



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 Adam Urbanowicz

Gdańsk, miesiąc, 2100 obrony

Streszczenie: Celem niniejszej pracy było stworzenie w pełni funkcjonalnej i działającej aplikacji internetowej pozwalającej na szybkie wyszukiwanie spotów w okolicy oraz dzielenie się zdjęciami, filmami oraz doświadczeniem z innymi użytkownikami. W ramach pracy stworzono system składający się z trzech komponentów: [Frontendu](#), [Backendu](#) oraz bazy-danych. Aplikacja internetowa została wykonana przy pomocy [Frameworka](#) React w językach Javascript oraz Typescript, do stylów został użyty Tailwind. Serwis backendowy został stworzony w języku Java oraz biblioteki Spring Boot. Baza danych to PostgreSql.

Komunikacja między komponentami odbywała się zgodnie ze standardem REST. Projekt został zrealizowany w podejściu ewolucyjno-przyrostowym z elementami Kanban.

Słowa kluczowe: — brak —



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Karta projektu

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

Wykonawcy	Numer 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: 21 grudnia 2025	Recenzent: dr Elżbieta Puźniakowska-Gałuch
---	--

Spis treści

1 Wstęp	8
Uwagi redakcyjne	8
1.1 O projekcie	8
1.2 Cel i zakres prac	8
1.3 Geneza pomysłu	8
2 Opis problemu	9
2.1 Rich picture	9
2.2 Udziałowcy	9
2.3 Istniejące rozwiązania	11
2.4 Wizja rozwiązania	11
2.5 Aspekty społeczne i biznesowe	11
2.5.1 Aspekty społeczne	11
2.5.2 Aspekty biznesowe	11
3 Planowanie	12
3.1 Metodologia pracy	12
3.1.1 Przegląd rozważanych podejść	12
3.1.2 Odrzucone podejścia	12
3.1.3 Wybrane podejście: Disciplined Agile Delivery (Lean Life Cycle)	13
3.1.4 Narzędzia i komunikacja	13
3.1.5 Podział ról w zespole	14
3.2 Harmonogram projektu	14
3.3 Technologie i narzędzia	15

3.3.1	Technologie	15
3.3.2	Narzędzia	24
3.4	Zasoby i ograniczenia	27
3.4.1	Zasoby	27
3.4.2	Ograniczenia	27
3.4.3	Usługi zewnętrzne	27
3.5	Analiza ryzyka	30
4	Analiza wymagań	31
4.1	Przypadki użycia	32
4.1.1	Aktorzy	32
4.1.2	Diagramy przypadków użycia	34
4.1.3	Scenariusze przypadków użycia	41
4.1.3.1	Scenariusze przypadków użycia – funkcje ogólne .	41
4.1.3.2	Scenariusze przypadków użycia dla funkcji premium	48
4.1.3.3	Scenariusze przypadków użycia dla wyszukiwarki .	51
4.1.3.4	Scenariusze przypadków użycia dla mapy	53
4.1.3.5	Scenariusze przypadków użycia dla czatu	57
4.1.3.6	Scenariusze przypadków użycia dla forum	65
4.1.3.7	Scenariusze przypadków użycia dla panelu użytkownika	73
4.2	Wymagania	82
4.2.1	Wymagania ogólne	83
4.2.1.1	Wymagania ogólne dla czatu	83
4.2.2	Wymagania funkcjonalne	85
4.2.2.1	Funkcjonalności dla mapy	85
4.2.2.2	Wymagania funkcjonalne dla czatu	85
4.2.2.3	Funkcjonalności dla forum	95
4.2.2.4	Funkcjonalności dla konta użytkownika	95
4.2.2.5	Funkcjonalności dla logowania i rejestracji	105
4.2.2.6	Funkcjonalności dla wyszukiwarki spotów	106
4.2.2.7	Funkcjonalności dla motywu	108

4.2.3	Wymagania pozafunkcjonalne	110
4.2.3.1	Wymagania pozafunkcjonalne dla czatu	110
5	Projekt	116
5.1	Wzorce projektowe	116
5.2	Architektura systemu	116
5.2.1	Diagram architektury	117
5.2.2	Komponenty systemu	119
5.3	Projekt bazy danych	120
5.3.1	Model danych	120
5.3.2	Diagram ERD	120
5.4	Architektura interfejsu użytkownika	120
5.4.1	Projekt strony głównej	120
5.4.2	Projekt panelu logowania	124
5.4.3	Projekt mapy	126
5.4.4	Projekt chatu	126
5.4.5	Projekt forum	126
5.4.6	Projekt panelu użytkownika	126
5.4.6.1	Profile	126
5.4.6.2	Spots list	132
5.4.6.3	Photos list	133
5.4.6.4	Movies list	135
5.4.6.5	Friends	136
5.4.6.6	Add spot	137
5.4.6.7	Comments	140
5.4.6.8	Settings	140
6	Przebieg realizacji projektu	143
6.1	Faza przedprojektowa (lipiec–wrzesień 2024)	143
6.2	Etap 1 (październik 2024 – styczeń 2025)	146
6.3	Etap 2 (luty 2025 – wrzesień 2025)	150
6.4	Etap 3 (październik 2025 – styczeń 2026)	159

7 Realizacja Projektu	162
7.1 Implementacja backendu	162
7.1.1 Struktura projektu	162
7.1.2 Endpointy systemu	165
7.1.3 Integracja z bazą danych	211
7.1.4 Obsługa uwierzytelnienia	214
7.1.5 Konteneryzacja	214
7.2 Implementacja frontendu	215
7.2.1 Struktura aplikacji	216
7.2.2 Zarządzanie stanem i przepływ danych	221
7.2.3 Integracja i komunikacja z backendem	224
7.2.4 Style	227
7.2.5 Wyszukiwarka spotów	231
7.2.6 Mapa	238
7.2.7 Chat	238
7.2.8 Forum	238
7.2.9 Panel użytkownika	238
7.2.9.1 Profile	239
7.2.9.2 Spots	243
7.2.9.3 Photos	246
7.2.9.4 Movies	250
7.2.9.5 Social	251
7.2.9.6 Add spot	258
7.2.9.7 Comments	262
7.2.9.8 Settings	264
7.2.9.9 Komponenty wspólne	266
7.2.10 Panel logowania	271
7.2.10.1 Logowanie	272
7.2.10.2 Rejestracja	274
7.2.10.3 Reset hasła	277
7.2.10.4 Nowe hasło	280

7.2.10.5	Komponenty wspólne panelu logowania	283
7.3	Implementacja CI/CD	288
7.4	Implementacja WebSocket	288
7.4.1	Charakterystyka protokołu WebSocket	289
7.4.2	Zastosowanie WebSocket w naszym projekcie	290
7.4.3	Implementacja na backendzie	291
7.4.3.1	Konfiguracja WebSocket	291
7.4.3.2	Kontroler STOMP	293
7.4.3.3	Serwis dystrybucji wiadomości	293
7.4.3.4	Zestawienie wykorzystywanych destynacji	295
7.4.4	Implementacja na frontendzie	295
7.4.4.1	Serwis komunikacji WebSocket	297
7.4.4.2	Mechanizm rejestracji subskrypcji	303
7.4.4.3	Kontekst i zarządzanie połączeniem	305
7.4.4.4	Definicja subskrypcji czatu	307
7.4.5	Przebieg komunikacji	308
8	Testy	309
8.1	Testy jednostkowe	309
8.2	Testy integracyjne	309
8.3	Testy E2E	309
8.4	Wyniki testów i wnioski	309
9	Prezentacja systemu	310
9.1	Strona główna	310
9.2	Strona mapy	310
9.3	Strona chatu	310
9.4	Strona forum	310
9.5	Panel logowania	310
9.6	Panel konta użytkownika	310
10	Nakład pracy	311
10.1	Ogólny nakład pracy	311

10.2 Indywidualne nakłady pracy	311
10.2.1 Adam Langmesser	311
10.2.2 Mateusz Redosz	311
10.2.3 Stanisław Oziemczuk	314
10.2.4 Kacper Badek	314
11 Podsumowanie	315
11.1 Osiągnięte rezultaty	315
11.2 Napotkane wyzwania	315
11.3 Plany na przyszłość	315
12 Słownik pojęć i skrótów	316
Spis tabel	337
Bibliografia	344
Załączniki	346

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

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 wyma-

gano możliwości częstych rewizji założeń oraz wprowadzania istotnych zmian w docelowej wizji rozwiązania; dlatego z podejścia zrezygnowano.

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

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

Kluczowe argumenty wyboru:

- **Brak sprintów.** Zastosowano przepływ ciągły, co pozwala dopasować tempo do zmiennej dostępności zespołu i unikać sztucznego „domykania” iteracji.
- **Rozbudowana faza startowa.** Na początku przewidziano większy wysiłek planistyczny: doprecyzowanie zakresu, wstępna wizja architektury, identyfikacja ryzyk, plan publikacji oraz kryteria jakości – bez zamrażania szczegółów.
- **Ciągła weryfikacja wymagań.** W trakcie realizacji przewidziano bieżące doprecyzowywanie backlogu, regularny feedback promotoru 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 **Discordu** oraz **Messengera**.

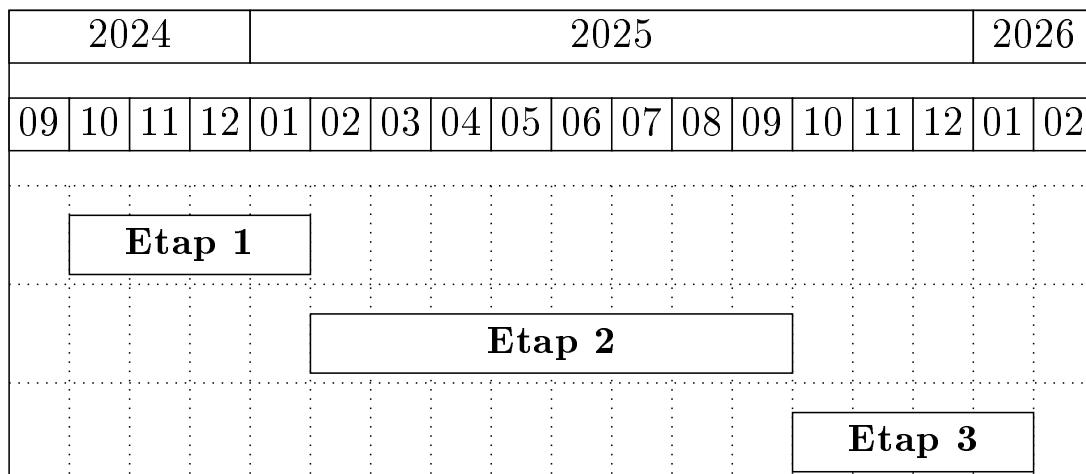
3.1.5 Podział ról w zespole

- 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.

Biblioteka	Opis
junit-jupiter	Integracja biblioteki Testcontainers z frameworkiem testowym JUnit ułatwiająca pisanie testów integracyjnych z użyciem kontenerów.
lombok	Biblioteka ułatwiająca generowanie kodu (getters, setters, buildery, konstruktor itp.) przez adnotacje zmniejszające ilość boilerplate'u w kodzie.
postgresql	Umożliwia połaczenie i komunikację z bazą danych PostgreSQL.
shedlock-spring	Zarządzanie zadaniami okresowymi (cron/scheduled).
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.

Biblioteka	Opis
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.
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 react-hook-form – ułatwia integrację z bibliotekami walidacji (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.

Biblioteka / plugin	Opis / przeznaczenie
@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.
@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 multimediiów w przeglądarce.
motion	Narzędzia służące do animacji interfejsu użytkownika.
query-string	Narzędzia do parsowania i generowania parametrów zapytań w URL.

Biblioteka / plugin	Opis / przeznaczenie
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.
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.

Biblioteka / plugin	Opis / przeznaczenie
@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 react-dom.
@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.
@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.

Biblioteka / plugin	Opis / przeznaczenie
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 nieustrukturyzowanych, 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 [BPMN](#)([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

4.1.1 Aktorzy

Niniejszy rozdział przedstawia aktorów wraz z opisami.

Użytkownik systemu - Reprezentuje każdą osobę korzystającą z aplikacji.

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

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

Użytkownik premium - Użytkownik z wykupioną subskrypcją: ma dostęp do funkcji premium np. oznaczenie stref **PANSA** na mapie.

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

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

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

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

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

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

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

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

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



Rysunek 4.1: Diagram hierarchii użytkowników systemu

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

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

4.1.2 Diagramy przypadków użycia

Niniejszy rozdział przedstawia diagramy przypadków użycia.



Rysunek 4.2: Wysokopoziomowy diagram przypadków użycia

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



Rysunek 4.3: Diagram przypadków użycia dla użytkownika niezalogowanego



Rysunek 4.4: Diagram przypadków użycia dla użytkownika zalogowanego



Rysunek 4.5: Diagram przypadków użycia dla użytkownika premium



Rysunek 4.6: Diagram przypadków użycia wyszukiwarki spotów oraz mapy



Rysunek 4.7: Diagram przypadków użycia forum



Rysunek 4.8: Diagram przypadków użycia czatu



Rysunek 4.9: Diagram przypadków użycia profilu użytkownika

4.1.3 Scenariusze przypadków użycia

4.1.3.1 Scenariusze przypadków użycia – funkcje ogólne

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU1
Nazwa:	Rejestracja użytkownika
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany
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:	Wysoki
Aktorzy:	Użytkownik niezalogowany
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:	Wykupienie subskrypcji premium
Priorytet:	Niski
Aktorzy:	Użytkownik zalogowany, Bramka płatnicza, System finansowo-księgowy
Opis:	Użytkownik opłaca subskrypcję premium w celu uzyskania dodatkowych funkcji.
Warunki wstępne:	Użytkownik jest zalogowany i znajduje się w module subskrypcji.
Warunki końcowe:	Subskrypcja premium jest aktywna, a użytkownik ma dostęp do funkcji premium.

Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik wybiera plan subskrypcji. 2. Użytkownik przechodzi do bramki płatniczej. 3. Użytkownik podaje dane płatnicze i zatwierdza transakcję. 4. Bramka płatnicza przetwarza płatność i zwraca wynik do systemu. 5. System zapisuje informację o opłaconej subskrypcji i aktualizuje uprawnienia. 6. System generuje wpis w systemie finansowo-księgowym.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 4a. Płatność nie powiodła się – system informuje użytkownika i umożliwia ponowną próbę. 5a. W czasie aktualizacji subskrypcji wystąpił błąd – system cofa zmiany i wyświetla komunikat o problemie.

