2E: Wyświetlenie profilu innego użytkownika (akcje follow/friends)	472
Tabela 8.28: Scenariusz E2E: Wysłanie żądania follow oraz zaproszenia do znajomych	473
Tabela 8.29: Scenariusz E2E: Załadowanie profilu własnego po logowaniu na Rzeczywistym backendzie	473
Tabela 8.30: Scenariusz E2E: Wyświetlenie danych konta i otwarcie for- mularza zmiany nazwy użytkownika	474
Tabela 8.31: Scenariusz E2E: Zmiana nazwy użytkownika	475
Tabela 8.32: Scenariusz E2E: Zmiana adresu e-mail	475
Tabela 8.33: Scenariusz E2E: Zmiana hasła	476
Tabela 8.34: Scenariusz E2E: Ograniczone ustawienia dla konta OAuth .	476
Tabela 8.35: Scenariusz E2E: Wyświetlenie ustawień po logowaniu na Rzeczywistym backendzie	477
Tabela 8.36: Scenariusz E2E: Zmiana hasła na Rzeczywistym backendzie oraz ponowne logowanie	477

Tabela 8.37: Scenariusz E2E: Stan pusty listy znajomych	478
Tabela 8.38: Scenariusz E2E: Przełączanie zakładek friends/followed/fol-	
lowers	478
Tabela 8.39: Scenariusz E2E: <i>Infinite scroll</i> na liście znajomych	479
Tabela 8.40: Scenariusz E2E: Obsługa zaproszeń do znajomych	479

Bibliografia

- [1] *Disciplined Agile Delivery*. Project Management Institute. URL: <https://www.pmi.org/disciplined-agile/process/introduction-to-dad/why> (dostęp 30.10.2025).
- [2] *Disciplined Agile Delivery — Lean Life Cycle*. Project Management Institute. 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-wskazovek-poprawiajacych-modelowanie-procesow-biznesowych-w-notacji-bpmn/> (dostęp 19.11.2025).
- [5] *About billing for GitHub Actions*. GitHub Docs. 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. URL: <https://learn.microsoft.com/azure/storage/blobs/scalability-targets> (dostęp 2.11.2025).
- [7] *What are the limitations in Mailtrap?* Mailtrap Docs. URL: <https://help.mailtrap.io/article/111-what-are-the-limitations-in-mailtrap/> (dostęp 2.11.2025).
- [8] *LocationIQ Pricing*. LocationIQ. URL: <https://locationiq.com/pricing> (dostęp 2.11.2025).
- [9] *Google Maps (Maps URLs)*. Google Maps. URL: <https://developers.google.com/maps/documentation/urls/get-started?hl=pl> (dostęp 2.11.2025).
- [10] *OpenFreeMap Documentation*. OpenFreeMap. URL: <https://openfreemap.org/docs> (dostęp 2.11.2025).

- [11] *Open-Meteo API Usage & Pricing*. Open-Meteo. URL: <https://open-meteo.com/en/docs/usage-and-pricing> (dostęp 2.11.2025).
- [12] *Tenor API — Documentation*. Tenor. URL: <https://tenor.com/gifapi/documentation> (dostęp 2.11.2025).
- [13] *Where the ISS at? API*. wheretheiss.at. URL: <https://wheretheiss.at/> (dostęp 2.11.2025).
- [14] Aleksander Shvets., „Wzorce Projektowe Nowoczesny Podręcznik”. Pamplona: Refactoring Guru, 2022.
- [15] *React useState*. URL: <https://react.dev/reference/react/useState> (dostęp 3.11.2025).
- [16] *Redux*. URL: <https://redux.js.org/> (dostęp 3.11.2025).
- [17] *Axios*. URL: <https://axios-http.com/> (dostęp 3.11.2025).
- [18] *Tanstack Query*. URL: <https://tanstack.com/query/latest> (dostęp 3.11.2025).
- [19] *Tailwind*. URL: <https://tailwindcss.com/> (dostęp 3.11.2025).
- [20] *Motion*. URL: <https://motion.dev/> (dostęp 3.11.2025).
- [21] *RFC 6455: The WebSocket Protocol*. IETF. URL: <https://datatracker.ietf.org/doc/html/rfc6455> (dostęp 16.12.2025).
- [22] *WebSocket (Spring Framework Reference Documentation)*. Spring. URL: <https://docs.spring.io/spring-framework/reference/web/websocket.html> (dostęp 16.12.2025).
- [23] *STOMP over WebSocket (Spring Framework Reference Documentation)*. Spring. URL: <https://docs.spring.io/spring-framework/reference/web/websocket/stomp.html> (dostęp 16.12.2025).
- [24] *sockjs-client (WebSocket emulation — JavaScript client)*. SockJS. URL: <https://github.com/sockjs/sockjs-client> (dostęp 16.12.2025).

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.