



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: spoty-na-drony.pl

Rodzaj pracy

inżynierska

Imię i nazwisko promotora

mgr Adam Urbanowicz

Gdańsk, miesiąc, 2100 obrony

Streszczenie: Celem niniejszej pracy było stworzenie w pełni funkcjonalnej i działającej aplikacji internetowej pozwalającej na szybkie wyszukiwanie spotów w okolicy oraz dzielenie się zdjęciami, filmami oraz doświadczeniem z innymi użytkownikami. W ramach pracy stworzono system składający się z trzech komponentów: Frontendu, Backendu oraz bazy-danych. Aplikacja internetowa została wykonana przy pomocy Frameworka React w językach Javascript oraz Typescript, do 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: spotty-na-drony.pl Temat projektu po angielsku: Web application: spotty-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: 19 listopada 2025	Recenzent: dr Elżbieta Puźniakowska-Gałuch
---	--

Spis treści

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

3.3.2	Narzędzia	14
3.4	Zasoby i ograniczenia	14
3.4.1	Zasoby	14
3.4.2	Ograniczenia	15
3.4.3	Usługi zewnętrzne	15
3.5	Analiza ryzyka	17
4	Analiza wymagań	18
4.1	Przypadki użycia	18
4.1.1	Aktorzy	18
4.1.2	Diagram przypadków użycia	20
4.1.3	Scenariusz przypadków użycia	20
4.2	Wymagania ogólne i dziedzinowe	20
4.3	Wymagania funkcjonalne	20
4.3.1	Funkcjonalności dla mapy	20
4.3.2	Funkcjonalności dla chatu	20
4.3.3	Funkcjonalności dla forum	20
4.3.4	Funkcjonalności dla konta użytkownika	20
4.3.5	Funkcjonalności dla logowania i rejestracji	30
4.3.6	Funkcjonalności dla wyszukiwarki spotów	31
4.3.7	Funkcjonalności dla motywu	33
4.4	Wymagania pozafunkcjonalne	35
4.5	Wymagania interfejs z otoczeniem	35
4.6	Wymagania na środowisko docelowe	35
5	Projekt	36
5.1	Wzorce projektowe	36
5.2	Architektura systemu	36
5.2.1	Diagram architektury	36
5.2.2	Komponenty systemu	36
5.3	Projekt bazy danych	36
5.3.1	Model danych	36

5.3.2	Diagram ERD	36
5.4	Architektura interfejsu użytkownika	36
5.4.1	Projekt strony głównej	36
5.4.2	Projekt panelu logowania	36
5.4.3	Projekt mapy	36
5.4.4	Projekt chatu	36
5.4.5	Projekt forum	36
5.4.6	Projekt konta użytkownika	36
6	Przebieg realizacji projektu	37
6.1	Sprint 1	37
6.2	Sprint 2	37
7	Realizacja Projektu	38
7.1	Implementacja backendu	38
7.1.1	Struktura projektu	38
7.1.2	Integracja z bazą danych	38
7.1.3	Obsługa uwierzytelnienia	38
7.1.4	Konteneryzacja	38
7.2	Implementacja frontendu	38
7.2.1	Struktura aplikacji	38
7.2.2	Zarządzanie stanem i przepływ danych	43
7.2.3	Integracja i komunikacja z backendem	45
7.2.4	Style	48
7.2.5	Strona główna	52
7.2.6	Mapa	60
7.2.7	Chat	60
7.2.8	Forum	60
7.2.9	Konto użytkownika	60
7.2.10	Panel logowania	61
7.3	Implementacja CI/CD	61

8 Testy	62
8.1 Testy jednostkowe	62
8.2 Testy integracyjne	62
8.3 Testy E2E	62
8.4 Wyniki testów i wnioski	62
9 Prezentacja systemu	63
9.1 Strona główna	63
9.2 Strona mapy	63
9.3 Strona chatu	63
9.4 Strona forum	63
9.5 Panel logowania	63
9.6 Panel konta użytkownika	63
10 Nakład pracy	64
10.1 Ogólny nakład pracy	64
10.2 Indywidualne nakłady pracy	64
10.2.1 Adam Langmesser	64
10.2.2 Mateusz Redosz	64
10.2.3 Stanisław Oziemczuk	67
10.2.4 Kacper Badek	67
11 Podsumowanie	68
11.1 Osiągnięte rezultaty	68
11.2 Napotkane wyzwania	68
11.3 Plany na przyszłość	68
12 Słownik pojęć i skrótów	69
Spis tabel	73
Bibliografia	75
Załączniki	77

Rozdział 1

Wstęp

1.1 O projekcie

1.2 Cel i zakres prac

1.3 Geneza pomysłu

Rozdział 2

Opis problemu

2.1 Rich picture

2.2 Udziałowcy

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

Tabela 2.1: Zespół projektowy

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

Tabela 2.2: Promotor

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

Tabela 2.3: Droniarze

- 2.3 Istniejące rozwiązania**
- 2.4 Wizja rozwiązania**
- 2.5 Aspekty społeczne i biznesowe**
 - 2.5.1 Aspekty społeczne**
 - 2.5.2 Aspekty biznesowe**

Rozdział 3

Planowanie

3.1 Metodologia pracy

3.1.1 Przegląd rozważanych podejść

Przy wyborze metodologii pracy rozważono trzy podejścia do prowadzenia projektu informatycznego:

- klasyczny Agile (w praktyce: Scrum),
- model kaskadowy (Waterfall),
- Disciplined Agile Delivery - Lean Life Cycle.

3.1.2 Odrzucone podejścia

„Klasyczny Agile” (Scrum). Mimo elastyczności i popularności zakłada pracę w iteracjach 2–4 tygodni oraz stały zestaw ceremonii (planowanie, przegląd, retrospektyna). Ze względu na nierównomierną dostępność zasobów w kolejnych miesiącach studiów nie zapewniono możliwości utrzymania stałej kadencji sprintów, dlatego z podejścia zrezygnowano.

Model kaskadowy (Waterfall). Przewiduje sekwencyjne przechodzenie przez z góry określone etapy i ogranicza bieżącą weryfikację wymagań w trakcie prac deweloperskich. W projekcie wymagano możliwości częstych rewizji założeń oraz

wprowadzania istotnych zmian w docelowej wizji rozwiązania; dlatego z podejścia zrezygnowano.

