POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki
Filia w Gdańsku

Langmesser Adam
Nr albumu s27119
Nazwa specjalizacji: Aplikacje Internetowe
Redosz Mateusz
Nr albumu s27094
Nazwa specjalizacji: Aplikacje Internetowe
Oziemczuk Stanisław
Nr albumu s26982
Nazwa specjalizacji: Aplikacje Internetowe
Badek Kacper
Nr albumu s29168
Nazwa specjalizacji: Aplikacje Internetowe

Aplikacja webowa: spoty-na-drony.pl

Rodzaj pracy
inżynierska
Imię i nazwisko promotora
mgr Adam Urbanowicz

Gdańsk, miesiąc, 2100 obrony

Streszczenie: Celem niniejszej pracy było stworzenie w pełni funkcjonalnej i działającej aplikacji internetowej pozwalającej na szybkie wyszukiwanie spotów w okolicy oraz dzielenie się zdjęciami, filmami oraz doświdczeniem 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 styli 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:	**Recenzent:**
30 listopada 2025	dr Elżbieta Puźniakowska-Gałuch

Spis treści

Rozdział 1

Wstęp

1.1 O projekcie

1.2 Cel i zakres prac

1.3 Geneza pomysłu

Rozdział 2

Opis problemu

2.1 Rich picture

2.2 Udziałowcy

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

Tabela 2.1: Karta udziałowca: Zespół projektowy

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

Tabela 2.2: Karta udziałowca: Promotor

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

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

Tabela 2.3: Karta udziałowca: Droniarze

2.3 Istniejące rozwiązania

2.4 Wizja rozwiązania

2.5 Aspekty społeczne i biznesowe

2.5.1 Aspekty społeczne

2.5.2 Aspekty biznesowe

Rozdział 3

Planowanie

3.1 Metodologia pracy

3.1.1 Przegląd rozważanych podejść

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

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

3.1.2 Odrzucone podejścia

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

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

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

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

Podjęto decyzję o zastosowaniu **Disciplined Agile Delivery** [1] w wariancie **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**.

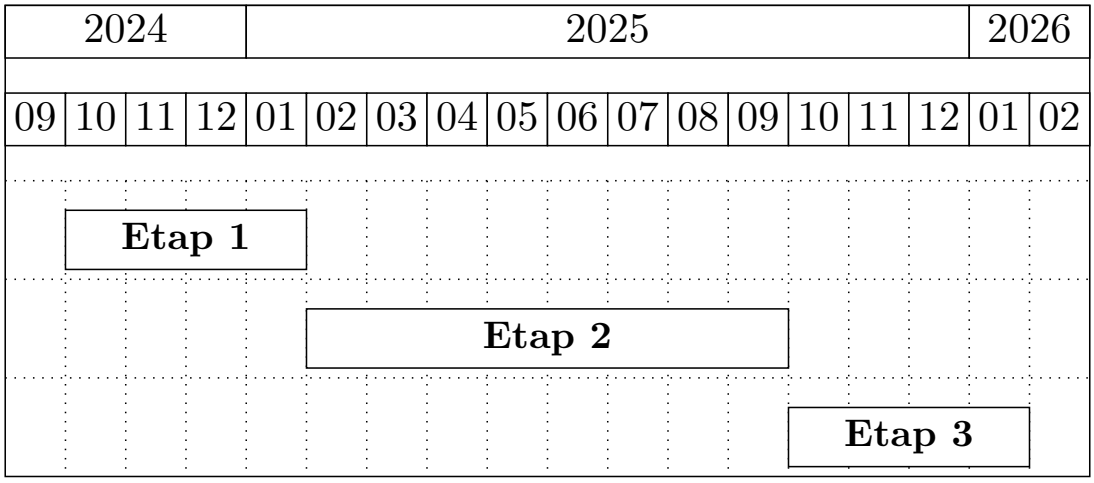
3.1.5 Podział ról w zespole

- **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.



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

  - Wstępne opracowanie wymagań.
  - Rozpoczęcie prac deweloperskich.

- **Etap 2 (luty 2025 – wrzesień 2025)**
  - Prace deweloperskie nad aplikacją.
  - Ciągła weryfikacja oraz korekcja wymagań postawionych systemowi.
  - Prace nad dokumentacją.

- **Etap 3 (październik 2025 – styczeń 2026)**
  - Finalizacja prac deweloperskich.
  - Implementacja testów automatycznych głównych funkcjonalności.
  - Ukończenie dokumentacji.

3.3 Technologie i narzędzia

Do realizacji projektu wykorzystano wiele technologii oraz narzędzi informatycznych. Przy wyborze technologii kierowaliśmy się ich popularnością, dostępnością dokumentacji oraz artykułów, a także łatwością użycia. Narzędzia zostały dopasowane do wybranych technologii i specyfikacji zadań. Poniżej przedstawiono opis wybranych opcji.

3.3.1 Technologie

3.3.2 Narzędzia

Do niektórych płatnych narzędzi mieliśmy bezpłatny dostęp za pośrednictwem uczelni, w innych mogliśmy założyć konta edukacyjne, które oferowały dostęp do wszystkich funkcji narzędzia. Gdy żadna z wymienionych opcji nie była udostępniona, wybieraliśmy rozwiązania darmowe.

- **IntelliJ IDEA Ultimate**

Jest to IDE od firmy JetBrains. Dzięki licznie dostępnym pluginom oferuje obsługę wielu języków programowania oraz innych składni. Pozwala również na integrację z repozytorium. Używaliśmy go do programowania zarówno frontendu, jak i backendu oraz tworzenia dokumentacji w LaTeX.