Tabela 4.3: Scenariusz przypadku użycia: Wykupienie subskrypcji premium

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU4
Nazwa:	Resetowanie hasła
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany, 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:	<ol style="list-style-type: none"> 2a. Nie istnieje konto dla podanego adresu – system wyświetla komunikat o błędzie. 4a. Występuje błąd połączenia z usługą SMTP – system informuje użytkownika o problemie technicznym.

Tabela 4.4: Scenariusz przypadku użycia: Resetowanie hasła

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU5
Nazwa:	Zmiana hasła w ustawieniach konta
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany
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"> 1. Użytkownik wpisuje aktualne hasło. 2. Użytkownik wpisuje nowe hasło i powtarza je. 3. Użytkownik zatwierdza formularz zmiany hasła. 4. System zapisuje nowe hasło i informuje o powodzeniu operacji.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 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 4.5: Scenariusz przypadku użycia: Zmiana hasła w ustawieniach konta

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU6
Nazwa:	Wylogowanie użytkownika
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany
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:	<ol style="list-style-type: none"> 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 4.6: Scenariusz przypadku użycia: Wylogowanie użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU7
Nazwa:	Przeglądanie powiadomień
Priorytet:	Niski
Aktorzy:	Użytkownik zalogowany
Opis:	Użytkownik przegląda listę powiadomień.
Warunki wstępne:	Użytkownik jest na ekranie centra powiadomień.
Warunki końcowe:	Powiadomienia zostały wyświetcone, 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 4.7: Scenariusz przypadku użycia: Przeglądanie powiadomień

4.1.3.2 Scenariusze przypadków użycia dla funkcji premium

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU8
Nazwa:	Przeszukiwanie historii czatu
Priorytet:	Niski
Aktorzy:	Użytkownik premium
Opis:	Użytkownik wyszukuje konkretne wiadomości w historii czatu.
Warunki wstępne:	Użytkownik jest zalogowany jako użytkownik premium i znajduje się w widoku czatu.
Warunki końcowe:	Wiadomości spełniające kryteria wyszukiwania zostały wyświetlane.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik otwiera pole wyszukiwania historii w czacie. Użytkownik wpisuje frazę lub filtr (np. zakres dat, autor). System filtryuje wiadomości zgodnie z kryteriami. System prezentuje listę dopasowanych fragmentów rozmowy.
Alternatywne przepływy zdarzeń:	4a. Nie znaleziono wiadomości spełniających kryteria – system informuje o braku wyników wyszukwiania.

Tabela 4.8: Scenariusz przypadku użycia: Przeszukiwanie historii czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU9
Nazwa:	Przeglądanie wysłanych plików na czacie

Priorytet:	Niski
Aktorzy:	Użytkownik premium, Usługa do przechowywania plików w chmurze
Opis:	Użytkownik przegląda listę plików wysłanych w ramach czatów.
Warunki wstępne:	Użytkownik jest zalogowany jako użytkownik premium.
Warunki końcowe:	Użytkownik widzi listę wysłanych plików i może przechodzić do powiązanych czatów.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> 1. Użytkownik otwiera sekcję „Wysłane pliki”. 2. System pobiera metadane plików z usługi przechowywania. 3. System wyświetla listę plików z podstawowymi informacjami (nazwa, typ, data). 4. Użytkownik wybiera plik, aby otworzyć go lub przejść do powiązanego czatu.
Alternatywne przepływy zdarzeń:	<p>4a. Doszło do błędu podczas pobierania pliku - system wyświetla odpowiedni komunikat.</p>

Tabela 4.9: Scenariusz przypadku użycia: Przeglądanie wysłanych plików na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU10
Nazwa:	Zmiana typu mapy
Priorytet:	Niski

Aktorzy:	Użytkownik premium, Usługa do wyświetlania mapy
Opis:	Użytkownik zmienia typ mapy (np. standardowa, satelitarna, hybrydowa).
Warunki wstępne:	Użytkownik premium 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:	<p>3a. Wybrany typ mapy nie jest dostępny (błąd usługi mapowej) – system przywraca poprzedni typ i informuje o błędzie.</p>

Tabela 4.10: Scenariusz przypadku użycia: Zmiana typu mapy

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU11
Nazwa:	Przeglądanie stref PANSA
Priorytet:	Niski
Aktorzy:	Użytkownik premium, Usługa do wyświetlania mapy
Opis:	Użytkownik wyświetla na mapie strefy przestrzeni powietrznej PANSA.
Warunki wstępne:	Użytkownik premium 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:	<ol style="list-style-type: none"> <li value="2">2a. Dane o strefach są chwilowo niedostępne – system komunikuje problem i nie włącza warstwy.

Tabela 4.11: Scenariusz przypadku użycia: Przeglądanie stref PANSA

4.1.3.3 Scenariusze przypadków użycia dla wyszukiwarki

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU12
Nazwa:	Wyszukiwanie spota w globalnej wyszukiwarce
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany, Usługa do wyświetlania mapy, Usługa do pogody
Opis:	Użytkownik wyszukuje spedy 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 spedy spełniające kryteria. 3. System wyświetla listę wyników.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 3a. System informuje o braku wyników spełniających kryteria.

Tabela 4.12: Scenariusz przypadku użycia: Wyszukiwanie spota w globalnej wyszukiwarce

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU13
Nazwa:	Przejście do spota na mapie z wyszukiwarki
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany
Opis:	Użytkownik przechodzi z wyników wyszukiwarki do widoku mapy ustawionego 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 4.13: Scenariusz przypadku użycia: Przejście do spota na mapie z wyszukiwarki

4.1.3.4 Scenariusze przypadków użycia dla mapy

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU14
Nazwa:	Przeglądanie mapy spotów
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany, 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:	<ol style="list-style-type: none"> System inicjuje widok mapy z domyślnym obszarem. System pobiera listę spotów. System rysuje znaczniki spotów na mapie.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> a. Usługa mapy jest niedostępna – system wyświetla komunikat o błędzie.

Tabela 4.14: Scenariusz przypadku użycia: Przeglądanie mapy spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU15
Nazwa:	Otwarcie szczegółów spota
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany, Użytkownik zalogowany
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"> 1. Użytkownik wybiera spota z mapy. 2. System pobiera dane szczegółowe spota (informacje opisowe, lokalizacja). 3. System otwiera widok szczegółów spota.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> 2a. Spot nie istnieje (został usunięty lub ukryty) – system informuje użytkownika i powraca do poprzedniego widoku. 2b. Wystąpił błąd podczas pobierania danych spota – system wyświetla komunikat o błędzie i umożliwia ponowną próbę.

Tabela 4.15: Scenariusz przypadku użycia: Otwarcie szczegółów spota

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU16
Nazwa:	Przeglądanie komentarzy do spota
Priorytet:	Średni
Aktorzy:	Użytkownik niezalogowany

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"> 1. System pobiera komentarze powiązane ze spotem. 2. System wyświetla komentarze w kolejności chronologicznej lub według popularności. 3. Użytkownik przewija listę komentarzy.
Alternatywne przepływy zdarzeń:	1a. Spot nie ma jeszcze komentarzy – system wyświetla odpowiednią informację.

Tabela 4.16: Scenariusz przypadku użycia: Przeglądanie komentarzy do spota

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU17
Nazwa:	Przeglądanie pogody na spocie
Priorytet:	Średni
Aktorzy:	Użytkownik niezalogowany, 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"> 1. Użytkownik otwiera zakładkę pogody. 2. System wysyła zapytanie do usługi pogodowej z lokalizacją spota. 3. System odbiera prognozę i prezentuje ją (temperatura, prędkość wiatru, opady).
Alternatywne przepływy zdarzeń:	<p>2a. Usługa pogodowa jest niedostępna – system wyświetla komunikat o braku danych pogodowych.</p>

Tabela 4.17: Scenariusz przypadku użycia: Przeglądanie pogody na spocie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU18
Nazwa:	Wyszukiwanie spota na mapie
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany
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:	2a. Brak wyników dla podanej frazy – system informuje użytkownika o braku dopasowań.
--	--

Tabela 4.18: Scenariusz przypadku użycia: Wyszukiwanie spota na mapie

4.1.3.5 Scenariusze przypadków użycia dla czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU19
Nazwa:	Utworzenie prywatnego czatu
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany
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 4.19: Scenariusz przypadku użycia: Utworzenie prywatnego czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU20
Nazwa:	Otworzenie czatu
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany
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 4.20: Scenariusz przypadku użycia: Otworzenie czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU21
Nazwa:	Utworzenie czatu grupowego
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany
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 4.21: Scenariusz przypadku użycia: Utworzenie czatu grupowego

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU22
Nazwa:	Przeglądanie listy czatów
Priorytet:	Wysoki

Aktorzy:	Użytkownik zalogowany
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 4.22: Scenariusz przypadku użycia: Przeglądanie listy czatów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU23
Nazwa:	Wysyłanie wiadomości na czacie
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany
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 4.23: Scenariusz przypadku użycia: Wysyłanie wiadomości na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU24
Nazwa:	Wysyłanie GIF-a na czacie
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany, 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 4.24: Scenariusz przypadku użycia: Wysyłanie GIF-a na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU25
Nazwa:	Wysyłanie pliku na czacie
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany, 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:	<p>3a. Przesyłanie pliku nie powiodło się – system informuje użytkownika i umożliwia ponowną próbę.</p>

Tabela 4.25: Scenariusz przypadku użycia: Wysyłanie pliku na czacie

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU26
Nazwa:	Edycja ustawień czatu
Priorytet:	Niski
Aktorzy:	Użytkownik zalogowany
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"> 1. Użytkownik otwiera panel ustawień czatu. 2. Użytkownik wprowadza zmiany (np. nazwę, opis, avatar). 3. Użytkownik zapisuje zmiany. 4. System waliduje dane i aktualizuje konfigurację czatu.

Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów poza walidacja pól.
--	--

Tabela 4.26: Scenariusz przypadku użycia: Edycja ustawień czatu

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU27
Nazwa:	Dodanie członka do czatu grupowego
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany
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:	<ol style="list-style-type: none"> Operacja nie powiodła się – system informuje o błędzie.

Tabela 4.27: Scenariusz przypadku użycia: Dodanie członka do czatu grupowego

4.1.3.6 Scenariusze przypadków użycia dla forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU28
Nazwa:	Przeglądanie postów na forum
Priorytet:	Wysoki
Aktorzy:	Użytkownik niezalogowany, Użytkownik zalogowany
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"> Użytkownik przechodzi do widoku listy postów forum. System pobiera listę postów i domyślnie wyświetla je w kolejności od najnowszych. Użytkownik wybiera sposób sortowania listy. System aktualizuje listę postów zgodnie z wybranym kryterium sortowania. Użytkownik wybiera post, który chce przeczytać. System otwiera szczegółowy widok wybranego posta.
Alternatywne przepływy zdarzeń:	<ol style="list-style-type: none"> System nie może pobrać szczegółów posta – system wyświetla komunikat o błędzie.

Tabela 4.28: Scenariusz przypadku użycia: Przeglądanie postów na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA

Identyfikator:	PU29
Nazwa:	Wyszukiwanie postów na forum
Priorytet:	Średni
Aktorzy:	Użytkownik niezalogowany, Użytkownik zalogowany
Opis:	Użytkownik wyszukuje posty na forum na podstawie tytułu, kategorii, tagów oraz autora.
Warunki wstępne:	Użytkownik znajduje się w module forum lub na stronie wyszukiwarki postów.
Warunki końcowe:	Lista postów spełniających zadane kryteria wyszukiwania jest wyświetlona.
Główny przepływ zdarzeń:	<ol style="list-style-type: none"> Użytkownik otwiera panel wyszukiwania postów na forum. Użytkownik określa kryteria wyszukiwania (np. tytuł, kategoria, tagi, autor). Użytkownik uruchamia wyszukiwanie. System filtryuje posty zgodnie z podanymi kryteriami i wyświetla listę wyników.
Alternatywne przepływy zdarzeń:	<p>4a. Brak postów spełniających zadane kryteria – system wyświetla informację o braku wyników.</p> <p>4b. Wystąpił błąd podczas wyszukiwania – system wyświetla komunikat o błędzie i umożliwia ponowną próbę.</p>

Tabela 4.29: Scenariusz przypadku użycia: Wyszukiwanie postów na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU30
Nazwa:	Dodanie posta na forum

Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany, 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 4.30: Scenariusz przypadku użycia: Dodanie posta na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU31
Nazwa:	Dodanie komentarza na forum
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany
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"> Użytkownik wpisuje treść komentarza w formularzu pod postem. (Opcjonalnie) Użytkownik dodaje załączniki (zdjęcia/filmy) do komentarza. Użytkownik publikuje komentarz. System zapisuje komentarz (oraz poprawne załączniki) i odświeża listę komentarzy.
Alternatywne przepływy zdarzeń:	<p>3a. Treść komentarza lub załączniki są niepoprawne (np. naruszają walidację) – system wyświetla komunikat o błędzie i blokuje publikację do czasu poprawy danych.</p>

Tabela 4.31: Scenariusz przypadku użycia: Dodanie komentarza na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU32
Nazwa:	Przeglądanie historii interakcji z postami

Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany
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 4.32: Scenariusz przypadku użycia: Przeglądanie historii interakcji z postami

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU33
Nazwa:	Zarządzanie komentarzami na forum
Priorytet:	Niski
Aktorzy:	Użytkownik zalogowany
Opis:	Użytkownik zarządza komentarzami pod postami forum (edytacja, 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 4.33: Scenariusz przypadku użycia: Zarządzanie komentarzami na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU34
Nazwa:	Zgłoszenie komentarza naruszającego regulamin
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany

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 4.34: Scenariusz przypadku użycia: Zgłoszenie komentarza naruszającego regulamin

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU35
Nazwa:	Zgłoszenie posta na forum
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany

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 4.35: Scenariusz przypadku użycia: Zgłoszenie posta na forum

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU36
Nazwa:	Przeglądanie komentarzy pod postem
Priorytet:	Średni
Aktorzy:	Użytkownik niezalogowany, Użytkownik zalogowany
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 4.36: Scenariusz przypadku użycia: Przeglądanie komentarzy pod postem

4.1.3.7 Scenariusze przypadków użycia dla panelu użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU37
Nazwa:	Dodanie spota w panelu użytkownika
Priorytet:	Wysoki

Aktorzy:	Użytkownik zalogowany, 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 specie (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:	3a. Podane dane wejściowe są niepoprawne – system wyświetla komunikat i zaznacza wymagające poprawy pola.