3.1.3 Wybrane podejście: Disciplined Agile Delivery (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 promotora oraz możliwość korygowania kierunku bez kosztów „przeskakiwania” między fazami.
- **Praktyki Lean i koncentracja na wartości.** Priorytetyzacja wartości biznesowej, wizualizacja pracy, małe partie dostaw.
- **Lekka governance i kamienie milowe.** Zastosowano lekkie mechanizmy nadzoru (peer review, prezentacje postępów) zapewniające przejrzystość bez nadmiernej biurokracji.

3.1.4 Narzędzia i komunikacja

Do zarządzania zadaniami zastosowana została **Jira** (monitorowanie postępu prac oraz ewidencja zadań członków zespołu). Komunikację w zespole zaplanowano w

formie regularnych spotkań oraz asynchronicznie z wykorzystaniem **Discorda** oraz **Messengera**.

3.1.5 Podział ról w zespole

- Adam - fullstack developer, lider zespołu
- Stanisław - fullstack developer
- Kacper - fullstack developer
- Mateusz - fullstack developer

Każdy z członków zespołu uczestniczy również w przygotowaniu dokumentacji.

3.2 Harmonogram projektu

W poniższym harmonogramie przedstawiono plan prac nad poszczególnymi częściami projektu, rozłożony na miesiące.

Rok 2024

Czerwiec • Zebranie zespołu.

- Rozważenie potencjalnych pomysłów.

Lipiec • Wybór technologii.

- Wstępne założenia architektoniczne.

Sierpień • *(do uzupełnienia)*

- *(do uzupełnienia)*

Wrzesień • *(do uzupełnienia)*

- *(do uzupełnienia)*

Październik • *(do uzupełnienia)*

- *(do uzupełnienia)*

Listopad • (do uzupełnienia)

- (do uzupełnienia)

Grudzień • (do uzupełnienia)

- (do uzupełnienia)

Rok 2025

Styczeń • (do uzupełnienia)

- (do uzupełnienia)

Luty • (do uzupełnienia)

- (do uzupełnienia)

Marzec • (do uzupełnienia)

- (do uzupełnienia)

Kwiecień • (do uzupełnienia)

- (do uzupełnienia)

Maj • (do uzupełnienia)

- (do uzupełnienia)

Czerwiec • (do uzupełnienia)

- (do uzupełnienia)

Lipiec • (do uzupełnienia)

- (do uzupełnienia)

Sierpień • (do uzupełnienia)

- (do uzupełnienia)

Wrzesień • (do uzupełnienia)

- (do uzupełnienia)

Październik • (*do uzupełnienia*)

- (*do uzupełnienia*)

Listopad • (*do uzupełnienia*)

- (*do uzupełnienia*)

Grudzień • (*do uzupełnienia*)

- (*do uzupełnienia*)

Rok 2026

Styczeń • (*do uzupełnienia*)

- (*do uzupełnienia*)

3.3 Technologie i narzędzia

3.3.1 Technologie

3.3.2 Narzędzia

3.4 Zasoby i ograniczenia

3.4.1 Zasoby

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

3.4.2 Ograniczenia

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

3.4.3 Usługi zewnętrzne

Usługa	GitHub Actions (CI) [3]
Opis	Uruchomienia pipeline'ów CI/CD dla repozytorium GitHub.
Limit	3000 min/mies.

Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)

Usługa	Azure Blob Storage [4]
Opis	Magazyn plików (m.in. zdjęcia spotów, załączniki z czatu).
Limit	1 GB/mies.

Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage

Usługa	Mailtrap [5]
Opis	Środowisko testowe SMTP oraz Email API do wysyłki maili.
Limit	150 maili/dzień

Tabela 3.3: Usługa zewnętrzna: Mailtrap

Usługa	LocationIQ [6]
Opis	Geokodowanie adresu przy dodawaniu nowych spotów.
Limit	5 000 zapytań/dzień

Tabela 3.4: Usługa zewnętrzna: LocationIQ

Usługa	Google Maps (Maps URLs) [7]
Opis	Otwieranie nawigacji w aplikacji Map Google (deep link/URL).
Limit	Brak limitu w ramach dokumentowanego sposobu użycia.

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

Usługa	OpenFreeMap [8, 9]
Opis	Publiczny serwer kafelków do renderu mapy na froncie.
Limit	30 000 zapytań/mies.

Tabela 3.6: Usługa zewnętrzna: OpenFreeMap

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

Tabela 3.7: Usługa zewnętrzna: Open-Meteo

Usługa	Tenor GIF API [11]
Opis	Wyszukiwanie GIF-ów w czacie.
Limit	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.8: Usługa zewnętrzna: Tenor GIF API

Usługa	Where the ISS at? [12]
Opis	HTTP API z bieżącą pozycją satelity, używane pomocniczo.
Limit	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.9: Usługa zewnętrzna: Where the ISS at?

3.5 Analiza ryzyka

Rozdział 4

Analiza wymagań

4.1 Przypadki użycia

4.1.1 Aktorzy

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

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

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

Moderator Moderacja treści: posty na forum, komentarze spotów.

Deweloper Rozwija i utrzymuje system.

Aktorzy będący zewnętrznymi usługami Poniżej wymieniono aktorów, których opisy zamieszczono w rozdziale poświęconym integracji z zewnętrznymi usługami 3.4.3.

- Usługa mailowa (Mailtrap)
- Dostawca API do map (OpenFreeMap)

- Nawigacja (Google Maps)
- Dostawca API pogodowego (Open-Meteo)
- Dostawca API GIFów (Tenor)
- Dostawca API do określania strefy czasowej spota (“Where the ISS at?”)
- Dostawca API do geolokalizacji (LocationIQ)
- Azure Blob Storage
- Dostawca OAuth (Google)
- Dostawca OAuth (GitHub)

4.1.2 Diagram przypadków użycia

4.1.3 Scenariusz przypadków użycia

4.2 Wymagania ogólne i dziedzinowe

4.3 Wymagania funkcjonalne

4.3.1 Funkcjonalności dla mapy

4.3.2 Funkcjonalności dla chatu

4.3.3 Funkcjonalności dla forum

4.3.4 Funkcjonalności dla konta użytkownika

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

Tabela 4.1: Profil użytkownika

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

Tabela 4.2: Lista dodanych spotów

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

Tabela 4.3: Dodanie spota

KARTA WYMAGANIA		
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet: M
Nazwa:	Lista zdjęć	
Opis:	Jako użytkownik chcę mieć dostęp do listy zdjęć, które dodałem na forum, do komentarzy pod 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 4.4: Lista zdjęć