- **Docker Desktop**

  To narzędzie do zarządzania obrazami, kontenerami oraz wolumenami Docker. Zawiera w sobie również silnik tej technologii. Wykorzystywaliśmy je do lokalnego uruchamiana bazy danych oraz serwisu do cachowania.

- **One Drive**

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

- **Azure Blob Storage**

  To rozwiązanie chmurowe Microsoft, służące do bezpiecznego przechowywania dużej ilości danych nieustrukturyzowanych, takich jak pliki multimedialne, dokumenty czy kopie zapasowe. Dane są dostępne poprzez interfejs REST API usługi Azure Storage. Wykorzystaliśmy go do przechowywania zdjęć profilowych użytkownika oraz multimedii (zdjęcia i filmy) ze spotów i forum.

- **Jira**

  To narzędzie firmy Atlassian do zarządzania pracami nad projektem w metodykach zwinnych. Do Backlogu wpisywaliśmy zadania, a na tablicy Kanbanowej rejestrowaliśmy ich statusy oraz poświęcony czas.

- **GitHub**

  Zdalne repozytorium służące do przechowywania i wersjonowania kodu aplikacji. Zamieściliśmy tam kod naszego projektu. Do każdego zadania tworzyliśmy osobną gałąź z właściwą nazwą, a po zakończeniu prac przeprowadzaliśmy review kodu. Następnie łączyliśmy ją do głównej gałęzi deweloperskiej.

- **GitHub Actions**

  To narzędzie do implementacji procesów CI/CD na platformie GitHub, które umożliwiają automatyczne testowanie lub wdrażanie kodu. Uruchamiają się w reakcji na różne operacje w repozytorium, na przykład przesłanie zmian na wybraną gałąź. Stosowaliśmy je do automatycznego testowania i budowania projektu po każdorazowym wprowadzeniu zmian.

- **GitHub Copilot**

  To narzędzie sztucznej inteligencji będące asystentem programisty. W projekcie analizuje plik oraz pliki powiązane. Wykorzystywaliśmy go podczas review codu. Copilot skanował wszystkie pliki i w komentarzach opisywał sugerowane zmiany lub potencjalne błędy.

- **Discord**

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

- **Messenger**

  Komunikator będący usługą Facebooka. Daje możliwość tworzenia czatów grupowych lub prywatnych, a także udostępniania plików. Używaliśmy go do ustalania spotkań na Discordzie oraz szybkiej komunikacji.

- **Postman**

  To narzędzie służące do testowania endpointów API. Pozwala grupować zapytania w kolekcje, wysyłać ich różne typy oraz analizować odpowiedzi z serwera. Wykorzystywaliśmy go do testowania stworzonych endpointów oraz debugowania.

- **Figma**

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

- **Visual Paradigm**

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

3.4 Zasoby i ograniczenia

3.4.1 Zasoby

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

3.4.2 Ograniczenia

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

3.4.3 Usługi zewnętrzne

Niniejszy rozdział zawiera spis zewnętrznych API oraz usług użytych w projekcie.

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

Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)

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

Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage

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

Tabela 3.3: Usługa zewnętrzna: Mailtrap

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ4

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

Tabela 3.4: Usługa zewnętrzna: LocationIQ

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ5
Nazwa:	Google Maps (Maps URLs) [9]
Opis:	Otwieranie nawigacji w aplikacji Map Google (deep link/URL).
Limit:	Brak limitu w ramach dokumentowanego sposobu użycia.

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

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

Tabela 3.6: Usługa zewnętrzna: OpenFreeMap

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ7

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

Tabela 3.7: Usługa zewnętrzna: Open-Meteo

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

Tabela 3.8: Usługa zewnętrzna: Tenor GIF API

KARTA USŁUGI ZEWNĘTRZNEJ	
Identyfikator:	UZ9
Nazwa:	Where the ISS at? [13]
Opis:	HTTP API z bieżącą pozycją satelity, używane pomocniczo.
Limit:	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.9: Usługa zewnętrzna: Where the ISS at?

3.5 Analiza ryzyka

Rozdział 4

Analiza wymagań

4.1 Przypadki użycia

4.1.1 Aktorzy

Niniejszy rozdział przedstawia aktorów wraz z opisami.

Użytkownik systemu - Reprezentuje każdą osobę korzystającą z aplikacji.

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

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

Użytkownik premium - Użytkownik z wykupioną subskrypcją: ma dostęp do funkcji premium np. oznaczenie stref 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.