Tabela 4.37: Scenariusz przypadku użycia: Dodanie spota w panelu użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU38
Nazwa:	Przeglądanie profilu użytkownika
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany

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:	<p>2a. Wystąpił błąd podczas pobierania danych użytkownika – system wyświetla informację o błędzie.</p>

Tabela 4.38: Scenariusz przypadku użycia: Przeglądanie profilu użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU39
Nazwa:	Przeglądanie profilu innego użytkownika
Priorytet:	Średni
Aktorzy:	Użytkownik niezalogowany
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:	<ol style="list-style-type: none"> <li value="2">2a. Wystąpił błąd podczas pobierania danych użytkownika – system wyświetla informację o błędzie.

Tabela 4.39: Scenariusz przypadku użycia: Przeglądanie profilu innego użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU40
Nazwa:	Dodanie użytkownika do znajomych
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany
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"> 1. Użytkownik kliką przycisk „Dodaj do znajomych”. 2. System sprawdza, czy relacja już istnieje. 3. System tworzy nowe zaproszenie. 4. System informuje o statusie o wysłaniu zaproszenia.

Alternatywne przepływy zdarzeń:	Brak istotnych alternatywnych przepływów.
--	---

Tabela 4.40: Scenariusz przypadku użycia: Dodanie użytkownika do znajomych

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU41
Nazwa:	Przeglądanie społeczności
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany
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:	<ol style="list-style-type: none"> Nie udało się pobrać danych – system wyświetla komunikat o błędzie.

Tabela 4.41: Scenariusz przypadku użycia: Przeglądanie społeczności

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU42
Nazwa:	Przeglądanie dodanych spotów
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany, 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 spedy”.
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"> Użytkownik przechodzi do sekcji „Moje spedy”. System pobiera listę spotów dodanych przez użytkownika. System wyświetla listę spotów oraz znaczniki na mapie. Użytkownik wybiera spota, aby przejść do jego szczegółów.
Alternatywne przepływy zdarzeń:	<p>2a. Użytkownik nie dodał jeszcze żadnego spota – system wyświetla komunikat i proponuje dodanie pierwszego spota.</p>

Tabela 4.42: Scenariusz przypadku użycia: Przeglądanie dodanych spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU43
Nazwa:	Edycja danych użytkownika
Priorytet:	Wysoki
Aktorzy:	Użytkownik zalogowany

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:	<p>4a. Dane są niepoprawne lub niekompletne – system wyświetla komunikat o błędzie i zaznacza pola do poprawy.</p>

Tabela 4.43: Scenariusz przypadku użycia: Edycja danych użytkownika

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU44
Nazwa:	Przeglądanie dodanych zdjęć do spotów
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany, 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 4.44: Scenariusz przypadku użycia: Przeglądanie dodanych zdjęć do spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU45
Nazwa:	Przeglądanie dodanych filmów do spotów
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany, 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 4.45: Scenariusz przypadku użycia: Przeglądanie dodanych filmów do spotów

KARTA SCENARIUSZA PRZYPADKU UŻYCIA	
Identyfikator:	PU46
Nazwa:	Przeglądanie dodanych komentarzy do spotów
Priorytet:	Średni
Aktorzy:	Użytkownik zalogowany
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 4.46: 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 przetwarzanie 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 4.47: 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 4.48: 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 4.49: 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 4.50: Karta wymagania ogólnego dla czatu: Tworzenie czatu

4.2.2 Wymagania funkcjonalne

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

4.2.2.1 Funkcjonalności dla 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 4.51: 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 4.52: 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 4.53: 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 4.54: 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 4.55: 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 4.56: 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 4.57: 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 4.58: 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 4.59: 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 4.60: 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 4.61: 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 górę automatycznie pobiera starsze wiadomości (mechanizm Infinite scroll). • Dociąganie wiadomości nie powoduje zauważalnych opóźnień interfejsu.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-03.

Tabela 4.62: 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 4.63: 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 4.64: 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 4.65: 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 4.66: 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 4.67: 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 4.68: 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 4.69: 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 4.70: 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 4.71: 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 4.72: 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 4.73: 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 4.74: 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 4.75: 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 4.76: Logowanie i rejestracja

4.2.2.6 Funkcjonalności dla wyszukiwarki spotów

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

Tabela 4.77: Strona główna — podstawowe filtry

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

Tabela 4.78: Strona główna — zaawansowane filtry

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 4.79: 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 4.80: Zapamiętanie preferencji motywu

KARTA WYMAGANIA		
Identyfikator:	FOXX	Priorytet: S
Nazwa:	Przełącznik motywów 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 motywów.	
Warunki początkowe:	FOXX, FOXX dostępne.	
Warunki końcowe:	Motyw zmieniony; preferencja zaktualizowana.	
Sytuacje wyjątkowe:	Brak.	
Szczegóły implementacji:	Przycisk typu <i>toggle</i> wywołuje funkcję, która przełącza klasę <code>dark</code> na <code>document.documentElement</code> oraz aktualizuje <code>localStorage (theme = 'dark' 'light')</code> ; brak przeładowania strony.	
Udziałowiec:	Zespół projektowy 2.1 , promotor 2.2 , droniarze 2.3 .	
Wymagania powiązane:		

Tabela 4.81: Szybki przełącznik motywów 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 4.63: 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 odrzucane kodem 401 i nie zwracają żadnych danych.
Udziałowiec:	U3
Realizator:	Adam Langmesser
Wymagania powiązane:	WOCZAT-01 , WOCZAT-02 , WOCZAT-03 , WOCZAT-04 .

Tabela 4.64: 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 4.65: 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 4.66: 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 4.67: 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 4.68: Wymaganie pozafunkcjonalne dla czatu: Natychmiastowe wysyłanie wiadomości

Rozdział 5

Projekt

5.1 Wzorce projektowe

5.2 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](#).

5.2.1 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

5.2.2 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](#)

5.3 Projekt bazy danych

5.3.1 Model danych

5.3.2 Diagram ERD

5.4 Architektura interfejsu użytkownika

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

5.4.1 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.

5.4.2 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

5.4.3 Projekt mapy

5.4.4 Projekt chatu

5.4.5 Projekt forum

5.4.6 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).

5.4.6.1 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.

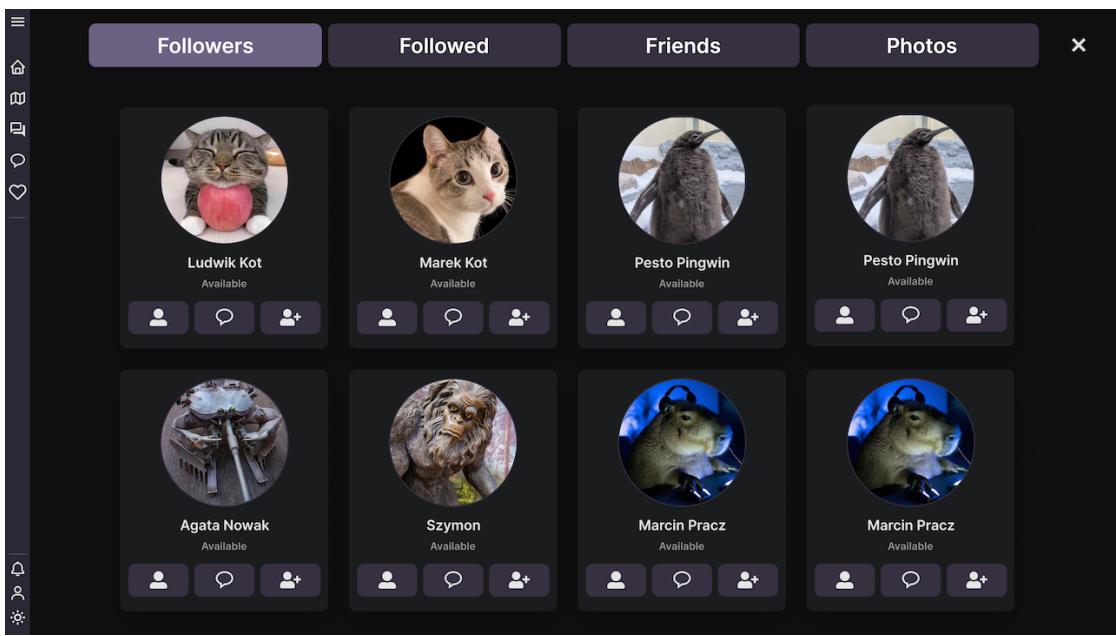
Analogiczne działanie przewidziano dla profilu własnego użytkownika. Po wybraniu statystyk związanych ze znajomymi lub zdjęciami następuje przejście odpowiednio do list *Friends* (rys. 5.20) oraz *Photos* (rys. 7.28).



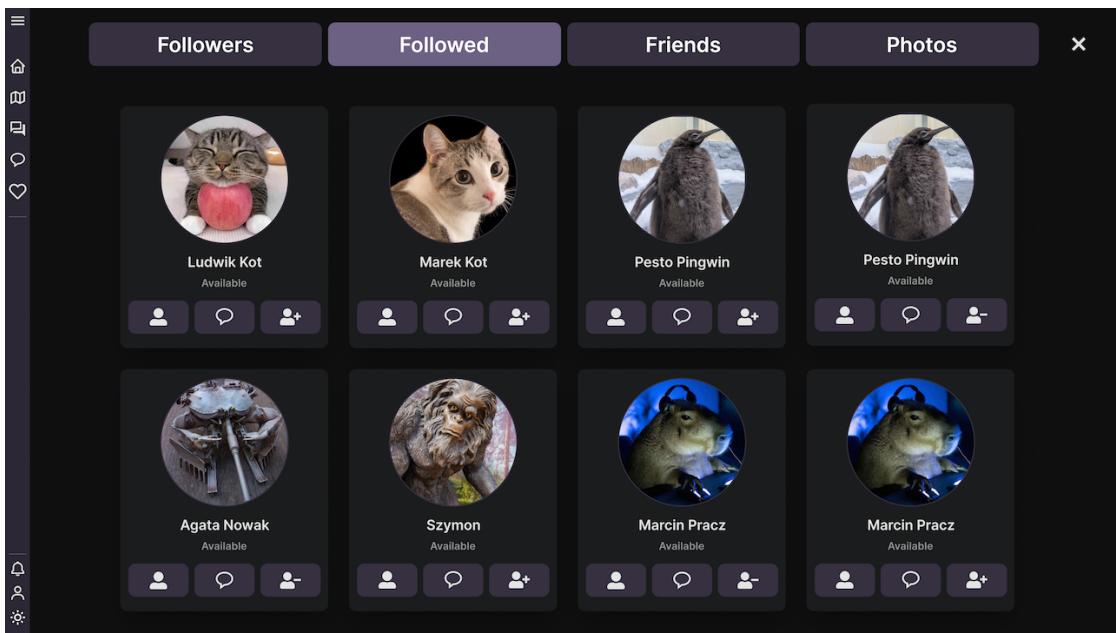
Rysunek 5.7: Widok prywatny profilu



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

5.4.6.2 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**

5.4.6.3 Photos list

Kolejnym widokiem jest lista zdjęć (rys. 7.28), 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

5.4.6.4 Movies list

Kolejnym widokiem jest lista filmów (rys. 7.30), 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.



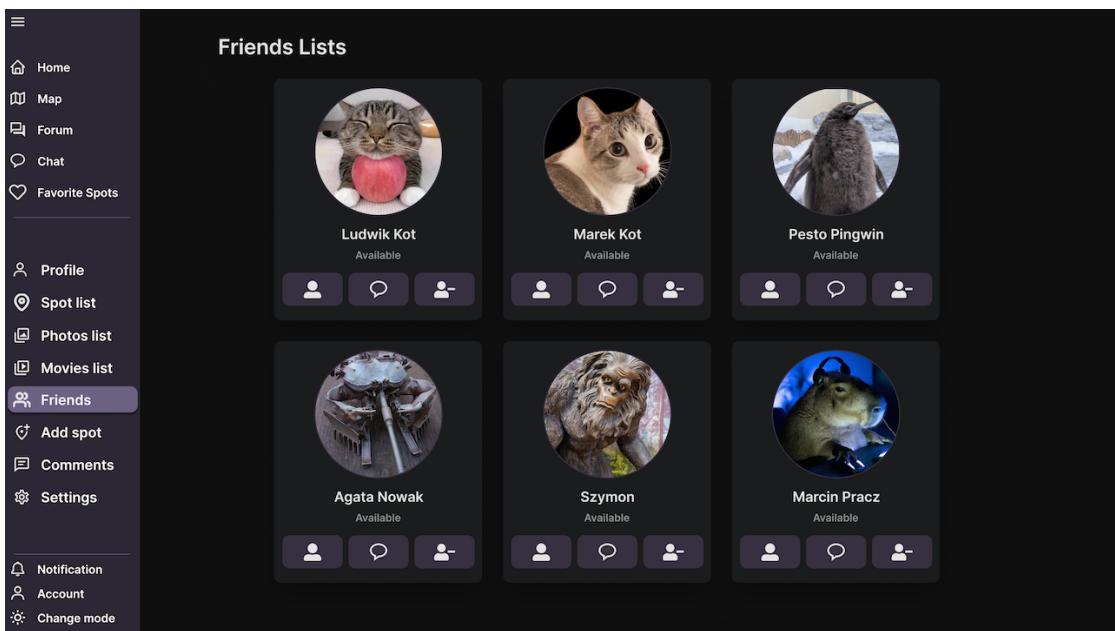
Rysunek 5.19: Projekt strony z listą dodanych filmów

5.4.6.5 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

5.4.6.6 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

5.4.6.7 Comments

Podobnie jak w pozostałych widokach, w górnej części strony umieszczono nagłówek z nazwą sekcji (rys. 7.35). 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

5.4.6.8 Settings

Ostatnią stroną panelu użytkownika są ustawienia (rys. 7.36). W górnej części widoku umieszczono nagłówek informujący, w jakiej sekcji aplikacji aktualnie znaj-

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

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

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

6.1 Faza przedprojektowa (lipiec–wrzesień 2024)

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

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

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

Lipiec 2024

Cały zespół

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

Adam Langmesser

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

Sierpień 2024

Cały zespół

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

Mateusz Redosz

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

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

Adam Langmesser

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

Kacper Badek

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

Stanisław Oziemczuk

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

Wrzesień 2024

Adam Langmesser

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

Mateusz Redosz

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

Kacper Badek

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

6.2 Etap 1 (październik 2024 – styczeń 2025)

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

Październik 2024

Cały zespół

- Formalny wybór tematu projektu inżynierskiego.