KARTA WYMAGANIA		
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet: M
Nazwa:	Lista filmów	
Opis:	Jako użytkownik chcę mieć dostęp do listy filmów, które dodałem na forum, do komentarzy pod 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 4.5: Lista filmów

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

Tabela 4.6: Lista znajomych

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

Tabela 4.7: Lista obserwujących

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

Tabela 4.8: Lista obserwowanych

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

Tabela 4.9: Lista polubionych/odwiedzonych/planowanych spotów

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

Tabela 4.10: Lista komentarzy

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

Tabela 4.11: Ustawienia profilu

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

Tabela 4.12: Resetowanie hasła

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

Tabela 4.13: Dodawanie do znajomych

4.3.5 Funkcjonalności dla logowania i rejestracji

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

Tabela 4.14: Logowanie i rejestracja

4.3.6 Funkcjonalności dla wyszukiwarki spotów

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

Tabela 4.15: Strona główna — podstawowe filtry

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Strona główna z zaawansowanymi filtrami		
Opis:	Jako użytkownik chcę mieć dostęp do strony głównej, która wyświetla listę spotów, które można filtrować i sortować.		
Kryteria akceptacji:	Użytkownik widzi listę, którą może filtrować według miasta, tagów i oceny 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 4.16: Strona główna — zaawansowane filtry

4.3.7 Funkcjonalności dla motywu

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Ustawienia motywu		
Opis:	Jako użytkownik chcę móc zmienić motyw aplikacji.		
Kryteria akceptacji:	Dostępna jest opcja przełączenia motywu na <i>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 <code>darkMode: 'class'</code> ; motyw przełączany przez dodanie/usunięcie klasy <code>dark</code> na elemencie <code><html></code> ;		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, drona-rze 2.3.		
Wymagania powiązane:			

Tabela 4.17: Ustawienia motywu (ręczna zmiana)

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

Tabela 4.18: 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 <i>dark</i> na <i>document.documentElement</i> oraz aktualizuje <i>localStorage (theme = 'dark' 'light')</i> ; brak przeładowania strony.
Udziałowiec:		Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.
Wymagania powiązane:		

Tabela 4.19: Szybki przełącznik motywów w interfejsie

4.4 Wymagania pozafunkcjonalne

4.5 Wymagania interfejs z otoczeniem

4.6 Wymagania na środowisko docelowe

Rozdział 5

Projekt

5.1 Wzorce projektowe

5.2 Architektura systemu

5.2.1 Diagram architektury

5.2.2 Komponenty systemu

5.3 Projekt bazy danych

5.3.1 Model danych

5.3.2 Diagram ERD

5.4 Architektura interfejsu użytkownika

5.4.1 Projekt strony głównej

5.4.2 Projekt panelu logowania

5.4.3 Projekt mapy

5.4.4 Projekt chatu

5.4.5 Projekt forum

5.4.6 Projekt konta użytkownika

Rozdział 6

Przebieg realizacji projektu

6.1 Sprint 1

6.2 Sprint 2

Rozdział 7

Realizacja Projektu

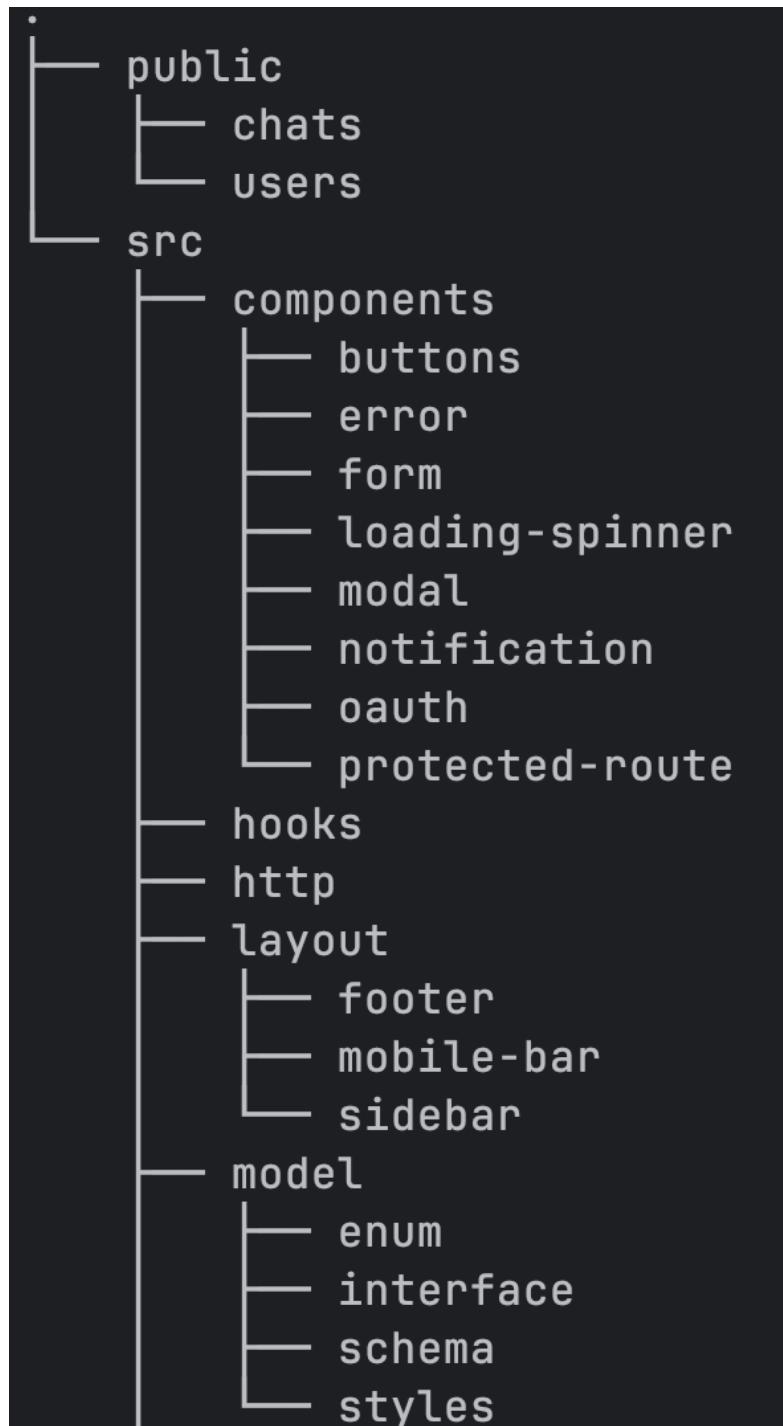
7.1 Implementacja backendu

- 7.1.1 Struktura projektu
- 7.1.2 Integracja z bazą danych
- 7.1.3 Obsługa uwierzytelnienia
- 7.1.4 Konteneryzacja