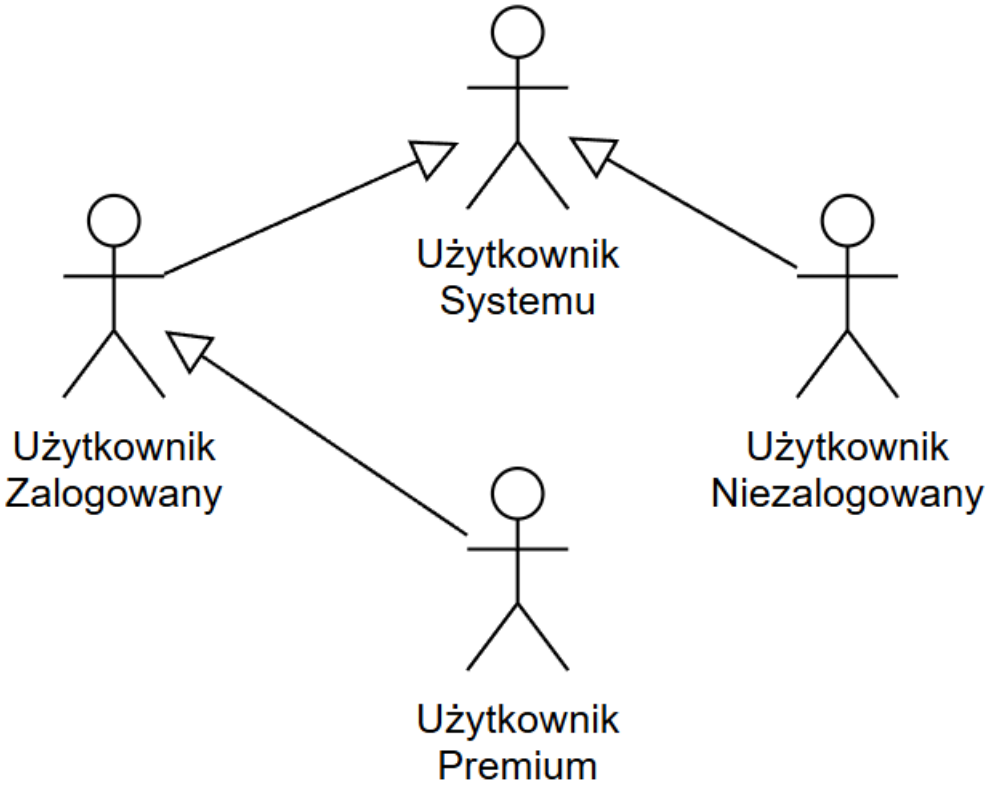
Usługa do przechowywania plików w chmurze - magazyn plików w chmurze służący do przechowywania załączników i multimediów użytkowników.

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 spota na podstawie jego współrzędnych geograficznych.