Adam Langmesser

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

Mateusz Redosz

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

Stanisław Oziemczuk

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

Kacper Badek

- Uporządkowanie pliku .gitignore oraz struktury repozytorium.

Listopad 2024

Adam Langmesser

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

Mateusz Redosz

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

Stanisław Oziemczuk

- Poprawa logiki logowania użytkownika po stronie backendu i frontendu.

Kacper Badek

- Implementacja logiki cyklicznego usuwania przeterminowanych tokenów resetu hasła.

Cały zespół

- Opracowanie harmonogramu projektu w formie opisowej oraz w postaci diagramu Gantta.
- Przygotowanie wstępnych założeń i wymagań w ramach przedmiotu PRO.

Grudzień 2024

Adam Langmesser

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

Mateusz Redosz

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

Stanisław Oziemczuk

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

Kacper Badek

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

Styczeń 2025

Adam Langmesser

- Poprawa testów logowania i rejestracji na [backendzie](#) oraz ujednolicenie assertji.
- Rozwiążanie problemu z relacjami między obiektami w testach integracyjnych.

Mateusz Redosz

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

Stanisław Oziemczuk

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

Kacper Badek

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

6.3 Etap 2 (luty 2025 – wrzesień 2025)

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

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

Luty 2025

Mateusz Redosz

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

Stanisław Oziemczuk

- Implementacja logiki dodawania [spota](#) do ulubionych po stronie [frontendu](#) oraz integracja z [backendem](#).
- Refaktoryzacja kodu na [frontendzie](#).

- Obsługa błędów przy wysyłaniu maili.
- Zmiana dostawy usługi do wysyłania maili.
- Optymalizacja zapytań do bazy danych.
- Testy integracyjne logiki logowania i rejestracji OAuth na backendzie oraz filtrowania spotów.
- Poprawa plików Dockerfile oraz Docker Compose.
- Przygotowanie skryptu uruchamiającego wszystkie wymagane kontenery Docker (bazy danych i usługi pomocnicze) na potrzeby środowiska deweloperskiego, co uprościło proces uruchamiania projektu na nowych stanowiskach.

Adam Langmesser

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

Marzec 2025

Cały Zespół

- Wypracowanie wstępnej wersji wyglądu interfejsu użytkownika w ramach PRZ 1.

Kacper Badek

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

Stanisław Oziemczuk

- Przygotowanie instrukcji uruchamiania projektu w README.

Mateusz Redosz

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

Kwiecień 2025

Cały Zespół

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

Adam Langmesser

- Konfiguracja projektu dokumentacji w [LaTeX](#), utworzenie pliku bibliografii oraz głównego dokumentu z przykładową strukturą rozdziałów.
- Implementacja podstaw funkcjonalności czatu w warstwie [backendowej](#) i [fronthandowej](#), a także dostosowanie layoutu i paska bocznego do nowej sekcji.

Mateusz Redosz

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

Maj 2025

Stanisław Oziemczuk

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

Mateusz Redosz

- Refactor panelu bocznego, dodanie ogólnych hooków do zarządzania stanem (w tym trybem ciemnym) oraz wprowadzenie zależności do biblioteki [motion](#) na potrzeby animacji.
- Dodanie sekcji „*Znajomi*” w panelu konta, obejmującej obsługę znajomych, obserwowanych i obserwujących, wprowadzenie nowych modeli i [endpointów](#) do zarządzania relacjami oraz dodanie testów jednostkowych.
- Refaktoryzacja [endpointów](#) użytkownika tak, aby login był pobierany z kontekstu uwierzytelnienia zamiast z parametru ścieżki, dostosowanie do tego wywołań na froncie oraz przeniesienie [slice Redux](#) odpowiedzialnego za konto użytkownika na [TypeScript](#) z uproszczonym stanem i uporządkowanymi importami.
- Poprawa wyglądu formularzy logowania i rejestracji.
- Poprawa zarządzania stanem [sidebara](#).

Adam Langmesser

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

Kacper Badek

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

Czerwiec 2025

Mateusz Redosz

- Przebudowa profilu użytkownika poprzez rozdzielenie widoków własnego profilu i profili innych użytkowników, dodanie przycisków akcji (follow/unfollow, zaproszenie do znajomych), usprawnienie nawigacji z kart znajomych i [sidebara](#) oraz aktualizacja testów.
- Dodanie sekcji „*Ulubione spoty*” w panelu konta (lista, filtrowanie, usuwanie oraz podgląd na mapie), wprowadzenie współdzielonych komponentów i modeli dla tej funkcji oraz uporządkowanie przestarzałego kodu i modeli współrzędnych.
- Przebudowa funkcjonalności społecznościowej: rozdzielenie widoków sekcji *social* dla właściciela profilu i osoby oglądającej, aktualizacja modelu danych i routingu oraz dodanie nawigacji z liczników profilu do odpowiednich list (friends, followers, followed), wraz z wprowadzeniem osobnego [slice Redux](#) do zarządzania aktywnym typem listy social.

Stanisław Oziemczuk

- Refaktoryzacja komentarzy do [spotów](#) – zastąpienie ogólnego modelu szczegółowym, wprowadzenie nowych komponentów [UI](#), uporządkowanie warstwy [API](#) oraz poprawa wyglądu i układu sekcji komentarzy.

Lipiec 2025

Mateusz Redosz

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

Adam Langmesser

- Dodanie pobierania danych czatu i wysyłania wiadomości po [WebSocket](#).
- Uporządkowanie formatowania kodu z wykorzystaniem narzędzia [Prettier](#), dodanie skryptu `format:check` oraz sprawdzania formatowania w potokach [CI/CD](#).
- Ujednolicenie modelu danych czatu, uproszczenie logiki komunikacji z [API](#) i struktury store'a [Redux](#), dostosowanie komponentów czatu do nowego modelu oraz poprawa wyglądu czatu.
- Poprawa danych deweloperskich w bazie (m.in. przykładowych [spotów](#) i użytkowników).

- Usprawnienie mechanizmu nieskończonego przewijania i nazewnictwa listy czatów.
- Stworzenie ogólnej logiki i komponentów pomocniczych umożliwiających wszystkim członkom zespołu wygodne korzystanie z [websocketów](#) na froncie.
- Dodanie grupowania wiadomości na czacie po ich dacie wysłania, implementacja wielowierszowego pola tekstowego.
- Wprowadzenie drobnych poprawek w potokach [CI/CD](#).

Stanisław Oziemczuk

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

Kacper Badek

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

Sierpień 2025

Adam Langmesser

- Dodanie do czatu możliwości wyszukiwania i wysyłania animowanych obrazów ([gifów](#)) oraz wysyłania [emoji](#).

- Usprawnienie czatu poprzez wprowadzenie optymistycznego wysyłania wiadomości, dodanie nowych [endpointów](#) do stronicowanego pobierania starszych wiadomości.

Stanisław Oziemczuk

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

Mateusz Redosz

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

Kacper Badek

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

Wrzesień 2025

Adam Langmesser

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

Mateusz Redosz

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

Stanisław Oziemczuk

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

Kacper Badek

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

6.4 Etap 3 (październik 2025 – styczeń 2026)

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

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

Październik 2025

Adam Langmesser

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

Mateusz Redosz

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

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

Stanisław Oziemczuk

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

Kacper Badek

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

Listopad 2025

TODO: do uzupełnienia

Grudzień 2025

TODO: do uzupełnienia

Styczeń 2026

TODO: do uzupełnienia

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

Rozdział 7

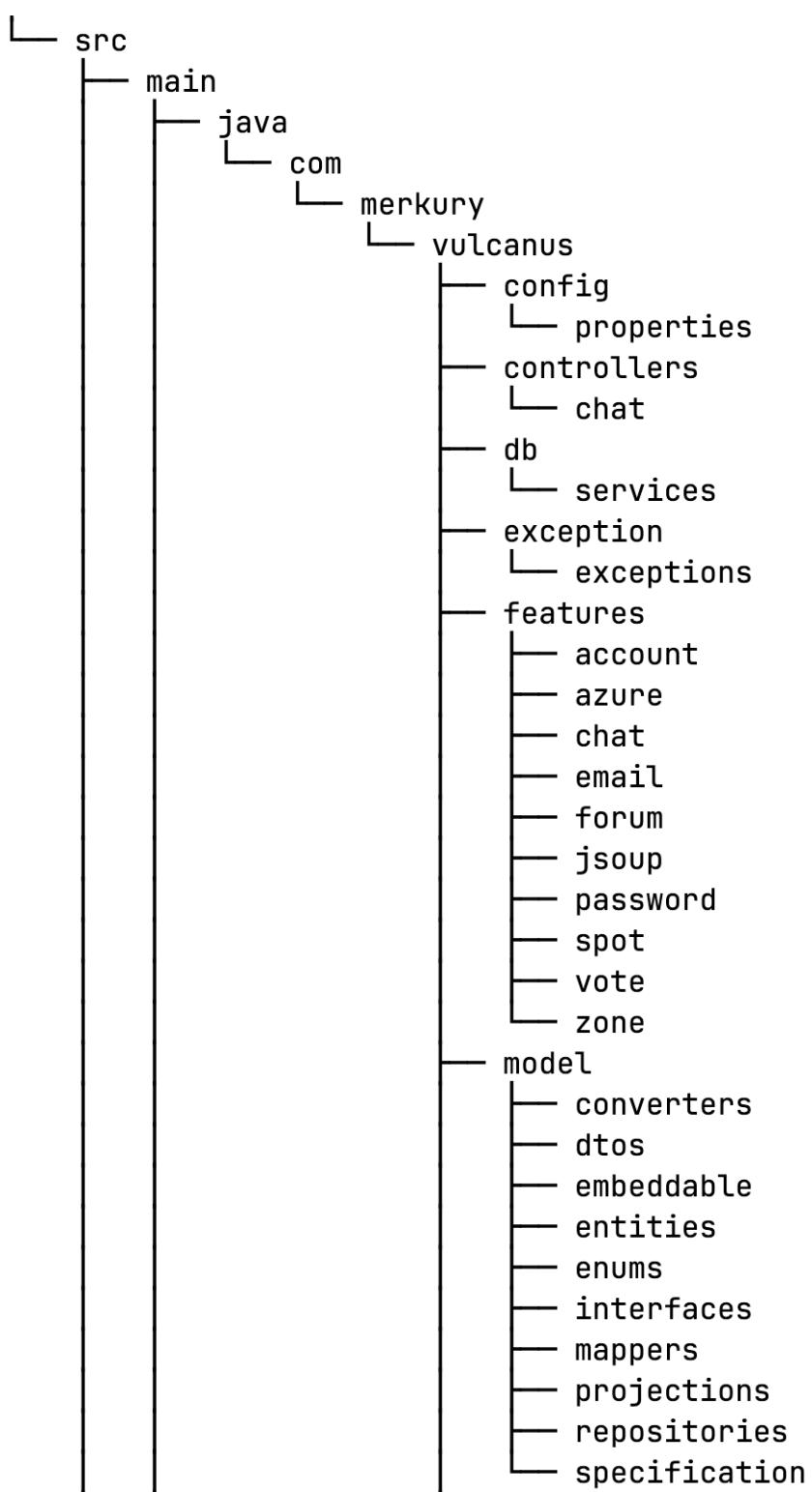
Realizacja Projektu

7.1 Implementacja backendu

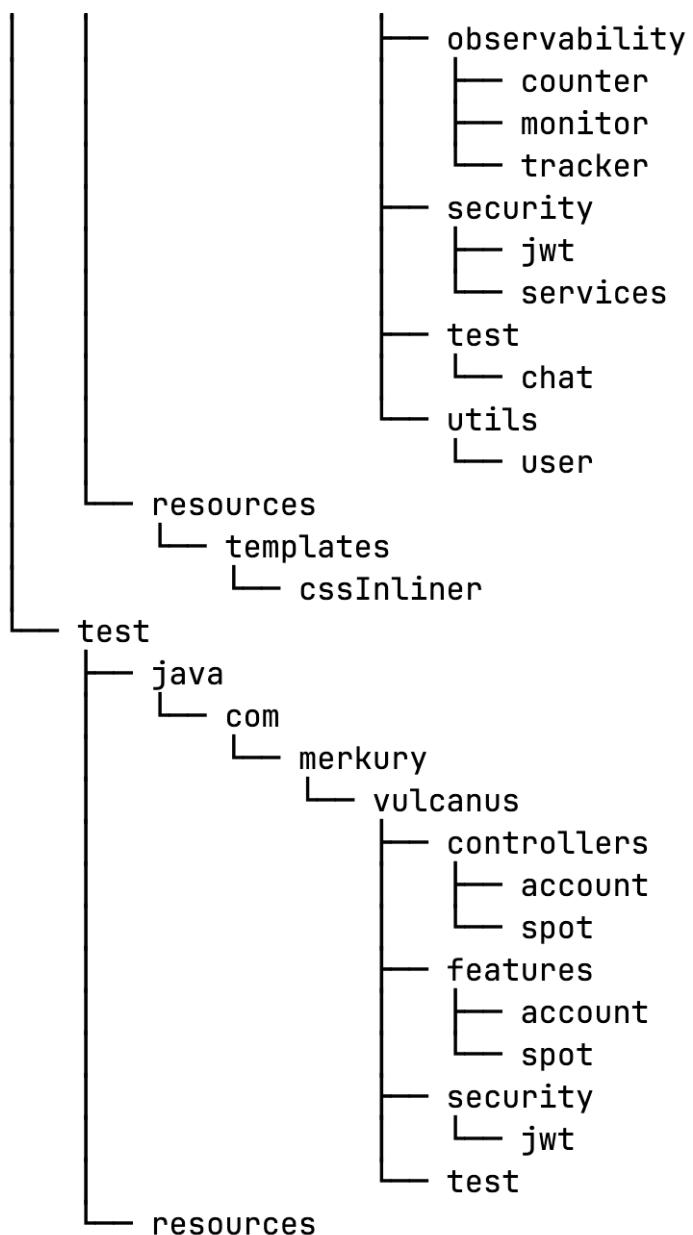
W niniejszym rozdziale przedstawiono strukturę backendu aplikacji, opis implementowanych endpointów, integrację z bazą danych, mechanizmy uwierzytelniania oraz proces konteneryzacji systemu.

7.1.1 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 7.1: Struktura katalogów (1)



Rysunek 7.2: 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.

7.1.2 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

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

Spoty 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

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

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

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

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 górę) – 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 górę) – 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

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

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 pasujacych 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> (stocionowana 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}

7.1.3 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.