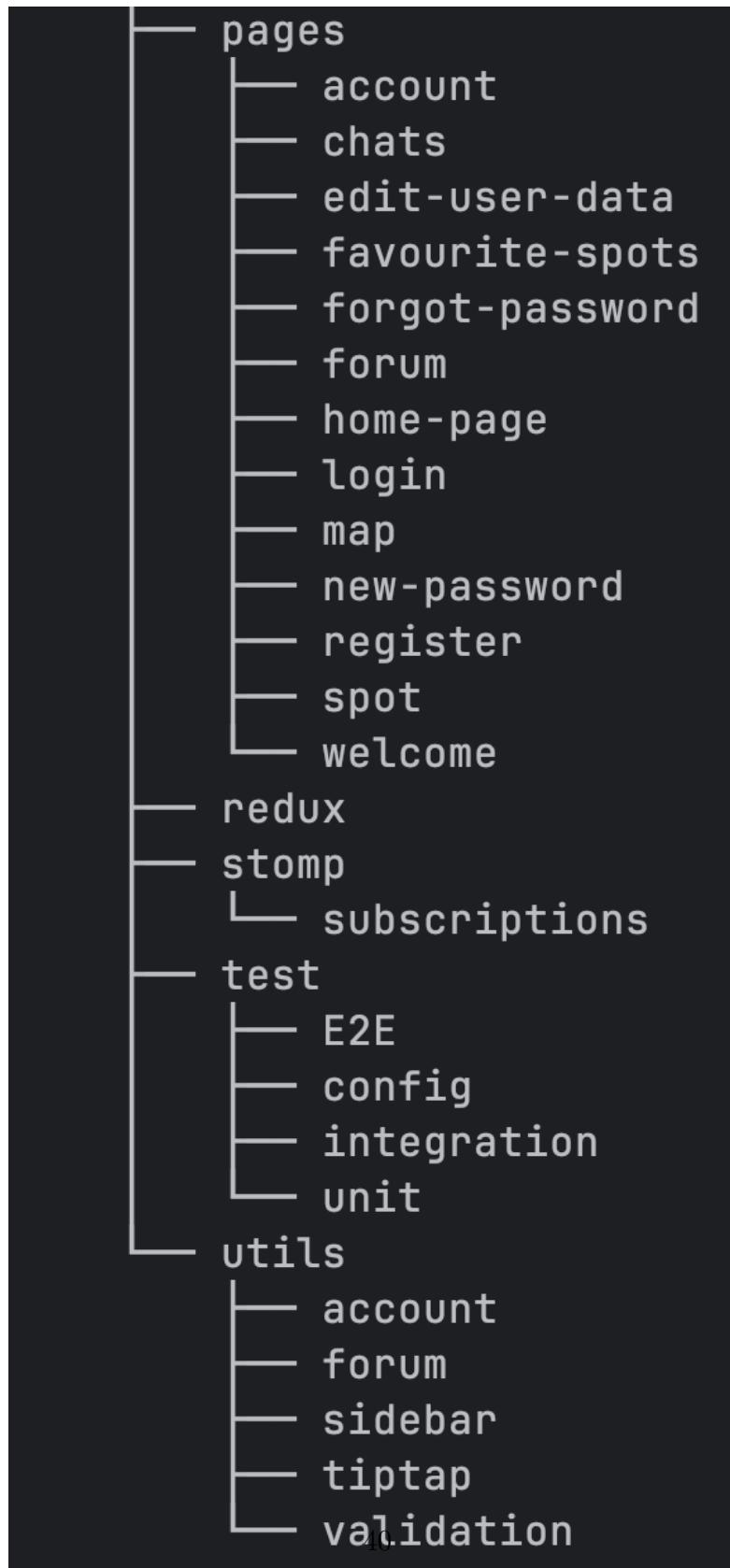
7.2 Implementacja frontendu

7.2.1 Struktura aplikacji

Architektura aplikacji frontendowej została zaprojektowana w strukturze Folder by type, która polega na podziale kodu według typu zasobu (komponenty, strony, modele itd.). Każdy plik znajduje się w katalogu odpowiadającym jego przeznaczeniu, co jest przedstawione na rysunkach 7.1 oraz 7.2.



Rysunek 7.1: Struktura katalogów (1)



Rysunek 7.2: Struktura katalogów (2)

Głównym elementem aplikacji jest mechanizm routingu oparty na Bibliotece React Router. Definiuje on ścieżki do poszczególnych funkcjonalności aplikacji. Dzięki temu możliwa jest płynna nawigacja między różnymi widokami bez konieczności przeładowywania strony.

```
const router : Router = createBrowserRouter([
  {
    path: "/",
    element: <Layout />,
    errorElement: <Error error={undefined} />,
    children: [
      {
        index: true,
        element: <HomePage />,
      },
      {
        path: "advanced",
        element: <AdvanceHomePage />,
      },
      {
        path: "account",
        children: [ 11 elements... ],
      },
      {
        path: "register",
        element: <Register />,
      },
      {
        path: "login",
        element: <Login />,
      },
      {
        path: "forgot-password",
        element: <ForgotPassword />,
      },
    ],
  },
])
```

Rysunek 7.3: Implementacja routera (1)

```
        {
          path: "new-password",
          element: <NewPassword />,
        },
        {
          path: "forum",
          element: <Forum />,
        },
        {
          path: "forum/:postId/:slugTitle?",
          element: <ForumThread />,
        },
        {
          path: "map",
          element: <MapPage />,
        },
        {
          path: "chat",
          element: (
            <ProtectedRoute>
              <ChatsPage />
            </ProtectedRoute>
          ),
        },
      ],
    );
  }

export default router;
```

Rysunek 7.4: Implementacja routera (2)

W projekcie zastosowano również wzorzec Protected route, który służy do zabezpieczania wybranych tras przed dostępem użytkowników niezalogowanych. W pliku `router.tsx`, znajdującym się w głównym katalogu projektu, w konfiguracji przekazywanej do funkcji `createBrowserRouter` (rysunki 7.3 oraz 7.4), wybrane

ścieżki zostały opakowane w komponent `ProtectedRoute`. Komponent ten pełni rolę bramki (rysunek 7.5).