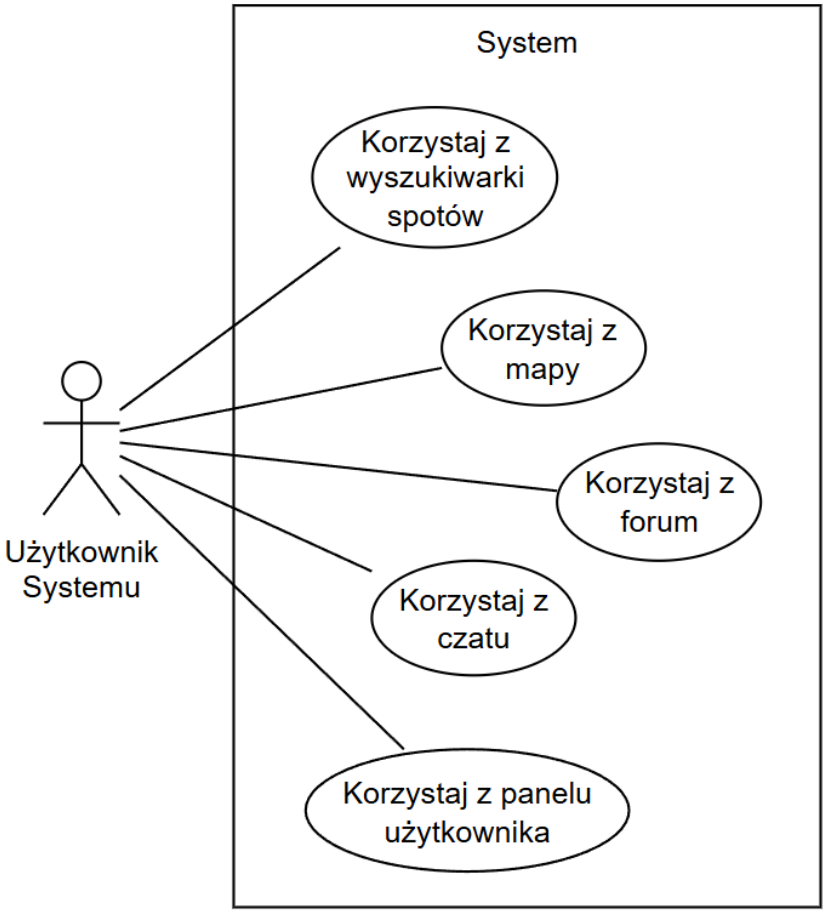


Rysunek 4.1: Diagram hierarchii użytkowników systemu

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

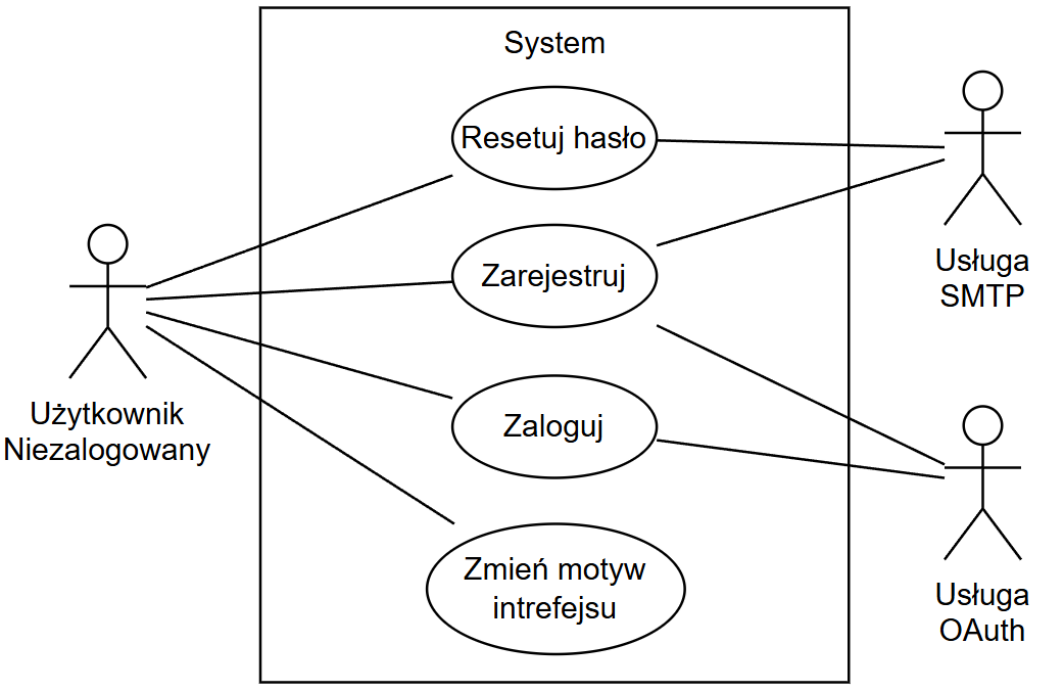
4.1.2 Diagramy przypadków użycia

Niniejszy rozdział przedstawia diagramy przypadków użycia.