7.1.4 Obsługa uwierzytelnienia

7.1.5 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. 7.3), 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 7.3: Plik konfiguracyjny Docker Compose

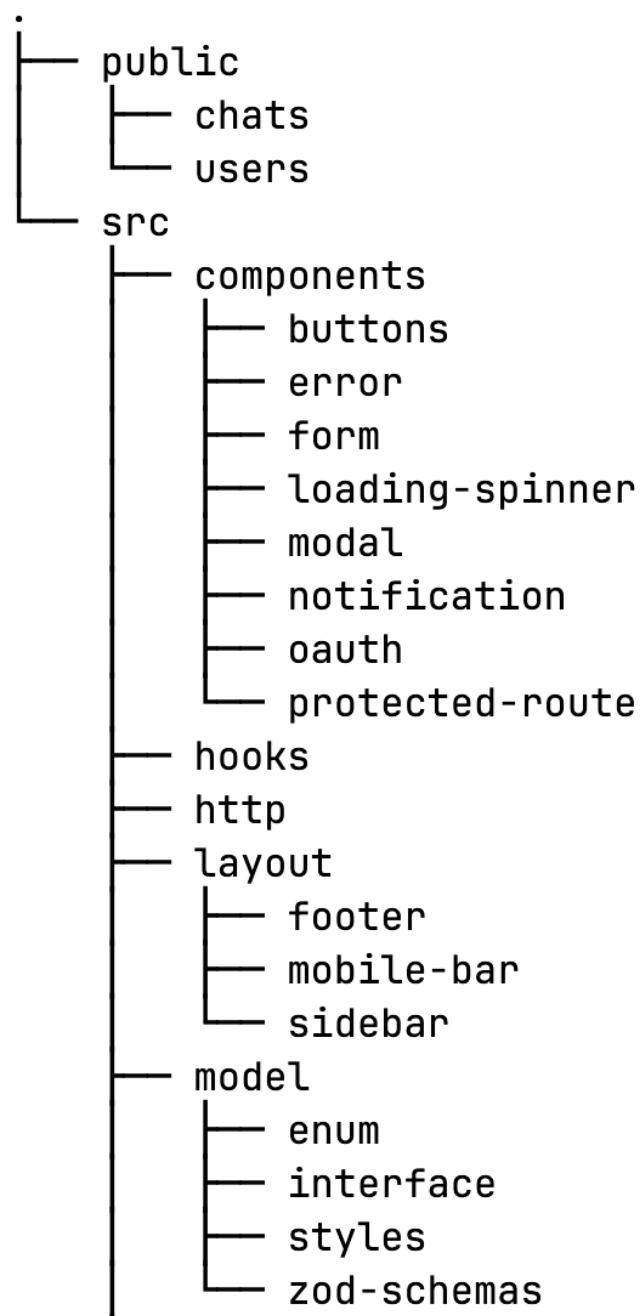
7.2 Implementacja frontendu

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

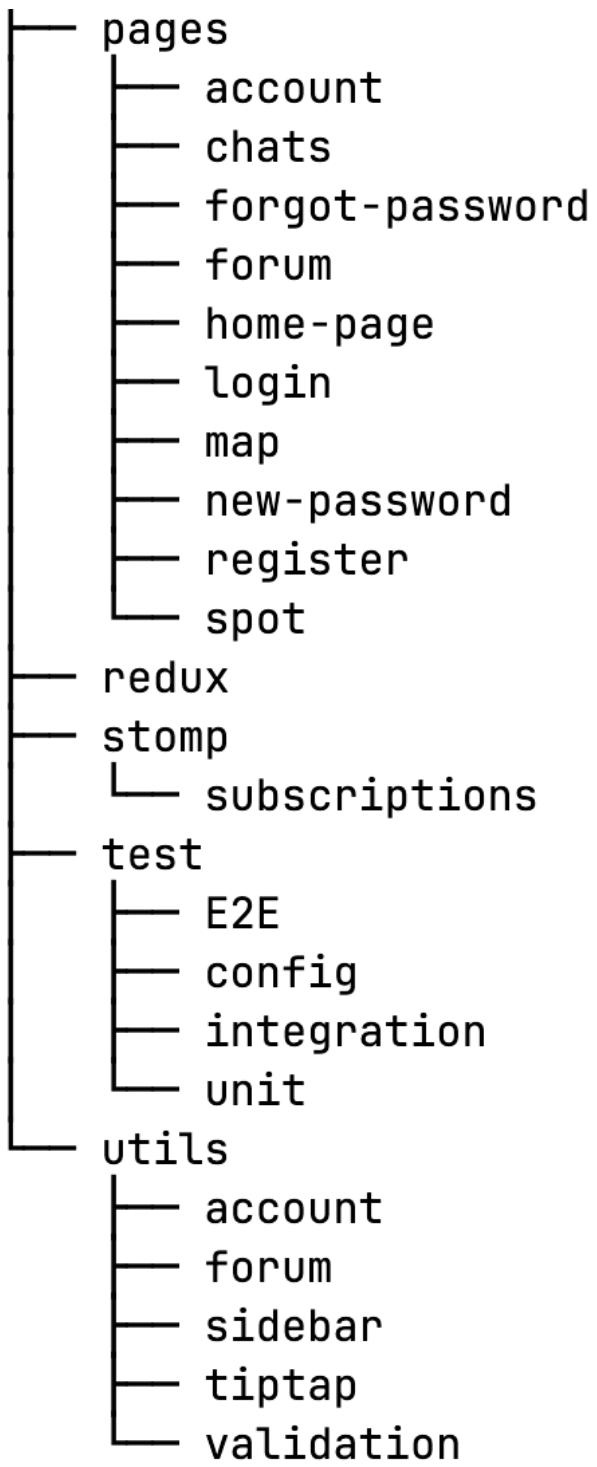
7.2.1 Struktura aplikacji

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 [7.4](#) oraz [7.5](#).



Rysunek 7.4: Struktura katalogów (1)



Rysunek 7.5: 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 7.6: 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 7.7: 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 7.6 oraz 7.7), wybrane ścieżki opakowano w komponent `ProtectedRoute`. Komponent ten pełni

rolę bramki (rysunek 7.8).

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 7.8: Implementacja komponentu bramki (`ProtectedRoute`)

7.2.2 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**) [14], jak i ze **stanu** globalnego, utrzymywanej przez bibliotekę **React Redux** [15]. 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 7.9 oraz 7.10.

```

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 7.9: 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 7.10: Przykładowy slice odpowiedzialny za sprawdzenie czy użytkownik jest zalogowany

7.2.3 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](#) [16] oraz [biblioteki TanStack Query](#) [17]. We

wszystkich ścieżkach wymagających zalogowania użytkownika do zapytania dołączany jest token **JWT**. Token przekazywany jest w ciasteczku dzięki ustawieniu parametru **withCredentials** na wartość **true**. Przykładem pliku odpowiedzialnego za taką komunikację jest **account.js** (rys. 7.11 i 7.12), 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 7.11: 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 7.12: 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. 7.13.

```
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 7.13: Wykorzystanie TanStack Query przy logowaniu użytkownika

7.2.4 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 [18]. 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. 7.14 i 7.15). 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 7.14: 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 7.15: 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. 7.16), 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 7.16: 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. 7.17).

```
<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 7.17: Przykładowe użycie klas Tailwind (w tym wariantu **dark:**)

Aby uzyskać płynniejsze i przyjemniejsze animacje, wykorzystano **bibliotekę Motion [19]**. 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. 7.18). 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 7.18: Implementacja animacji z wykorzystaniem Motion

7.2.5 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. 7.19 oraz 7.20).

```

<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 7.19: 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 7.20: Implementacja zaawansowanej wersji wyszukiwarki

Przełączanie pomiędzy tymi widokami odbywa się za pomocą przycisku umieszczonego w górnej części strony (rys. 7.21).

```
<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 7.21: Implementacja komponentu do przełączania trybów

W trybie prostym prezentowana jest karuzela (rys. 7.22) 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 7.22: 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. 7.20).

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. 7.21) zawiera dwa elementy `NavLink` z biblioteki React Router, co pozwala na przełączanie widoków bez konieczności przeładowywania całej strony.

W komponencie `SearchBar` (rys. 7.23) 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 7.23: Implementacja prostej wyszukiwarki

Komponent `SearchSpotList` (rys. 7.24) 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 7.24: 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.

7.2.6 Mapa

7.2.7 Chat

7.2.8 Forum

7.2.9 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 [7.2.9.9](#)).

7.2.9.1 Profile

Zakładka **Profile** odpowiada za prezentację profilu użytkownika. Została zbudowana z pięciu podstawowych komponentów:

- **UserOwnProfile**,
- **ProfileForViewer**,
- **Profile**,
- **ProfileStat**,
- **MostPopularImage**.

UserOwnProfile (rys. [7.25](#)) Ten komponent służy do wyświetlania profilu zalogowanego 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 7.25: Komponent UserOwnProfile.

ProfileForViewer (rys. 7.26) 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 7.26: 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 [7.2.9.9](#)).

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 [7.2.9.5](#)); 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 [7.2.9.5](#)).

`MostPopularImage` Komponent odpowiada za wyświetlenie pojedynczego zdjęcia wraz z liczbą polubień i wyświetleń. Informacje umieszczone są na półprzezroczystym pasku w dolnej części kafelka.

7.2.9.2 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. 7.27) 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 7.27: Komponent FavoriteSpots.

`FavoriteSpotTile` Komponent 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.

7.2.9.3 Photos

Zakładka `Photos` odpowiada za prezentację zdjęć dodanych przez użytkownika. Korzysta z następujących głównych komponentów:

- `Photos`,
- `Media`,
- `Photo`.

`Photos` (rys. 7.28) 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}
      onChange={handleChangeDate}
      isLoading={isLoading}
      mediaList={allItems}
      loadMoreRef={loadMoreRef}
      isFetchingNextPage={isFetchingNextPage}
    />
  );
}

```

Rysunek 7.28: 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. 7.29) Jest to uniwersalny komponent używany zarówno w zakładce `Photos`, jak i `Movies` (por. sekcja 7.2.9.4). Widok jest opakowany w `AccountWrapper`, a nagłówek sekcji wyświetlany przez `AccountTitle` z tekstem zależnym od wariantu.

Poniżej nagłówka znajduje się panel `SortAndDateFilters` (por. sekcja 7.2.9.9) 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 7.29: Komponent Media.

7.2.9.4 Movies

Zakładka **Movies** odpowiada za prezentację filmów dodanych przez użytkownika. Pod względem struktury jest bardzo podobna do zakładki **Photos** (por. sekcja [7.2.9.3](#)) i korzysta z tych samych komponentów ogólnych:

- **Movies**,
- **Media**,
- **Movie**.

Movies (rys. [7.30](#)) 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 7.30: Komponent `Movies`.

`Movie` 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.

7.2.9.5 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. 7.31 i 7.32) 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 7.31: 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 7.32: 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. 7.33) 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 7.33: 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 `debouncowane` 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 [7.2.9.1](#)). 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".

7.2.9.6 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. 7.34) 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 nagłówk `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 7.34: 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 nadzawanego. W razie błędu walidacji konturu wyświetlany jest komunikat pod mapą.

AddSpotModal Komponent odpowiada za dodawanie nowego [spota](#). W stanie przechowywanym 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).

7.2.9.7 Comments

Zakładka **Comments** wyświetla wszystkie komentarze dodane przez użytkownika pod [spotami](#). Składa się z dwóch głównych komponentów:

- **Comments**,
- **CommentsList**.

Comments (rys. 7.35) 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](#) ustawiany jest komunikat błędu, wyświetlany przez [Notification](#). Widok opakowany jest w [AccountWrapper](#) z wariantem **COMMENTS** i nagłówek [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 7.35: Komponent Comments.

`CommentsList` Komponent otrzymuje listę komentarzy i wyświetla je w postaci kafelków zawierających godzinę dodania oraz treść komentarza.

7.2.9.8 Settings

Zakładka `Settings` umożliwia użytkownikowi edycję podstawowych danych konta. Została zrealizowana za pomocą dwóch głównych komponentów:

- `Settings`,
- `DisableInput`.

`Settings` (rys. 7.36) 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 7.36: 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.

7.2.9.9 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. 7.37) 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 7.37: Komponent AccountWrapper.

AccountTitle (rys. 7.38) 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 7.38: 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. 7.39) 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. 7.40) 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 7.40: Komponent SortAndDateFilters.

DateBadge (rys. 7.41) 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 7.41: 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.

7.2.10 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.

7.2.10.1 Logowanie

W skład formularza logowania wchodzą następujące komponenty:

- `Login`,
- `FormContainer`,
- `FormInput`,
- `SubmitFormButton`.

`Login` (rys. 7.42 i 7.43) 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 7.42: 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 7.43: Komponent Login (2).

7.2.10.2 Rejestracja

W skład formularza rejestracji wchodzą następujące komponenty:

- Register,

- `FormContainer`,
- `FormInput`,
- `SubmitFormButton`.

`Register` (rys. 7.44 i 7.45) 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 7.44: 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 7.45: Komponent Register (2).

7.2.10.3 Reset hasła

W skład formularza resetu (przypomnienia) hasła wchodzą następujące komponenty:

- `ForgotPassword`,
- `FormContainer`,
- `FormInput`.

ForgotPassword (rys. 7.46) 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 7.46: Komponent ForgotPassword.

7.2.10.4 Nowe hasło

W skład formularza ustawiania nowego hasła wchodzą następujące komponenty:

- `NewPassword`,
- `FormContainer`,
- `FormInput`.