Przykładem takiej chronionej ścieżki jest trasa `/chat`, prowadząca do modułu czatu dostępnego wyłącznie dla zalogowanych użytkowników. Jeśli niezalogowany użytkownik spróbuje uzyskać dostęp do tej ścieżki, zostanie automatycznie przekierowany na stronę główną.

```
export default function ProtectedRoute({ children }) {
  const isLoggedIn = useSelector((state) => state.account.isLoggedIn);

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

Rysunek 7.5: Implementacja komponentu bramki (`ProtectedRoute`)

7.2.2 Zarządzanie stanem i przepływ danych

W projekcie postawiliśmy na zrównoważone podejście do zarządzania Stanem. Korzystamy zarówno z lokalnego Stanu komponentów (za pomocą Hook (React) `useState`) [13], jak i ze Stanu globalnego, utrzymywanego przez Bibliotekę React Redux [14]. Globalny Stan został wprowadzony po to, aby możliwie najbardziej ograniczyć przekazywanie Propsów w głąb drzewa komponentów oraz uniknąć niepotrzebnych ponownych renderów.

Do przechowywania Stanu lokalnego, ograniczonego tylko do danego komponentu (lub jego najbliższych elementów podległych), wykorzystujemy Hook (React) `useState`. Natomiast efekty uboczne i synchronizację realizujemy za pomocą `useEffect`. W przypadku bardziej złożonej logiki lub potrzeby ponownego wykorzystania kodu powstały Hook (React)i niestandardowe, takie jak `useScreenSize`, `useDarkMode` czy `useClickOutside`. Dzięki temu większość logiki prezentacji została wydzielona z warstwy UI, co poprawia czytelność i ułatwia utrzymanie kodu.

Z racji tego, że korzystamy z Reacta w połączeniu z TypeScriptem, przygotowaliśmy również własne Hook (React)i wspomagające typowanie, takie jak `useDispatchTyped` oraz `useSelectorTyped`. Pozwalają one na bezpieczne typo-

wanie akcji oraz selektorów Reduxa bez konieczności powtarzania adnotacji typów w każdym komponencie. Fragmenty tej implementacji przedstawiono na rysunkach 7.6 oraz 7.7.

```
const store : EnhancedStore<{ account: AccountSliceProp... }> = configureStore({
  reducer: {
    account: accountSlice.reducer,
    notification: notificationSlice.reducer,
    spotDetails: spotDetailsModalSlice.reducer,
    searchedSpotsListModal: searchedSpotListModalSlice.reducer,
    expandedSpotMediaGallery: expandedSpotMediaGallerySlice.reducer,
    spotFilters: spotFiltersSlice.reducer,
    chats: chatsSlice.reducer,
    map: mapSlice.reducer,
    sidebar: sidebarSlice.reducer,
    searchedSpots: searchedSpotsSlice.reducer,
    social: socialSlice.reducer,
    spotComments: spotCommentSlice.reducer,
    currentViewSpots: currentViewSpotsSlice.reducer,
    currentViewSpotsListModal: currentViewSpotsListModalSlice.reducer,
    currentViewSpotsParams: currentViewSpotParamsSlice.reducer,
    spotWeather: spotWeatherSlice.reducer,
    expandedSpotGalleryMediaList: expandedSpotGalleryMediaListSlice.reducer,
    expandedSpotMediaGalleryModals: {
      expandedSpotMediaGalleryModalsSlice.reducer,
    },
    expandedSpotMediaGalleryFullscreenSizeModal: {
      expandedSpotMediaGalleryFullscreenSizeSlice.reducer,
    },
    expandedSpotGalleryCurrentMedia: {
      expandedSpotGalleryCurrentMediaSlice.reducer,
    },
  },
});

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

Rysunek 7.6: Konfiguracja sklepu (Redux store)

```

interface AccountSliceProps { Show usages & Mredosz +1
  isLoggedIn: boolean;
  username: string;
}

const initialState: AccountSliceProps = {
  isLoggedIn: localStorage.getItem("is_logged_in") === "true",
  username: localStorage.getItem("username") || "",
};

export const accountSlice : Slice<AccountSliceProps, { setisLoggedIn(st... } = createSlice({ Show usages & Mredosz +1
  name: "account",
  initialState,
  reducers: {
    setIsLoggedIn(state : WritableDraft<AccountSliceProps> ) : void { Show usages & Mredosz +1
      localStorage.setItem("is_logged_in", "true");
      state.isLoggedIn = true;
    },
    signOut(state : WritableDraft<AccountSliceProps> ) : void { Show usages & Mredosz +1
      localStorage.removeItem("is_logged_in");
      localStorage.removeItem("username");
      state.isLoggedIn = false;
      state.username = "";
    },
    setUsername(state : WritableDraft<AccountSliceProps> , action: PayloadAction<string>) : void { Show usages & Mredosz +1
      localStorage.setItem("username", action.payload);
      state.username = action.payload;
    },
  },
},
);

export const accountAction : CaseReducerActions<{ setisLoggedIn(state: W... } = accountSlice.actions; Show usages & Mredosz

```

Rysunek 7.7: Przykładowy slice odpowiedzialny za sprawdzenie czy użytkownik jest zalogowany

7.2.3 Integracja i komunikacja z backendem

Jest to kluczowy element aplikacji, ponieważ wymaga bezpiecznego przesyłania danych użytkownika. W celu uproszczenia komunikacji z serwerem skorzystaliśmy z biblioteki `axios` [15] oraz Biblioteki `TanStack Query` [16]. We wszystkich ścieżkach, które wymagają aby użytkownik był zalogowany, do zapytania dołączany jest token JWT. Token jest przekazywany w ciasteczku dzięki ustawieniu parametru `withCredentials` na wartość `true`. Przykładem pliku odpowiedzialnego za taką komunikację jest `account.js` (rys. 7.8 i 7.9), który obsługuje operacje związane z

logowaniem, rejestracją, zmianą hasła oraz wylogowaniem.