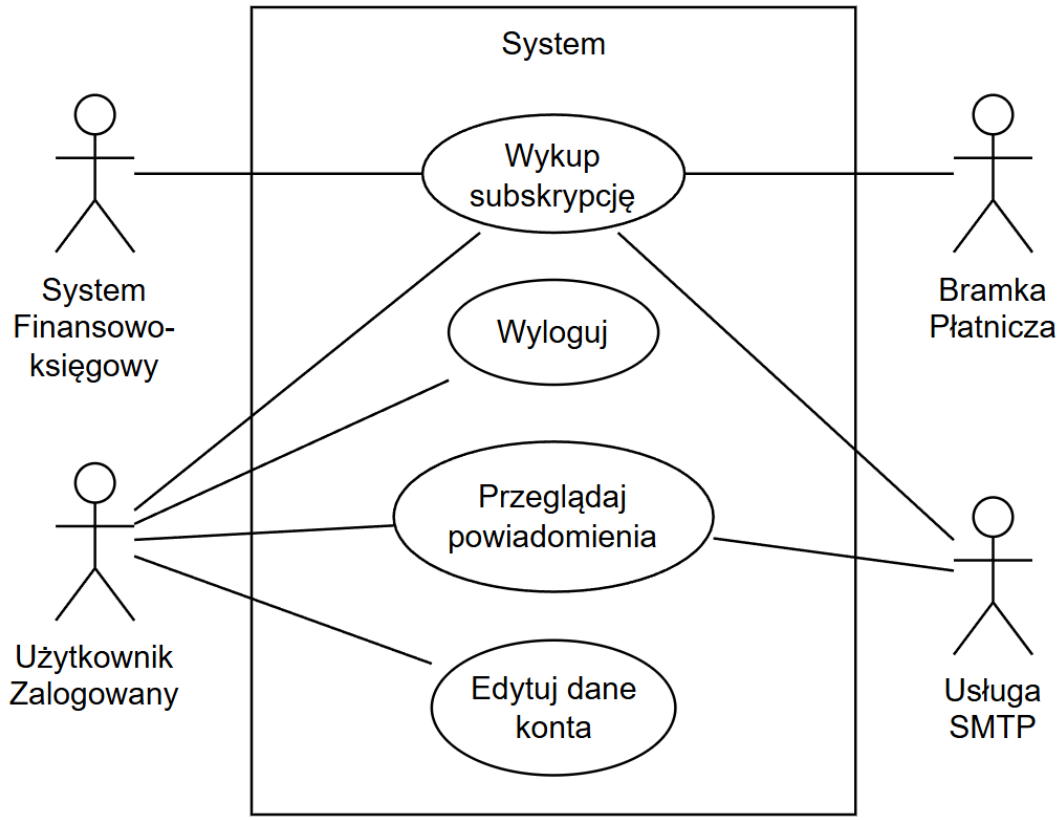


Rysunek 4.2: Wysokopoziomowy diagram przypadków użycia

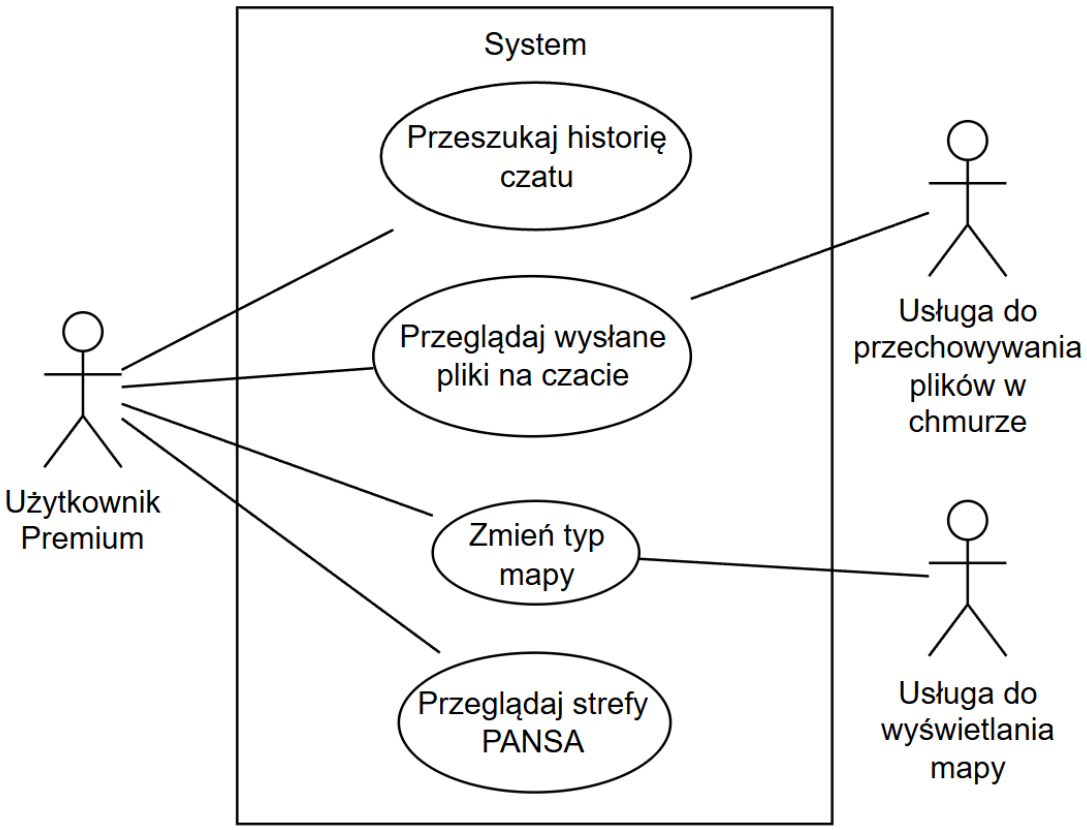
Diagram przedstawia podstawowe interakcje użytkownika z systemem. Na jego podstawie zespół projektowy podzielił architekturę aplikacji na 5 modułów: wyszukiwarkę spotów, mapę spotów, forum, czat oraz profil użytkownika. Pozostałe diagramy są bardziej szczegółowe.



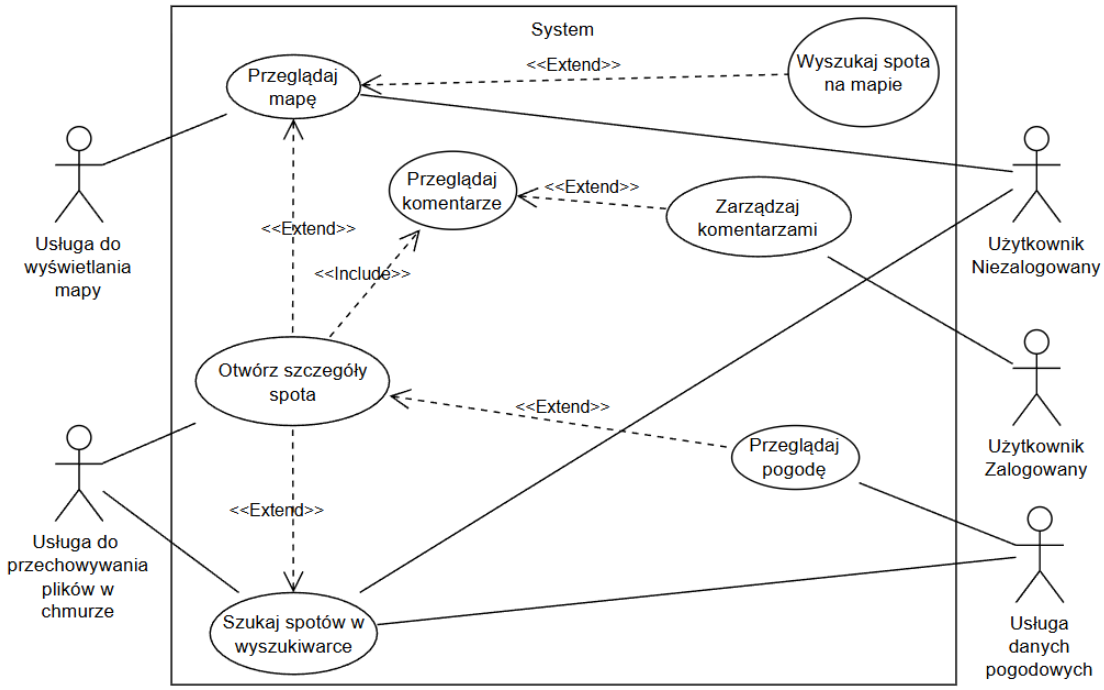
Rysunek 4.3: Diagram przypadków użycia dla użytkownika niezalogowanego