`NewPassword` (rys. 7.47 i 7.48) 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 7.47: 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>

```

282

Rysunek 7.48: Komponent NewPassword (2).

7.2.10.5 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. 7.49 i 7.50) 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 7.49: 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 7.50: Komponent `FormContainer` (2).

`FormInput` (rys. 7.51 i 7.52) Komponent realizuje pojedyncze pole wejściowe formularza z animowaną etykietą. 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 7.51: 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 7.52: Komponent FormInput (2).

`SubmitFormButton` (rys. 7.53) 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 7.53: Komponent `SubmitFormButton`.

7.3 Implementacja CI/CD

7.4 Implementacja WebSocket

Niniejszy rozdział prezentuje implementację protokołu [WebSocket](#). Rozdział przygotowano w oparciu o podane źródła:

- Specyfikacja protokołu WebSocket (RFC6455) [20].
- Dokumentacja Spring dotycząca obsługi WebSocket [21].
- Dokumentacja Spring dotycząca integracji STOMP [22].
- Dokumentacja biblioteki klienckiej SockJS [23].

7.4.1 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.

7.4.2 Zastosowanie WebSocket w naszym projekcie

Główną metodą komunikacji między backendem 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 zamknięte.

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. Rozwiązań opartych na long

[pollingu](#) są z kolei bardziej złożone w implementacji i wciąż nie zapewniają tak naturalnego, dwukierunkowego kanału komunikacji, jakiego oczekuje się od współczesnego 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](#).

7.4.3 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.

7.4.3.1 Konfiguracja WebSocket

Na rysunku 7.54 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 7.54: 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).

7.4.3.2 Kontroler STOMP

Rysunek 7.55 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 7.55: 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.

7.4.3.3 Serwis dystrybucji wiadomości

Rysunek 7.56 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()) Optional.ofNullable()
            .orElseThrow(() -> new IllegalArgumentException("Chat not found")) Chat
            .getParticipants();
        log.info("Broadcasting chat message to {} participants: {}", (long) chatParticipants.size(), chatMessageDto);
        chatParticipants.forEach( ChatParticipant 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 7.56: 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**.

7.4.3.4 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**).

7.4.4 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.

7.4.4.1 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 7.57: 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 7.58: 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 7.59: Serwis komunikacji WebSocket po stronie frontendu (część 3/4).

```

Publikuje wiadomość JSON-ując 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 7.60: 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**.

7.4.4.2 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 7.61: 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 7.62: Mechanizm rejestracji 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).

7.4.4.3 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(
    (state: RootState) :AccountSliceProps => state.account,
  );
  const selectedChatId :number | null = useSelectorTyped((s :{account: AccountSliceProps; notifications: NotificationsSliceProps} ) :number | null => s.chats.selectedChatId);
  const dispatch :ThunkDispatch<{ account: AccountSliceProp... }> = 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; Show usages & Adam Langmesser