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

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

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

Rysunek 7.8: Implementacja modułu account (1)

```

export async function changePassword(userData) { Show usages  ↳ stanoz +1
    return await axios.post(
        `${BASE_URL}/public/account/set-new-password`,
        userData,
    );
}

export async function logout() { Show usages  ↳ stanoz +1
    await axios.post(
        `${BASE_URL}/account/oauth2/logout`,
        {},
        {
            withCredentials: true,
        },
    );
}

export const googleLoginUrl = `${BASE_URL}/oauth2/authorization/google`; Show usages  ↳ stanoz
export const githubLoginUrl = `${BASE_URL}/oauth2/authorization/github`; Show usages  ↳ stanoz

```

Rysunek 7.9: Implementacja modułu `account` (2)

Funkcje odpowiedzialne za komunikację z backendem zostały umieszczone w katalogu `/http`. Dzięki temu są one skoncentrowane i mogą być w prosty sposób wykorzystywane w różnych częściach aplikacji. Zastosowaliśmy TanStack Query, ponieważ znaczco ogranicza on powtarzalny kod oraz upraszcza obsługę błędów i stanów zapytania (takich jak ładowanie danych, błąd, sukces). udostępnia m.in. wartość `isLoading`, dzięki czemu komponent może łatwo wyświetlić ekran ładowania bez ręcznego zarządzania własnym stanem. Dodatkowo Hook (React) `useQuery` z tej Biblioteki umożliwia automatyczne pobieranie danych po wejściu na daną podstronę. Oznacza to, że komponent deklaruje jedynie „jakie dane są mu potrzebne”, a TanStack Query zajmuje się ich pobraniem, cache’owaniem oraz odświeżaniem. Do operacji, które wymagają wywołania akcji po stronie użytkownika (np. wysłania formularza logowania), wykorzystujemy Hook (React) `useMutation` z TanStack Query. Przykład użycia tego rozwiązania w procesie logowania został przedstawiony na rys. 7.10.

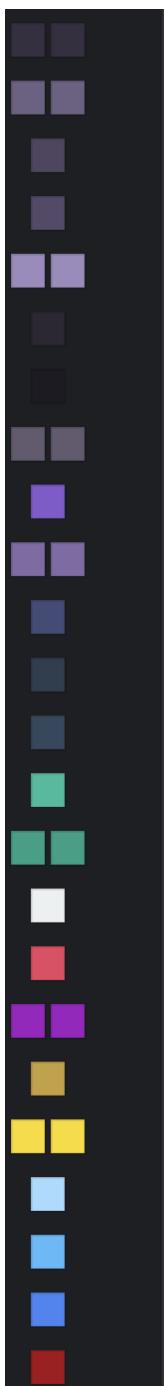
```
const { mutateAsync, isSuccess, error } = useMutation({
  mutationFn: loginUser,
});

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

Rysunek 7.10: Wykorzystanie TanStack Query przy logowaniu użytkownika

7.2.4 Style

Do stylowania interfejsu wykorzystaliśmy Framework Tailwind CSS [17]. Dzięki gotowym klasom udostępnianym przez Tailwind mogliśmy definiować wygląd elementów bezpośrednio w kodzie komponentu, bez konieczności przechodzenia do osobnych plików ze stylami. Ułatwia to zarówno tworzenie widoków, jak i późniejsze modyfikacje — w przypadku zmiany stylu dokładnie wiadomo, gdzie należy jej dokonać. Korzystanie ze zdefiniowanych klas pozwoliło nam również zachować spójność wizualną w całej aplikacji. W pliku `index.css` zdefiniowaliśmy zmienne kolorystyczne (rys. 7.11 i 7.12). Dzięki temu zmiana motywu kolorystycznego w przyszłości sprowadza się do edycji wartości w jednym miejscu.



	--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: #fadbd1;
	--color-locationMarkerDarkerBlue: #a3dcff;
	--color-locationMarkerLightBlue: #52bafb;
	--color-userLocationDot: #4285f4;
	--color-spotLocationMarker: #a8071a;

Rysunek 7.11: Implementacja zmiennych kolorystycznych (1)



Rysunek 7.12: Implementacja zmiennych kolorystycznych (2)

W niektórych miejscach konieczne było zapisanie stylów w czystym CSS, ponieważ część użytych Bibliotek tego wymagała. W innych przypadkach wystarczyło skorzystać z klas zdefiniowanych w `index.css` oraz klas Tailwinda. Cała aplikacja

jest Responsywna. Tailwind udostępnia predefiniowane prefiksy Responsywne (np. `md:`, `lg:`) (rys. 7.13), stworzyliśmy również własny (`3xl:`) na ekrany o rozdzielczości 2560px. Pozwalają one przypisywać style zależnie od szerokości ekranu bez pisania własnych reguł @media. Dzięki temu implementacja widoków mobilnych i desktopowych była 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 7.13: Przykładowe użycie klas Tailwind (w tym prefiksów responsywności)

Tailwind został też wykorzystany do obsługi trybu jasnego i ciemnego. Wystarczy dodać klasę z prefiksem `dark:` (np. `dark:bg-black`), aby zmienić kolorystykę elementu, gdy aplikacja jest w trybie ciemnym (rys. 7.14).

```
<input
  id={id}
  value={value}
  type={type}
  onChange={onChange}
  onFocus={setFocusedToTrue}
  onBlur={handleOnBlur}
  className="dark:bg-darkBgMuted bg-lightBgMuted dark:text-darkText text-lightText w-full
  rounded-md p-2 shadow-md focus:outline-none dark:shadow-black/50"
/>
```

Rysunek 7.14: Przykładowe użycie klas Tailwind (w tym wariantu `dark:`)

Aby uzyskać płynniejsze i przyjemniejsze animacje, wykorzystaliśmy Bibliotekę Motion [18]. Pozwala ona w prosty sposób tworzyć animacje elementów interfejsu, bez potrzeby ręcznego pisania złożonych reguł CSS. W naszej aplikacji użyliśmy jej m.in. w polach formularza logowania i rejestracji (rys. 7.15). Na początku etykieta pola (np. „username”) jest wyświetlana wewnątrz pola tekstowego, natomiast po

kliknięciu w pole jest płynnie przesuwana nad to pole, co poprawia czytelność i ergonomię formularza.