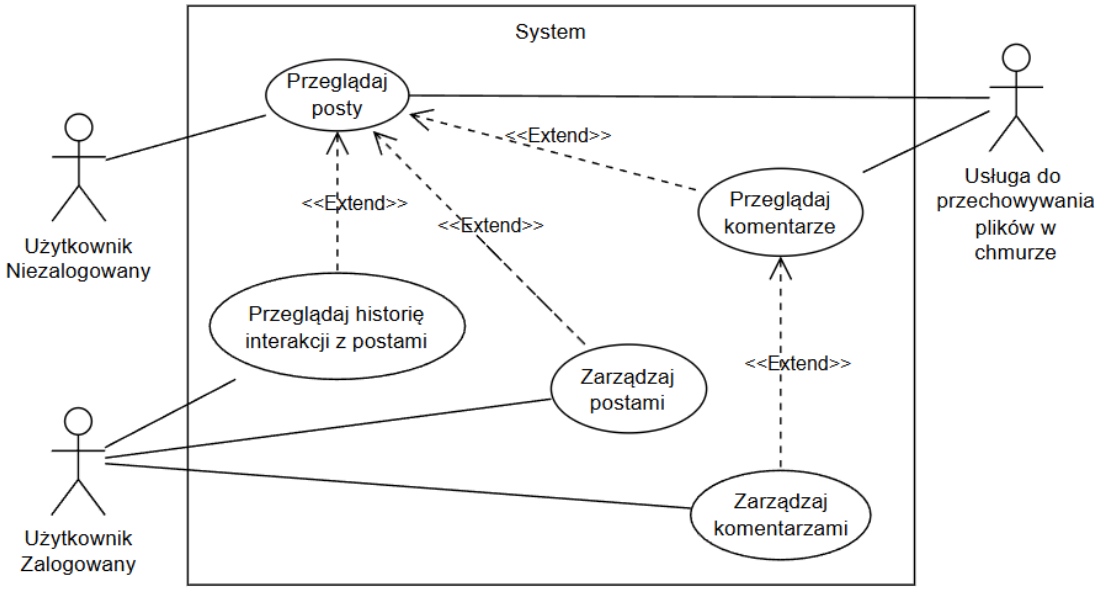
Rysunek 4.4: Diagram przypadków użycia dla użytkownika zalogowanego



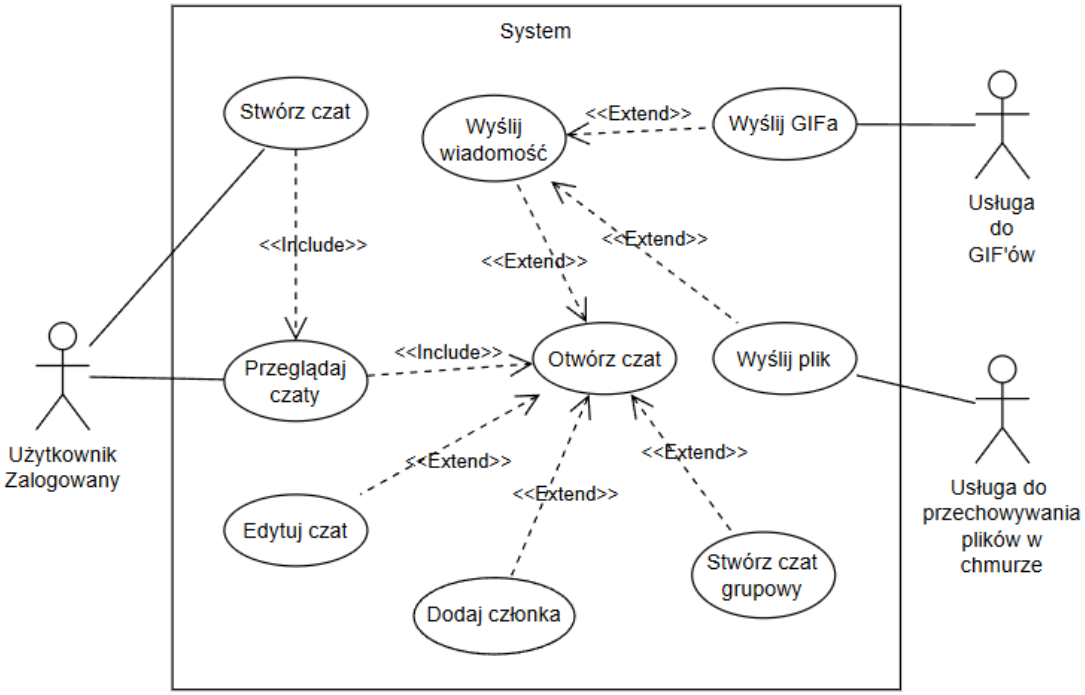
Rysunek 4.5: Diagram przypadków użycia dla użytkownika premium