    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 => Show usages & Adam Langmesser
useContext(WebSocketContext);
```

Rysunek 7.63: 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.

7.4.4.4 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 7.64: 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.

7.4.5 Przebieg komunikacji

Rozdział 8

Testy

- 8.1 Testy jednostkowe
- 8.2 Testy integracyjne
- 8.3 Testy E2E
- 8.4 Wyniki testów i wnioski

Rozdział 9

Prezentacja systemu

9.1 Strona główna

9.2 Strona mapy

9.3 Strona chatu

9.4 Strona forum

9.5 Panel logowania

9.6 Panel konta użytkownika

Rozdział 10

Nakład pracy

10.1 Ogólny nakład pracy

10.2 Indywidualne nakłady pracy

10.2.1 Adam Langmesser

10.2.2 Mateusz Redosz

Na projekt poświęciłem łącznie 324 godziny, z czego 237 przeznaczyłem na prace deweloperskie, 111 na pisanie dokumentacji, 19 godzin na [Review kodu](#), 19 na spotkania dotyczące omówienia dalszych prac projektowych oraz przy pomocy innym członkom zespołu oraz 49 godzin poświęciłem nad stworzeniem widoków na figmie. Prace nad częścią deweloperską rozpoczęłem 04.08.2024 a zakończyłem 08.09.2025. W projekcie pracowałem nad Rejestracją użytkownika, tokenem [JWT](#), częściową implementacją [CI/CD](#), stroną główną, zaimplementowaniem [Sidebara](#) oraz podstroną dla użytkownika. Moje wylistowane zadania z Jira:

1. Dokumentacja
 - TODO
2. Design
 - Ustalić paletę kolorystyczną

- Propozycja wyglądu

3. Backend i Frontend

- Formularz rejestracji
- Routing
- Formatowanie w React (prettier)
- Obsługa JWT na frontend
- oAuth Frontend
- Update JWT
- Refactor JWT
- Stworzenie komponentu Notification i poprawa błędów
- Implementacja pierwszych testów
- Zaimplementowanie kolejki w komponencie notification
- Dodanie reduxa do rejestracji
- Zmiana sposobu pobierania danych o spotach
- Obsługa customowych błędów z jakarta.validation
- Obsługa auto wylogowania przy starcie
- Domyślna wiadomość w notification
- Poprawa headera
- Ciemny motyw
- Refactor pogody
- Propozycja wyglądu
- Przeniesienie zdjęć z google drive
- Dodać Type script do Reacta
- Aktualizacja tailwinda i dodanie kolorów
- Podstawowy Sidebar

- Strona główna z prostymi filtrami
- Strona główna z zaawansowanymi filtrami
- [Sidebar](#)
- Strona profilu
- Ustawienia
- Listy spotów
- Lista zdjęć
- Lista filmów
- Lista znajomych
- Dodanie spotów
- Lista komentarzy
- Strona główna profilu
- Listy
- Poprawa [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

11.1 Osiągnięte rezultaty

11.2 Napotkane wyzwania

11.3 Plany na przyszłość

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.. [290](#), [293–295](#)

akcja Redux

Obiekt opisujący zdarzenie/operację zmiany stanu w Redux, obsługiwany przez reducery (np. `chatActions.setLastMessage(...)`). [308](#)

API

(ang. *application programming interface*); zbiór reguł i operacji do komunikacji z oprogramowaniem.. [26](#), [27](#), [111](#), [112](#), [145](#), [150](#), [153–155](#), [159](#), [240](#), [257](#), [321](#)

Azure Blob Storage

Usługa magazynu obiektowego w chmurze Microsoft Azure do przechowywania nieustrukturyzowanych danych (*blobs*) takich jak obrazy, wideo i pliki. Udostępnia kontenery, warstwy dostępu, wersjonowanie oraz tokeny SAS; często używana do hostowania multimediów w aplikacjach webowych.. [154](#)

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](#), [15](#), [24](#), [116](#), [117](#), [119](#), [144–152](#), [154](#), [156](#), [157](#), [159](#), [160](#), [162](#), [211](#), [224](#), [239–241](#), [244](#), [246](#), [258](#), [259](#), [261](#), [262](#), [264](#), [272](#), [275](#), [278](#), [280](#), [283](#), [290](#), [293](#), [302](#), [312](#), [320](#), [331](#)

Backlog

Lista zadań, które należy wykonać w ramach projektu, używana w metodykach zwinnych.. [25](#)

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. [116](#), [117](#), [119](#), [211](#), [214](#), [293](#), [295](#)

Biblioteka

Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. [19](#), [117](#), [211](#), [219](#), [221](#), [224](#), [230](#), [235](#), [239](#), [251](#), [258](#), [261](#), [264](#), [268](#)

BPMN

(ang. *Business Process Model and Notation*); standardowa notacja graficzna, która umożliwia szczegółowe przedstawienie i dokumentowanie procesów biznesowych.. [26](#)

Cache

Mechanizm przechowywania danych w celu przyspieszenia ich ponownego odczytu. [23](#), [24](#), [116](#), [117](#), [156](#), [214](#)

callback

Funkcja przekazywana jako parametr, wywoływana automatycznie przy wystąpieniu zdarzenia (np. gdy nadaje się wiadomość z WebSocket).. [296](#), [301](#), [304](#), [308](#)

CDN

Skrót od *Content Delivery Network*. Rozproszona sieć serwerów służąca do szybkiego dostarczania statycznych zasobów (np. obrazów, arkuszy CSS, skryptów JavaScript) z węzłów geograficznie najbliższych użytkownikowi, co zmniejsza opóźnienia i odciąża serwer aplikacji. [152](#)

CI/CD

Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. [25](#), [149–151](#), [154–156](#), [311](#), [314](#)

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. [146](#)

cleanup

„Sprzątanie” zasobów: usuwanie subskrypcji, rozłączanie połączeń i czyszczenie struktur pomocniczych, aby nie pozostawiać aktywnych uchwytów. [302](#), [306](#)

Convention Over Configuration

Zasada programowania polegająca na przyjmowaniu domyślnych, bazowych reguł, zamiast ręcznego implementowania konfiguracji. [16](#)

CORS

Skrót od *Cross-Origin Resource Sharing*. Mechanizm bezpieczeństwa w przeglądarkach, który kontroluje, czy aplikacja z jednej domeny może wykonywać zapytania HTTP do serwera w innej domenie; konfigurowany za pomocą nagłówków HTTP. [146](#)

CSS

Kaskadowe arkusze stylów (Cascading Style Sheets) — język opisu prezentacji dokumentów (np. HTML). Definiuje wygląd interfejsu: układ, kolory, typografię,

odstęp, animacje i zachowania responsywne, oddzielając warstwę treści od warstwy prezentacji.. [230](#)

cykl życia komponentu

Etapy działania komponentu (np. montowanie, aktualizacja, odmontowanie) determinujące momenty inicjalizacji i zwalniania zasobów. [296](#), [304](#)

Debounce

Technika programistyczna polegająca na ograniczaniu liczby wywołań funkcji po przez jej uruchomienie dopiero po upływie zadanego czasu od ostatniego wywołania.. [257](#), [261](#)

deserializacja

Proces odtworzenia obiektu z formatu przenośnego (np. JSON) po stronie odbiorcy. [308](#)

Design

Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). [311](#)

destynacja

Adres logiczny (ścieżka) w komunikacji **STOMP**, na który wysyła się wiadomości lub który się **subskrybuje** (np. `/app/...` albo `/subscribe/...`).. [289](#), [291–293](#), [295](#), [296](#), [301](#), [302](#), [304](#), [307](#), [333](#), [334](#)

diagram Gantta

Graficzne narzędzie do planowania i monitorowania przebiegu projektu, przedstawiające zadania jako poziome paski na osi czasu wraz z ich początkiem, końcem oraz ewentualnymi zależnościami.. [147](#)

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 mniejszą 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. . [12](#), [143](#)

dispatch

Funkcja w Redux służąca do wysyłania akcji, które modyfikują stan przechowywany w store. [306](#)

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. [151](#), [214](#), [325](#)

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. [145](#), [151](#), [214](#), [215](#)

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. [214](#)

Dockerfile

Plik tekstowy zawierający instrukcje opisujące, jak zbudować obraz Dockera (jakiej podstawy użyć, jakie pliki skopiować, jakie polecenia uruchomić). Na jego

podstawie Docker tworzy gotowy obraz kontenera. [151](#)

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). [19](#), [261](#)

Droniarz

Potoczne określenie osoby, która jest jednocześnie pilotem oraz operatorem drona. Zwykle entuzjasta dronów.. [10](#), [11](#), [337](#)

Droniarz foto/video

Pilot wykorzystujący drony fotograficzne/filmowe do rejestracji materiałów wizualnych (zdjęcia, wideo), zwykle z naciskiem na stabilizację i jakość obrazu.. [27](#)

emoji

Małe graficzne ikonki używane do wyrażania emocji lub pojęć w komunikacji cyfrowej (np. uśmiechnięta buźka, kciuk w górę, symbol serca).. [89](#), [156](#), [158](#)

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).. [153](#), [155](#), [157](#), [160](#), [239](#), [254](#), [291](#), [292](#), [295](#)

ESLint

Statyczny analizator kodu JavaScript/TypeScript. Umożliwia wykrywanie błędów, niespójności stylu oraz potencjalnych problemów poprzez zestaw reguł, które można dostosować do projektu. [147](#)

fallback

Mechanizm awaryjnego przełączania na alternatywny sposób działania, gdy preferowana metoda jest niedostępna. W kontekście WebSocket/SockJS oznacza au-

tomatyczne przejście z WebSocket na techniki oparte o HTTP (np. streaming lub long polling), aby utrzymać komunikację.. [292](#)

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.. [162](#), [216](#)

Framework

Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. [2](#), [15](#), [16](#), [117](#), [162](#), [227](#)

Frontend

Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. [2](#), [15](#), [19](#), [24](#), [116](#), [117](#), [119](#), [144–150](#), [152](#), [154](#), [157](#), [158](#), [160](#), [215](#), [216](#), [224](#), [238](#), [271](#), [290](#), [295](#), [300](#), [302](#), [312](#), [324](#)

GIF

Format graficzny *Graphics Interchange Format* obsługujący krótkie, zapętlone animacje. W aplikacjach czatowych wykorzystywany do wysyłania „reakcji” w postaci ruchomych obrazków. [85](#), [86](#), [156](#), [158](#)

GitHub

Platforma hostingu repozytoriów *Git* w chmurze, oferująca m.in. pull requesty, system zgłoszeń (issues), zarządzanie wersjami oraz integrację z narzędziami CI/CD. [144](#), [146](#), [147](#), [149](#)

handshake

Etap początkowy zestawiania połączenia (np. [WebSocket](#)), w którym [klient](#) i [serwer](#) uzgadniają parametry komunikacji.. [292](#), [295](#), [301](#)

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`).. [221](#), [222](#), [226](#), [239–241](#), [243](#), [244](#), [246](#), [250](#), [251](#), [257](#), [259](#), [261](#), [262](#), [264](#), [268](#), [272](#), [280](#), [296](#), [304](#)

HTTP

Podstawowy protokół komunikacji w sieci WWW, wykorzystywany m.in. przez [REST API](#). Zwykle działa w modelu „żądanie–odpowiedź”, bez stałego połączenia.. [289](#), [290](#), [327](#)

IDE

(ang. *integrated development environment*); zintegrowane środowisko programistyczne, służące do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. [24](#), [144](#)

IETF

Organizacja opracowująca i publikująca standardy internetowe, m.in. dokumenty typu [RFC](#).. [289](#), [331](#)

Infinite scroll

Wzorzec interfejsu użytkownika, w którym kolejne porcje treści są automatycznie doładowywane podczas przewijania strony w dół, zamiast być podzielone na odrębne, ręcznie przełączane strony. [94](#), [236](#), [244](#), [246](#), [255](#), [259](#), [262](#)

IntelliJ IDEA

Zintegrowane środowisko programistyczne (IDE) firmy JetBrains, szeroko stosowane przy tworzeniu aplikacji backendowych w ekosystemie Spring. Oferuje m.in. podpowiedzi składni, refaktoryzację kodu, debugger oraz integrację z systemami kontroli wersji. [145](#)

jakarta validation

Jakarta Validation to specyfikacja (i zestaw adnotacji, typu @NotNull, @Size itd.) służąca do automatycznego sprawdzania poprawności danych w aplikacjach stworzonych za pomocą Java/Jakarta EE/Spring, np. przy walidacji pól DTO, encji czy parametrów metod.. [150](#)

Jira

Narzędzie firmy Atlassian do zarządzania projektami i zadaniami, szeroko stosowane w metodykach zwinnych. Umożliwia pracę z epikami, taskami, podtaskami oraz tablicami Scrum i Kanban. [144](#)

JSON

Format tekstowy do wymiany danych (JavaScript Object Notation), często używany jako reprezentacja komunikatów API i wiadomości w aplikacjach webowych. [302](#), [308](#)

jsoup

Biblioteka Java do przetwarzania dokumentów HTML, umożliwiająca parsowanie, przeszukiwanie i modyfikowanie struktury dokumentu w sposób zbliżony do pracy z DOM-em i selektorami CSS.. [157](#)

JVM

(ang. *Java Virtual Machine*); maszyna wirtualna oraz środowisko do wykonywania kodu bajtowego Javy. [16](#)

JWT

Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. [112](#), [146–148](#), [225](#), [311](#)

klient

Część systemu (zwykle po stronie [frontendu](#)), która korzysta z usług serwera i prezentuje dane użytkownikowi.. [290–293](#), [295](#), [316](#), [322](#), [325](#), [327](#), [331](#), [336](#)

klient STOMP

Biblioteka/komponent po stronie klienta realizujący protokół STOMP (subskrypcje, publikacja, ramki, nagłówki) nad transportem WebSocket/SockJS. [300](#), [301](#)

kontekst React'a

Mechanizm w React pozwalający udostępnić wspólny obiekt wielu komponentom bez przekazywania go przez [props](#).. [295](#), [296](#), [304](#), [305](#)

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](#). [214](#)

LaTeX

System składu tekstu wykorzystywany do przygotowywania profesjonalnych dokumentów technicznych i naukowych. Umożliwia precyzyjne formatowanie, zarządzanie odwołaniami, bibliografią i wzorami matematycznymi. [152](#), [161](#)

Leaflet

Lekka biblioteka JavaScript do tworzenia interaktywnych map w przeglądarce, często używana z danymi z OpenStreetMap. Umożliwia dodawanie znaczników, warstw oraz obsługę interakcji użytkownika. [147](#), [153](#)

long polling

Odmiana [polling'u](#): **klient** wysyła żądanie, a **serwer** wstrzymuje odpowiedź do momentu pojawiения się nowych danych (lub upłynięcia limitu czasu).. [290](#)

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*).. [230](#), [330](#)

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. [159](#), [240](#), [246](#), [258](#), [261](#)

montowanie

Moment „podpięcia” komponentu do drzewa UI.. [304](#)

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.. [246](#), [257](#), [258](#), [261](#), [264](#)

nagłówek

Metadane dołączane do wiadomości lub żądania (np. dodatkowe pola opisujące typ komunikatu, identyfikator, autoryzację).. [301](#), [302](#)

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. [147](#), [149](#), [151](#)

odmontowanie

Usunięcie komponentu z drzewa UI.. [304](#)

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.. [295](#)

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.. [152](#), [154](#), [156–158](#), [160](#), [161](#)

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. . [32](#), [50](#), [51](#), [338](#)

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`. [165](#), [171–181](#), [183](#), [185](#), [187–190](#), [192–195](#), [197–205](#), [207–211](#)

polling

Sposób komunikacji, w którym **klient** cyklicznie „odpytuje” **serwer** (np. co kilka sekund) o nowe dane, zwykle poprzez **HTTP**.. [290](#), [325](#)

prefiks

Ustalony początek ścieżki (np. `/app` lub `/subscribe`), który porządkuje i rozróżnia typy komunikacji w aplikacji.. [291](#), [292](#), [295](#)

Prettier

Narzędzie do automatycznego formatowania kodu (np. JavaScript, TypeScript, CSS, HTML). Narzuca spójny styl formatowania, zastępując ręczne ustawianie wcięć i łamań linii. [144](#), [155](#)

PRO

Przedmiot realizowany na 5. semestrze studiów, prowadzony przez dr. inż. Martę Łabudę. W ramach przedmiotu wybrano temat projektu oraz wytworzono wstępna

dokumentację projektu, w tym m.in. wymagania.. [147](#)

Props

Właściwości przekazywane do komponentu React przez komponent nadzędny; służą do konfiguracji i przekazywania danych. Powinny być traktowane jako tylko do odczytu (read-only) wewnątrz komponentu potomnego.. [221](#), [325](#)

Protected route

Trasa w aplikacji, do której dostęp jest ograniczony, zwykle tylko dla zalogowanych użytkowników lub użytkowników z odpowiednimi uprawnieniami. Jeżeli użytkownik nie spełnia warunków, jest przekierowywany (np. na stronę główną). [220](#)

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.. [120](#), [150–152](#)

PRZ 2

Przedmiot realizowany na 7. semestrze studiów, prowadzony w przypadku zespołu projektowego przez mgr. inż. Adama Urbanowicza. W ramach przedmiotu dokonano pracy nad pracą inżynierską. Pan Adam Urbanowicz jako promotor doradzał zespołowi projektowemu.. [159](#)

PSEM

Przedmiot realizowany na 7. semestrze studiów, prowadzony w przypadku zespołu projektowego przez dr. inż. Marka Bednarczyka. W ramach przedmiotu dokonano wytwarzanie dokumentacji.. [159](#)

publish–subscribe

Model komunikacji, w którym nadawca publikuje zdarzenia na kanał, a odbiorcy, którzy go **subskrybują**, automatycznie otrzymują wiadomości.. [290](#), [333](#)

ramka

Podstawowa jednostka danych przesyłana w ramach połączenia WebSocket.. [289](#), [291](#)

React

Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. [19](#), [116](#), [117](#), [144](#), [221](#), [304](#)

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. [153](#)

reconnect

Automatyczne ponowne nawiązanie połączenia po jego zerwaniu (np. po chwilowej utracie internetu).. [295](#), [301](#)

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. [23](#), [116](#), [117](#), [119](#), [151](#), [214](#)

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.. [148](#), [149](#), [155](#), [221](#), [222](#), [252](#), [255](#), [257](#), [271](#), [272](#), [275](#), [296](#), [306–308](#), [332](#)

Responsywność

Określenie związane z projektowaniem responsywnym (Responsive Web Design, RWD), czyli dostosowywaniem interfejsu do różnych rozmiarów i parametrów ekra-

nów. Obejmuje m.in. elastyczne siatki, grafiki i [Media queries](#), tak aby układ i czytelność były zachowane na telefonach, tabletach i desktopach.. [227](#), [230](#)

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. **Bezstanowosc** — każde żądanie przez klienta zawiera wszystkie informacje niezbędne do jego obsłużenia. Po otrzymaniu żądania serwer nie przechowuje o nim żadnych informacji.
3. **Buforowalność (cache)** — odpowiedzi z API powinny informować, czy dane można cache'ować. Jeśli tak, to przy kolejnym żądaniu mogą być zwrócone z cache'a.
4. **Jednolity interfejs:**
 - **Identyfikacja zasobów** — każdy zasób musi być jednoznacznie zidentyfikowany w interakcji klient-serwer.