```
<div className="relative">
  <motion.label
    htmlFor={id}
    initial={false}
    animate={{
      top: shouldFloat ? "-0.7rem" : "0.5rem",
      left: "0.75rem",
      fontSize: shouldFloat ? "0.75rem" : "1rem",
      opacity: shouldFloat ? 1 : 0.6,
    }}
    transition={{ type: "spring", stiffness: 300, damping: 25 }}
    className="■dark:text-darkText □text-lightText pointer-events-none absolute z-10 px-1 capitalize"
  >
    {label}
  </motion.label>
  <input
    id={id}
    value={value}
    type={type}
    onChange={onChange}
    onFocus={setFocusedToTrue}
    onBlur={handleOnBlur}
    className="■ dark:bg-darkBgMuted □bg-lightBgMuted ■dark:text-darkText □text-lightText w-full
    rounded-md p-2 shadow-md focus:outline-none dark:shadow-black/50"
  />
```

Rysunek 7.15: Implementacja animacji z wykorzystaniem Motion

7.2.5 Strona główna

Jednym z głównych modułów aplikacji jest strona główna. Pełni ona rolę centralnej wyszukiwarki Spotów, dzięki której użytkownik może w prosty sposób znaleźć interesujące go lokalizacje. Strona działa w dwóch trybach: prostym oraz zaawansowanym (rys. 7.16 i 7.17).

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

Rysunek 7.16: Implementacja prostej wersji strony głównej

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

Rysunek 7.17: Implementacja zaawansowanej wersji strony głównej

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

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

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

```

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

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

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

```

Rysunek 7.19: Implementacja karuzeli z najpopularniejszymi spotami

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

Strona główna 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. 7.18) zawiera dwa elementy `NavLink` z Biblioteki React Router, co pozwala na przełączanie widoków bez konieczności przeładowywania całej strony.

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

```

<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 7.20: Implementacja prostej wyszukiwarki

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

```

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

```

Rysunek 7.21: Implementacja listy do wyświetlania spotów

Komponent **SpotTile** zawiera następujące informacje:

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

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

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

7.2.6 Mapa

7.2.7 Chat

7.2.8 Forum

7.2.9 Konto użytkownika

Kolejnym z głównych modułów aplikacji jest panel użytkownika. Został on podzielony na 8 zakładek:

- Profile - profil użytkownika oraz możliwość zmiany zdjęcia profilowego,
- Spots - lista z spotami dodanymi do list (np. ulubionych, do ponownego odwiedzenia),
- Photos - lista zdjęć które użytkownik dodał do spotów, komentarzy pod spotami oraz na forum,
- Movies - lista filmów które użytkownik dodał do spotów, komentarzy pod spotami oraz na forum,
- Social - listy z znajomymi, obserwowanymi i obserwującymi,
- Add spot - lista ze spotami które dodał użytkownik oraz formularz do dodania nowego spota,
- Comments - lista komentarzy które użytkownik dodał pod spotami,
- Settings - ustawienia użytkownika (zmiana: nazwy użytkownika, e-mail, hasła),

Profil użytkownika został zbudowany z dwóch głównych komponentów (ProfileForViewer oraz UserOwnProfile). W obu jest używany reuzywalny komponent Profile dzięki czemu widok w większości jest identyczny, własnego profilu oraz profilu innego użytkownika, na profilu innego użytkownika dodatkowo wyświetla się dwa przyciski jeden do dodania bądź usunięcia ze znajomych, drugi do obserwowania oraz usunięcia z obserwowanych. W komponencie Profile znajduje się zdjęcie użytkownika, po najechaniu na zdjęcie pojawia się przycisk do zmiany zdjęcia. Są również cztery przyciski do przejścia do znajomych, obserwowanych, obserwujących oraz zdjęć danego użytkownika. Na samym dole znajduje się lista czterech najpopularniejszych zdjęć danego użytkownika.

7.2.10 Panel logowania

7.3 Implementacja CI/CD

Rozdział 8

Testy

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

Rozdział 9

Prezentacja systemu

9.1 Strona główna

9.2 Strona mapy

9.3 Strona chatu

9.4 Strona forum

9.5 Panel logowania

9.6 Panel konta użytkownika

Rozdział 10

Nakład pracy

10.1 Ogólny nakład pracy

10.2 Indywidualne nakłady pracy

10.2.1 Adam Langmesser

10.2.2 Mateusz Redosz

Na projekt poświęciłem łącznie 324 godziny, z czego 237 przeznaczyłem na prace deweloperskie, 111 na pisanie dokumentacji, 19 godzin na Review kodu, 19 na spotkania dotyczące omówienia dalszych prac projektowych oraz przy pomocy innym członkom zespołu oraz 49 godzin poświęciłem nad stworzeniem widoków na figmie. Prace nad częścią deweloperską 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 Sidebaru
- Zmiana kropki na przyciemnienie tła na Sidebar
- Poprawa strony do logowania i rejestracji
- Usunięcie username z account Redux
- Dodanie zamknięcia Sidebaru 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

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, 65

Biblioteka Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. 41, 43, 45, 47, 50, 51, 57

CI/CD Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. 64, 67

CSS Kaskadowe arkusze stylów (Cascading Style Sheets) — język opisu prezentacji dokumentów (np. HTML). Definiuje wygląd interfejsu: układ, kolory, typografię, odstępy, animacje i zachowania responsywne, oddzielając warstwę treści od warstwy prezentacji.. 50, 51

Design Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). 64

Disciplined Agile Delivery - Lean Life Cycle Disciplined Agile Delivery w wariantie Lean Life Cycle to sposób prowadzenia projektu, który łączy elastyczność Agile z przewidywalnością Waterfalla, ale bez stałych sprintów — praca toczy się w ciągłym przepływie. Na starcie zakłada mocniejszą

fazę przygotowawczą: doprecyzowanie zakresu, szkic architektury, identyfikację ryzyk i kryteria jakości. W realizacji następuje ciągłe doprecyzowywanie wymagań i backlogu, oparte na regularnym feedbacku udziałowców. Całość opiera się na praktykach Lean oraz lekkim governance: code review i regularnych przeglądach postępów. . 10

Droniarz Potoczne określenie osoby, która jest jednocześnie pilotem oraz operatorem drona. Zwykle entuzjasta dronów.. 8

Droniarz foto/video Pilot wykorzystujący drony fotograficzne/filmowe do rejestracji materiałów wizualnych (zdjęcia, wideo), zwykle z naciskiem na stabilizację i jakość obrazu.. 14