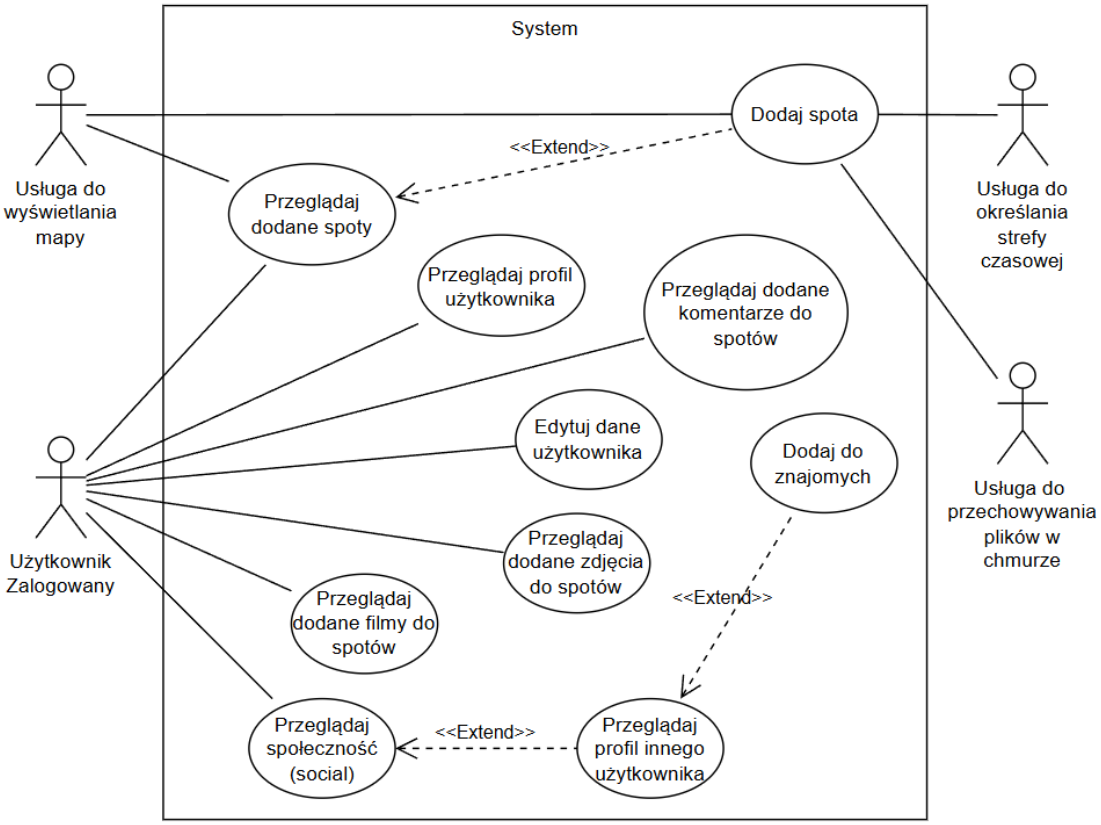
Rysunek 4.6: Diagram przypadków użycia wyszukiwarki spotów oraz mapy



Rysunek 4.7: Diagram przypadków użycia forum



Rysunek 4.8: Diagram przypadków użycia czatu



Rysunek 4.9: Diagram przypadków użycia profilu użytkownika

4.1.3 Scenariusz przypadków użycia

4.2 Wymagania ogólne i dziedzinowe

4.3 Wymagania funkcjonalne

4.3.1 Funkcjonalności dla mapy

4.3.2 Funkcjonalności dla chatu

4.3.3 Funkcjonalności dla forum

4.3.4 Funkcjonalności dla konta użytkownika

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

Tabela 4.1: Profil użytkownika

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

Tabela 4.2: Lista dodanych spotów

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

Tabela 4.3: Dodanie spota

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

Tabela 4.4: Lista zdjęć

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

Tabela 4.8: Lista obserwowanych

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

Tabela 4.9: Lista polubionych/odwiedzonych/planowanych spotów

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

Tabela 4.10: Lista komentarzy

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

Tabela 4.11: Ustawienia profilu

KARTA WYMAGANIA			
Identyfikator:	jednoznaczny symbol np. FO1, FO2 ..	Priorytet:	M
Nazwa:	Resetowanie hasła		
Opis:	Jako użytkownik chcę mieć możliwość zresetowania hasła do swojego konta.		
Kryteria akceptacji:	Po kliknięciu w odpowiedni link użytkownik może zresetować hasło do konta.		
Dane wejściowe:	Adres e-mail użytkownika do wysłania linku resetującego.		
Warunki początkowe:	Użytkownik podał poprawny adres e-mail użyty przy rejestracji.		
Warunki końcowe:	Hasło zresetowane po przejściu całej procedury.		
Sytuacje wyjątkowe:	Niepoprawny adres e-mail; wygasły lub nieprawidłowy token resetu; błąd połączenia z API.		
Szczegóły implementacji:	Frontend: formularz „zapomniałem hasła” (POST do endpointu wysyłającego link resetu) oraz formularz ustawienia nowego hasła (POST/PATCH z tokenem). Wysyłka przez `axios`; obsługa komunikatów o powodzeniu/błędach.		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; dronierze 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 `axios`; po akceptacji — aktualizacja listy (odświeżenie `react-query`).		
Udziałowiec:	Zespół projektowy 2.1; promotor 2.2; dronia-rze 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 `axios` z `withCredentials`. SSO: integracja z Google i GitHub (OAuth 2.0) z przekierowaniem i ustawieniem sesji po stronie backendu (httpOnly cookie). Obsługa statusu `401` zgodnie z mechanizmem wylogowania.		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, dronia-rze 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 `@tanstack/react-query` i `axios` (`GET` do backendu z parametrami lokalizacji). Filtry lokacji mapowane na parametry zapytania.		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.15: Strona główna — podstawowe filtry

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Strona główna z zaawansowanymi filtrami		
Opis:	Jako użytkownik chcę mieć dostęp do strony głównej, która wyświetla listę spotów, które można filtrować i sortować.		
Kryteria akceptacji:	Użytkownik widzi listę, którą może filtrować według miasta, tagów i oceny 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 `@tanstack/react-query` i `axios` z parametrami: lokalizacja, tagi, minimalna ocena oraz kryterium sortowania.		
Udziałowiec:	Zespół projektowy 2.1, promotor 2.2, droniarze 2.3.		
Wymagania powiązane:	SPXX		

Tabela 4.16: Strona główna — zaawansowane filtry

4.3.7 Funkcjonalności dla motywu

KARTA WYMAGANIA			
Identyfikator:	FOXX	Priorytet:	M
Nazwa:	Ustawienia motywu		
Opis:	Jako użytkownik chcę móc zmienić motyw aplikacji.		
Kryteria akceptacji:	Dostępna jest opcja przełączenia motywu na *jasny* lub *ciemny*; 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, droniarze 2.3.		
Wymagania powiązane:			