 - **Manipulacja zasobów poprzez reprezentację** — po otrzymaniu reprezentacji klient może zmienić stan zasobu przesyłając zmodyfikowaną reprezentację.
 - **Samoopisujące się wiadomości** — każde żądanie i odpowiedź powinny zawierać informacje do jego poprawnego przetworzenia.
 - **Hypermedia jako silnik stanu aplikacji (HATEOAS)** — po otrzymaniu odpowiedzi klient powinien móc dynamicznie poznać inne interakcje przez linki.
5. **Warstwowość** — klient nie wie, czy komunikuje się bezpośrednio z 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.

[25](#), [117](#), [162](#), [165](#), [290](#), [291](#), [323](#)

Review kodu

Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. [25](#), [143](#), [311](#)

RFC

Request for Comments to numerowane publikacje opisujące ustalenia dotyczące działania Internetu — m.in. protokoły, formaty danych i dobre praktyki. Są wydawane w ramach społeczności IETF.. [289](#), [323](#)

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. [156](#)

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.. [145](#)

serializacja

Proces zamiany obiektu (np. w TypeScript/Java) na format przenośny, np. JSON, w celu przesłania lub zapisu. [302](#)

serwer

Część systemu (zwykle po stronie [backendu](#)), która przetwarza żądania i udostępnia dane lub funkcje [klientom..](#) [289–292](#), [294](#), [322](#), [324](#), [325](#), [327](#), [334](#), [336](#)

Sidebar

Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. [110](#), [149](#), [152–154](#), [311–313](#)

Singleton

Wzorzec projektowy gwarantujący istnienie jednej, współdzielonej instancji obiektu w aplikacji. [306](#)

Skeleton loader

Wzorzec prezentowania stanu ładowania, w którym zamiast klasycznego „spinnera” wyświetlane są szare prostokąty imitujące docelowy układ treści. Poprawia subiektywne odczucie szybkości działania aplikacji. [161](#)

slice Redux

Slice Redux to wydzielona część globalnego stanu w [Redux](#), wraz z powiązanymi akcjami i reduktorami, odpowiedzialna za jeden obszar domeny (np. konto użytkownika, czat, mapę czy listę znajomych).. [153](#), [154](#)

Slug

Przyjazny dla użytkownika fragment adresu URL, zwykle oparty na tytule (np. `/post/jak-zaczac-latac-dronem`), ułatwiający identyfikację treści i pozycjonowanie w wyszukiwarkach. [159](#)

SockJS

Mechanizm kompatybilności dla [WebSocket](#): gdy przeglądarka lub sieć nie obsługuje WebSocket, SockJS może użyć alternatywnego sposobu komunikacji, aby zachować podobne zachowanie.. [300](#), [301](#)

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.. [19](#), [331](#)

Spot

Potencjalne miejsce do latania dronem, zaznaczone na mapie.. [25](#), [83](#), [122](#), [123](#), [126](#), [132](#), [133](#), [135](#), [137–140](#), [146](#), [148–159](#), [161](#), [233–238](#), [243–246](#), [258–262](#), [290](#)

Spring Boot

Framework w ekosystemie Spring dla języka Java, ułatwiający tworzenie aplikacji backendowych dzięki automatycznej konfiguracji, wbudowanemu serwerowi aplikacyjnemu oraz zestawowi gotowych starterów. [144](#)

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. [16](#), [291](#)

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. [148](#), [151](#)

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).. [221](#), [251](#), [259](#), [261](#), [268](#), [271](#), [296](#), [306–308](#)

STOMP

Protokół wiadomościowy działający „nad” [WebSocket](#). Wprowadza pojęcia [destynacji](#), [subskrypcji](#) i wysyłania komunikatów w modelu [publish–subscribe](#).. [289](#), [291](#), [293–295](#), [300–302](#), [304](#), [307](#), [316](#), [319](#)

StompJS

Biblioteka JavaScript/TypeScript (np. `@stomp/stompjs`) implementująca klienta protokołu STOMP. [301](#)

store

Centralne miejsce przechowywania stanu aplikacji w Redux. Pozwala odczytywać aktualny stan oraz wysyłać akcje zmieniające ten stan.. [255](#), [308](#)

subskrypcja

Mechanizm „zapisania się” na kanał ([destynacje](#)) w celu automatycznego odbierania wiadomości publikowanych przez serwer.. [289–293](#), [295](#), [296](#), [301](#), [304](#), [306](#), [307](#), [319](#), [328](#), [333](#)

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”.. [25](#), [144](#)

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. [144](#), [152](#), [266](#), [283](#)

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. [145](#), [239](#)

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. [146](#), [147](#)

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. [146](#), [149](#), [151](#)

testy jednostkowe

Testy sprawdzające poprawność działania pojedynczych, małych fragmentów kodu (np. funkcji, metod, klas) w izolacji od reszty systemu.. [149](#)

TinyMCE

Popularny edytor *rich text* osadzany w przeglądarce. Pozwala użytkownikowi formatować tekst (pogrubienia, listy, nagłówki, linki) w sposób przypominający klasyczny edytor tekstu, zapisując wynik zwykle w HTML. [156](#), [157](#), [159](#)

Tiptap

Nowoczesny, rozszerzalny edytor *rich text* dla aplikacji webowych oparty na silniku ProseMirror. Umożliwia budowanie rozbudowanych, modularnych edytorów treści, np. do postów na forum. [159](#)

TypeScript

Rozszerzenie do języka JavaScript dodający statyczne typowanie, interfejsy i narzędzia do większych projektów. Kompiluje się do czystego JavaScript, ułatwiając wykrywanie błędów w czasie kompilacji i refaktoryzacji.. [116](#), [117](#), [144](#), [152](#), [153](#), [221](#)

UI

Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. [19](#), [26](#), [88](#), [124](#), [154](#), [221](#)

UML

(ang. *Unified Modeling Language*); graficzny język wizualizacji, specyfikowania oraz dokumentowania składników systemów informatycznych.. [26](#)

URL

Adres zasobu w internecie (ang. *Uniform Resource Locator*), np. adres strony, widoku w aplikacji webowej lub konkretnego posta na forum. [159](#), [240](#), [280](#), [321](#)

useEffect

Hook React służący do uruchamiania efektów ubocznych po renderowaniu (oraz do definiowania funkcji sprzątającej wykonywanej przy odmontowaniu lub zmianie zależności). [304](#)

WebSocket

Protokół komunikacyjny umożliwiający utrzymanie stałego połączenia pomiędzy klientem a serwerem oraz dwukierunkową wymianę danych w czasie zbliżonym do rzeczywistego.. [155](#), [156](#), [288](#), [289](#), [291](#), [292](#), [295](#), [300](#), [304](#), [305](#), [322](#), [329](#), [332](#), [333](#)

Wolumen

Zarządzane przez Dockera magazyny danych dla kontenerów, które są odizolowane od maszyny hosta.. [24](#)

zmienna środowiskowa

Parametr konfiguracyjny przekazywany aplikacji z zewnątrz (np. plik `.env` lub konfiguracja CI/CD), pozwalający zmieniać zachowanie bez modyfikacji kodu. [305](#)

Spis tabel

Tabela 2.1: Karta udziałowca: Zespół projektowy	9
Tabela 2.2: Karta udziałowca: Promotor	10
Tabela 2.3: Karta udziałowca: Droniarze	11
Tabela 3.1: Zestawienie wszystkich bibliotek użytych na backendzie.	18
Tabela 3.2: Zestawienie wszystkich bibliotek i pluginów użytych na frontendzie.	23
Tabela 3.3: Usługa zewnętrzna: GitHub Actions (CI)	28
Tabela 3.4: Usługa zewnętrzna: Azure Blob Storage	28
Tabela 3.5: Usługa zewnętrzna: Mailtrap	28
Tabela 3.6: Usługa zewnętrzna: LocationIQ	28
Tabela 3.7: Usługa zewnętrzna: Google Maps (Maps URLs)	29
Tabela 3.8: Usługa zewnętrzna: OpenFreeMap	29
Tabela 3.9: Usługa zewnętrzna: Open-Meteo	29
Tabela 3.10: Usługa zewnętrzna: Tenor GIF API	30
Tabela 3.11: Usługa zewnętrzna: Where the ISS at?	30
Tabela 4.1: Scenariusz przypadku użycia: Rejestracja użytkownika	42
Tabela 4.2: Scenariusz przypadku użycia: Logowanie użytkownika	43
Tabela 4.3: Scenariusz przypadku użycia: Wykupienie subskrypcji premium	44
Tabela 4.4: Scenariusz przypadku użycia: Resetowanie hasła	45
Tabela 4.5: Scenariusz przypadku użycia: Zmiana hasła w ustawieniach konta	46
Tabela 4.6: Scenariusz przypadku użycia: Wylogowanie użytkownika	47
Tabela 4.7: Scenariusz przypadku użycia: Przeglądanie powiadomień . . .	47

Tabela 4.8: Scenariusz przypadku użycia: Przeszukiwanie historii czatu	48
Tabela 4.9: Scenariusz przypadku użycia: Przeglądanie wysłanych plików na czacie	49
Tabela 4.10: Scenariusz przypadku użycia: Zmiana typu mapy	50
Tabela 4.11: Scenariusz przypadku użycia: Przeglądanie stref PANSA	51
Tabela 4.12: Scenariusz przypadku użycia: Wyszukiwanie spota w globalnej wyszukiwarce	52
Tabela 4.13: Scenariusz przypadku użycia: Przejście do spota na mapie z wyszukiwarki	53
Tabela 4.14: Scenariusz przypadku użycia: Przeglądanie mapy spotów	53
Tabela 4.15: Scenariusz przypadku użycia: Otwarcie szczegółów spota	54
Tabela 4.16: Scenariusz przypadku użycia: Przeglądanie komentarzy do spota	55
Tabela 4.17: Scenariusz przypadku użycia: Przeglądanie pogody na spocie	56
Tabela 4.18: Scenariusz przypadku użycia: Wyszukiwanie spota na mapie	57
Tabela 4.19: Scenariusz przypadku użycia: Utworzenie prywatnego czatu	57
Tabela 4.20: Scenariusz przypadku użycia: Otworzenie czatu	58
Tabela 4.21: Scenariusz przypadku użycia: Utworzenie czatu grupowego	59
Tabela 4.22: Scenariusz przypadku użycia: Przeglądanie listy czatów	60
Tabela 4.23: Scenariusz przypadku użycia: Wysyłanie wiadomości na czacie	61
Tabela 4.24: Scenariusz przypadku użycia: Wysyłanie GIF-a na czacie	62
Tabela 4.25: Scenariusz przypadku użycia: Wysyłanie pliku na czacie	63
Tabela 4.26: Scenariusz przypadku użycia: Edycja ustawień czatu	64
Tabela 4.27: Scenariusz przypadku użycia: Dodanie członka do czatu grupowego	64
Tabela 4.28: Scenariusz przypadku użycia: Przeglądanie postów na forum	65
Tabela 4.29: Scenariusz przypadku użycia: Wyszukiwanie postów na forum	66
Tabela 4.30: Scenariusz przypadku użycia: Dodanie posta na forum	68
Tabela 4.31: Scenariusz przypadku użycia: Dodanie komentarza na forum	68
Tabela 4.32: Scenariusz przypadku użycia: Przeglądanie historii interakcji z postami	69

Tabela 4.33: Scenariusz przypadku użycia: Zarządzanie komentarzami na forum	70
Tabela 4.34: Scenariusz przypadku użycia: Zgłoszenie komentarza naruszającego regulamin	71
Tabela 4.35: Scenariusz przypadku użycia: Zgłoszenie posta na forum	72
Tabela 4.36: Scenariusz przypadku użycia: Przeglądanie komentarzy pod postem	73
Tabela 4.37: Scenariusz przypadku użycia: Dodanie spota w panelu użytkownika	74
Tabela 4.38: Scenariusz przypadku użycia: Przeglądanie profilu użytkownika	75
Tabela 4.39: Scenariusz przypadku użycia: Przeglądanie profilu innego użytkownika	76
Tabela 4.40: Scenariusz przypadku użycia: Dodanie użytkownika do znamionych	77
Tabela 4.41: Scenariusz przypadku użycia: Przeglądanie społeczności	77
Tabela 4.42: Scenariusz przypadku użycia: Przeglądanie dodanych spotów	78
Tabela 4.43: Scenariusz przypadku użycia: Edycja danych użytkownika	79
Tabela 4.44: Scenariusz przypadku użycia: Przeglądanie dodanych zdjęć do spotów	80
Tabela 4.45: Scenariusz przypadku użycia: Przeglądanie dodanych filmów do spotów	81
Tabela 4.46: Scenariusz przypadku użycia: Przeglądanie dodanych komentarzy do spotów	82
Tabela 4.47: Karta wymagania ogólnego dla czatu: Wysyłanie wiadomości na czacie	84
Tabela 4.48: Karta wymagania ogólnego dla czatu: Edycja czatu	84
Tabela 4.49: Karta wymagania ogólnego dla czatu: Przeglądanie historii czatu	84
Tabela 4.50: Karta wymagania ogólnego dla czatu: Tworzenie czatu	85
Tabela 4.51: Wymaganie funkcjonalne dla czatu: Wysyłanie GIF'ów	86

Tabela 4.52: Wymaganie funkcjonalne dla czatu: Wysyłanie plików	87
Tabela 4.53: Wymaganie funkcjonalne dla czatu: Wysyłanie wiadomości prywatnych	87
Tabela 4.54: Wymaganie funkcjonalne dla czatu: Wysyłanie wiadomości do wielu osób jednocześnie	88
Tabela 4.55: Wymaganie funkcjonalne dla czatu: Rozpoczynanie nowego czatu	89
Tabela 4.56: Wymaganie funkcjonalne dla czatu: Wysyłanie emotikonów .	89
Tabela 4.57: Wymaganie funkcjonalne dla czatu: Dostępność czatu po utworzeniu	90
Tabela 4.58: Wymaganie funkcjonalne dla czatu: Edytowanie nazwy czatu grupowego	91
Tabela 4.59: Wymaganie funkcjonalne dla czatu: Edycja zdjęcia czatu grupowego	92
Tabela 4.60: Wymaganie funkcjonalne dla czatu: Edycja wysłanej wiadomości	92
Tabela 4.61: Wymaganie funkcjonalne dla czatu: Dodawanie użytkowników do istniejącego czatu	93
Tabela 4.62: Wymaganie funkcjonalne dla czatu: Wyświetlanie starszych wiadomości	94
4.63 Profil użytkownika	95
4.64 Lista dodanych spotów	96
4.65 Dodanie spota	97
4.66 Lista zdjęć	98
4.67 Lista filmów	98
4.68 Lista znajomych	99
4.69 Lista obserwujących	99
4.70 Lista obserwowanych	100
4.71 Lista polubionych/odwiedzonych/planowanych spotów	100
4.72 Lista komentarzy	101
4.73 Ustawienia profilu	102

4.74 Resetowanie hasła	103
4.75 Dodawanie do znajomych	104
4.76 Logowanie i rejestracja	105
4.77 Strona główna — podstawowe filtry	106
4.78 Strona główna — zaawansowane filtry	107
4.79 Ustawienia motywu (ręczna zmiana)	108
4.80 Zapamiętanie preferencji motywu	109
4.81 Szybki przełącznik motywu w interfejsie	110
Tabela 4.63: Wymaganie pozafunkcjonalne dla czatu: Dostęp do czatów ograniczony do uczestników (autoryzacja)	111
Tabela 4.64: Wymaganie pozafunkcjonalne dla czatu: Korzystanie z czatu wymaga zalogowania (uwierzytelnienie)	112
Tabela 4.65: Wymaganie pozafunkcjonalne dla czatu: Grupowanie wiadomości według daty wysłania	113
Tabela 4.66: Wymaganie pozafunkcjonalne dla czatu: Wyraźne oznaczenie nadawcy i czasu wysłania	114
Tabela 4.67: Wymaganie pozafunkcjonalne dla czatu: Czas załadowania starszych wiadomości poniżej 10 sekund	114
Tabela 4.68: Wymaganie pozafunkcjonalne dla czatu: Natychmiastowe wysyłanie wiadomości	115
 Tabela 7.1: Zestawienie endpointów: panelu użytkownika	166
Tabela 7.2: Zestawienie endpointów: modułu spotów	167
Tabela 7.3: Zestawienie endpointów: komentarzy do spotów	168
Tabela 7.4: Zestawienie endpointów: postów forum	168
Tabela 7.5: Zestawienie endpointów: komentarzy forum	169
Tabela 7.6: Zestawienie endpointów: konta użytkownika i autoryzacji	169
Tabela 7.7: Zestawienie endpointów: integracji GIF-ów	170
Tabela 7.8: Zestawienie endpointów: modułu czatu	170
Tabela 7.9: Karta endpointu: /public/user-dashboard/profile/{targetUsername}	171
Tabela 7.10: Karta endpointu: /user-dashboard/favorite-spots	172
Tabela 7.11: Karta endpointu: /user-dashboard/photos	173

Tabela 7.12: Karta endpointu: /user-dashboard/add-spot	174
Tabela 7.13: Karta endpointu: /user-dashboard/add-spot	175
Tabela 7.14: Karta endpointu: /public/spot/gallery	176
Tabela 7.15: Karta endpointu: /public/spot/current-view	177
Tabela 7.16: Karta endpointu: /public/spot/get-spot-basic-weather . . .	178
Tabela 7.17: Karta endpointu: /public/spot/get-spot-detailed-weather . .	179
Tabela 7.18: Karta endpointu: /public/spot/get-spot-wind-speeds	180
Tabela 7.19: Karta endpointu: /public/spot/most-popular	181
Tabela 7.20: Karta endpointu: /public/spot/search/home-page	182
Tabela 7.21: Karta endpointu: /public/spot/search/home-page/locations	183
Tabela 7.22: Karta endpointu: /public/spot/search/home-page/advance .	184
Tabela 7.23: Karta endpointu: /public/spot/{spotId}/comments	186
Tabela 7.24: Karta endpointu: /public/spot/{spotId}/comments/{commentId}	187
Tabela 7.25: Karta endpointu: /spot/{spotId}/comments	188
Tabela 7.26: Karta endpointu: /spot/comments/{commentId}/vote . . .	189
Tabela 7.27: Karta endpointu: /spot/comments/vote-type	189
Tabela 7.28: Karta endpointu: /public/post/{postId}	191
Tabela 7.29: Karta endpointu: /post	192
Tabela 7.30: Karta endpointu: /post/{postId}	193
Tabela 7.31: Karta endpointu: /post/{postId}/vote	193
Tabela 7.32: Karta endpointu: /public/categories-tags	194
Tabela 7.33: Karta endpointu: /public/post/{postId}/comments	196
Tabela 7.34: Karta endpointu: /post/{postId}/comments	197
Tabela 7.35: Karta endpointu: /post/comments/{commentId}	198
Tabela 7.36: Karta endpointu: /post/comments/{commentId}/vote . . .	198
Tabela 7.37: Karta endpointu: /comments/{commentId}/replies	199
Tabela 7.38: Karta endpointu: /public/account/register	200
Tabela 7.39: Karta endpointu: /public/account/login	200
Tabela 7.40: Karta endpointu: /public/account/forgot-password	201
Tabela 7.41: Karta endpointu: /public/account/set-new-password	202
Tabela 7.42: Karta endpointu: /account/check	202

Tabela 7.43: Karta endpointu: /account/login-success	203
Tabela 7.44: Karta endpointu: /gifs/trending	204
Tabela 7.45: Karta endpointu: /gifs/search	205
Tabela 7.46: Karta endpointu: /chats/{chatId}/messages	206
Tabela 7.47: Karta endpointu: /chats/get-or-create-private-chat	207
Tabela 7.48: Karta endpointu: /chats/{chatId}/send-files	208
Tabela 7.49: Karta endpointu: /chats/create/group	209
Tabela 7.50: Karta endpointu: /chats/{chatId}	210
Tabela 7.51: Karta endpointu: /chats/group-chat/add/search/{chatId} .	211

Bibliografia

- [1] *Disciplined Agile Delivery*. PMI. URL: <https://www.pmi.org/disciplined-agile/process/introduction-to-dad/why> (dostęp 30.10.2025).
- [2] *Disciplined Agile Delivery — Lean Life Cycle*. PMI. 1 stycznia 2025. URL: <https://www.pmi.org/disciplined-agile/lifecycle/lean-lifecycle> (dostęp 30.10.2025).
- [3] Stanisław Wrycza, Bartosz Marcinkowski i Krzysztof Wyrzykowski. „Język UML 2.0 w modelowaniu systemów informatycznych”. Warszawa: Helion, 2006. ISBN: 83-736-1892-9, 8373618929.
- [4] Michał Wolski. *10 wskazówek poprawiających modelowanie procesów biznesowych w notacji BPMN*. 14 maja 2024. URL: <https://wolski.pro/2024/05/10-wskazowek-poprawiajacych-modelowanie-procesow-biznesowych-w-notacji-bpmn/> (dostęp 19.11.2025).
- [5] *About billing for GitHub Actions*. GitHub Docs. 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] *React useState*. URL: <https://react.dev/reference/react/useState> (dostęp 3.11.2025).
- [15] *Redux*. URL: <https://redux.js.org/> (dostęp 3.11.2025).
- [16] *Axios*. URL: <https://axios-http.com/> (dostęp 3.11.2025).
- [17] *Tanstack Query*. URL: <https://tanstack.com/query/latest> (dostęp 3.11.2025).
- [18] *Tailwind*. URL: <https://tailwindcss.com/> (dostęp 3.11.2025).
- [19] *Motion*. URL: <https://motion.dev/> (dostęp 3.11.2025).
- [20] *RFC 6455: The WebSocket Protocol*. IETF. URL: <https://datatracker.ietf.org/doc/html/rfc6455> (dostęp 16.12.2025).
- [21] *WebSocket (Spring Framework Reference Documentation)*. Spring. URL: <https://docs.spring.io/spring-framework/reference/web/websocket.html> (dostęp 16.12.2025).
- [22] *STOMP over WebSocket (Spring Framework Reference Documentation)*. Spring. URL: <https://docs.spring.io/spring-framework/reference/web/websocket/stomp.html> (dostęp 16.12.2025).
- [23] *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.