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. 38

Framework Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. 2, 48

Frontend Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. 2, 65

Hook (React) Prosta funkcja w React, która „dodaje” możliwości do elementu interfejsu — np. pozwala mu coś zapamiętać (stan) albo zrobić coś po zmianie/załadowaniu. Wszystkie hooki zaczynają się od `use...` (np. `useState`, `useEffect`).. 43, 47

Infinity 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. 58

JWT Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. 45, 64

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*)..
72

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. . 18

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..
43

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). 42

React Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. 43

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.. 43, 44

Responsywność Określenie związane z projektowaniem responsywnym (Responsive Web Design, RWD), czyli dostosowywaniem interfejsu do różnych rozmiarów i parametrów ekranów. Obejmuje m.in. elastyczne siatki, grafiki i

Media queries, tak aby układ i czytelność były zachowane na telefonach, tabletach i desktopach.. 51

Review kodu Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. 64

Sidebar Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. 35, 64–66

Spot Spotkanie zespołu projektowego, zazwyczaj krótkie i regularne, służące omówieniu postępów prac, problemów oraz planów na najbliższy okres. 52, 55, 57–59

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).. 43

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.. 43

UI Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. 43

Spis tabel

2.1	Zespół projektowy	7
2.2	Promotor	8
2.3	Droniarze	8
	Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)	15
	Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage	15
	Tabela 3.3: Usługa zewnętrzna: Mailtrap	16
	Tabela 3.4: Usługa zewnętrzna: LocationIQ	16
	Tabela 3.5: Usługa zewnętrzna: Google Maps (Maps URLs)	16
	Tabela 3.6: Usługa zewnętrzna: OpenFreeMap	16
	Tabela 3.7: Usługa zewnętrzna: Open-Meteo	16
	Tabela 3.8: Usługa zewnętrzna: Tenor GIF API	17
	Tabela 3.9: Usługa zewnętrzna: Where the ISS at?	17
4.1	Profil użytkownika	20
4.2	Lista dodanych spotów	21
4.3	Dodanie spota	22
4.4	Lista zdjęć	23
4.5	Lista filmów	23
4.6	Lista znajomych	24
4.7	Lista obserwujących	24
4.8	Lista obserwowanych	25
4.9	Lista polubionych/odwiedzonych/planowanych spotów	25
4.10	Lista komentarzy	26
4.11	Ustawienia profilu	27

4.12 Resetowanie hasła	28
4.13 Dodawanie do znajomych	29
4.14 Logowanie i rejestracja	30
4.15 Strona główna — podstawowe filtry	31
4.16 Strona główna — zaawansowane filtry	32
4.17 Ustawienia motywu (ręczna zmiana)	33
4.18 Zapamiętanie preferencji motywu	34
4.19 Szybki przełącznik motywów w interfejsie	35

Bibliografia

- [1] *Disciplined Agile Delivery*. PMI. 1 sty. 2025. URL: <https://www.pmi.org/disciplined-agile/process/introduction-to-dad/why> (term. wiz. 30.10.2025).
- [2] *Disciplined Agile Delivery — Lean Life Cycle*. PMI. 1 sty. 2025. URL: <https://www.pmi.org/disciplined-agile/lifecycle/lean-lifecycle> (term. wiz. 30.10.2025).
- [3] *About billing for GitHub Actions*. GitHub Docs. 1 sty. 2024. URL: <https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions> (term. wiz. 02.11.2025).
- [4] *Scalability and performance targets for Blob storage*. Microsoft Learn. 1 sty. 2024. URL: <https://learn.microsoft.com/azure/storage/blobs/scalability-targets> (term. wiz. 02.11.2025).
- [5] *What are the limitations in Mailtrap?* Mailtrap Docs. 1 sty. 2024. URL: <https://help.mailtrap.io/article/111-what-are-the-limitations-in-mailtrap/> (term. wiz. 02.11.2025).
- [6] *LocationIQ Pricing*. LocationIQ. 1 sty. 2024. URL: <https://locationiq.com/pricing> (term. wiz. 02.11.2025).
- [7] *Google Maps (Maps URLs)*. Google Maps. 1 sty. 2024. URL: <https://developers.google.com/maps/documentation/urls/get-started?hl=pl> (term. wiz. 02.11.2025).
- [8] *OpenFreeMap Documentation*. OpenFreeMap. 1 sty. 2024. URL: <https://openfreemap.org/docs> (term. wiz. 02.11.2025).
- [9] *OpenFreeMap Quick Start*. OpenFreeMap. 1 sty. 2024. URL: <https://openfreemap.org/docs/quick-start> (term. wiz. 02.11.2025).
- [10] *Open-Meteo API Usage & Pricing*. Open-Meteo. 1 sty. 2024. URL: <https://open-meteo.com/en/docs/usage-and-pricing> (term. wiz. 02.11.2025).
- [11] *Tenor API — Documentation*. Tenor. 1 sty. 2024. URL: <https://tenor.com/gifapi/documentation> (term. wiz. 02.11.2025).

- [12] *Where the ISS at? API*. wheretheiss.at. 1 sty. 2024. URL: <https://wheretheiss.at/> (term. wiz. 02. 11. 2025).
- [13] *React useState*. 1 sty. 2025. URL: <https://react.dev/reference/react/useState> (term. wiz. 03. 11. 2025).
- [14] *Redux*. 1 sty. 2025. URL: <https://redux.js.org/> (term. wiz. 03. 11. 2025).
- [15] *Axios*. 1 sty. 2025. URL: <https://axios-http.com/> (term. wiz. 03. 11. 2025).
- [16] *Tanstack Query*. 1 sty. 2025. URL: <https://tanstack.com/query/latest> (term. wiz. 03. 11. 2025).
- [17] *Tailwind*. 1 sty. 2025. URL: <https://tailwindcss.com/> (term. wiz. 03. 11. 2025).
- [18] *Motion*. 1 sty. 2025. URL: <https://motion.dev/> (term. wiz. 03. 11. 2025).

Załączniki

Płyta CD z następującą zawartością:

- *pliki projektowe* – pliki składające się na całość projektu
 - repozytorium kodu źródłowego wraz z instrukcją zbudowania i uruchomienia projektu
 - źródło pracy inżynierskiej.
- *Langmesser Adam_Redosz Mateusz_Oziemczuk Stanisław_Badek Kacper_praca pisemna* – katalog zawierający plik PDF z pracą inżynierską.