Tabela 4.17: Ustawienia motywu (ręczna zmiana)

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

Tabela 4.18: Zapamiętanie preferencji motywu

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

Tabela 4.19: Szybki przełącznik motywu w interfejsie

4.4 Wymagania pozafunkcjonalne

4.5 Wymagania interfejs z otoczeniem

4.6 Wymagania na środowisko docelowe

Rozdział 5

Projekt

5.1 Wzorce projektowe

5.2 Architektura systemu

5.2.1 Diagram architektury

5.2.2 Komponenty systemu

5.3 Projekt bazy danych

5.3.1 Model danych

5.3.2 Diagram ERD

5.4 Architektura interfejsu użytkownika

5.4.1 Projekt strony głównej

5.4.2 Projekt panelu logowania

5.4.3 Projekt mapy

5.4.4 Projekt chatu

5.4.5 Projekt forum

5.4.6 Projekt konta użytkownika

Rozdział 6

Przebieg realizacji projektu

W niniejszym rozdziale przedstawiono przebieg realizacji projektu w kolejnych Etapach zdefiniowanych w harmonogramie. Sposób realizacji projektu odzwierciedla sposób 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 (czerwiec–wrzesień 2024)

Faza przedprojektowa obejmowała okres od czerwca do września 2024 roku i poprzedzała formalne zatwierdzenie tematu pracy oraz opracowanie harmonogramu.

Zespół rozpoczął prace deweloperskie w czerwcu 2024 roku, co było możliwe dzięki temu, że niezależnie od ostatecznego tematu zakładano stworzenie aplikacji webowej(TODO) wymagającej kont użytkowników. Wynikało to bezpośrednio ze specyfiki specjalizacji „Aplikacje Internetowe”, na której studiują wszyscy członkowie zespołu. W efekcie część prac technicznych została wykonana jeszcze przed

formalnym wyborem tematu projektu.

Czerwiec 2024

Cały zespół

- Przygotowanie szkieletu projektu backendu w technologii Spring Boot – utworzono podstawowe pakiety, konfigurację aplikacji oraz minimalną strukturę modułów.

Adam Langmesser

- Utworzenie repozytorium w serwisie GitHub i skonfigurowanie podstawowych reguł pracy z repozytorium.
- Przygotowanie pierwszego potoku CI/CD dla backendu – automatyczne budowanie projektu i uruchamianie testów jednostkowych na serwerze ciągłej integracji.

Kacper Badek

- Konfiguracja narzędzia Jira: zapoznanie się z typowym podziałem na epiki, taski i podtaski, zdefiniowanie statusów przepływu pracy oraz utworzenie Tablica Kanban.

Lipiec 2024

Cały zespół

- Przygotowanie szkieletu aplikacji frontendowej (React + TypeScript) wraz z podstawową strukturą komponentów i konfiguracją narzędzi budujących.

Adam Langmesser

- Rozpoczęcie prac nad potokiem CI/CD dla frontendu, którego konfigurację finalnie ukończono we wrześniu 2024 roku.
- Implementacja podstawowej logiki logowania i rejestracji użytkownika zarówno po stronie backendu (endpointy REST API), jak i frontendu (formularze oraz obsługa żądań).

Sierpień 2024

W sierpniu 2024 roku zespół skupił się na zagadnieniach bezpieczeństwa oraz na dopracowaniu pierwszych ekranów aplikacji:

Adam Langmesser

- Dodanie Spring Security i implementacja logiki uwierzytelniania oraz autoryzacji użytkownika po stronie backendu.

Mateusz Redosz

- Konfiguracja biblioteki Tailwind CSS na froncie, umożliwiająca spójne i responsywne stylowanie komponentów.
- Konfiguracja narzędzia Prettier do automatycznego formatowania kodu frontendu.
- Stworzenie formularza rejestracji użytkownika wraz z podstawową walidacją po stronie frontendu.
- Dodanie i konfiguracja biblioteki TanStack Query do obsługi komunikacji z backendem i cachowania danych.

Stanisław Oziemczuk

- Implementacja logiki resetowania hasła (proces „zapomniałem hasła”) po stronie frontendu.

Kacper Badek

- Implementacja strony logowania użytkownika po stronie frontendu.

Wrzesień 2024

We wrześniu 2024 roku kontynuowano prace nad bezpieczeństwem i przygotowaniem środowiska uruchomieniowego:

Adam Langmesser

- Zastąpienie bazy danych działającej w pamięci (in-memory) instancją uruchamianą w kontenerze Docker, co urealniło środowisko deweloperskie i testowe.
- Dostosowanie potoku CI/CD backendu tak, aby uwzględniał uruchamianie bazy danych w kontenerze podczas wykonywania testów.
- Dokończenie konfiguracji potoku CI/CD dla frontendu.

Kacper Badek

- Implementacja logiki resetowania hasła po stronie backendu – generowanie tokenów, ich weryfikacja oraz integracja z istniejącym procesem resetowania na froncie.

Zespół frontendowy

- Implementacja przycisków na stronie logowania, umożliwiających logowanie i rejestrację z wykorzystaniem OAuth (GitHub i Google) po stronie frontendu. TODO: doprecyzować osobę odpowiedzialną za implementację.

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.

Mateusz Redosz

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

Stanisław Oziemczuk

- Implementacja logowania OAuth z wykorzystaniem Google i GitHub po stronie backendu oraz integracja z frontem.

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. Ze względu na ograniczone możliwości customizacji wyglądu biblioteki zdecydowano się później na zmianę dostawcy kafelków mapowych na usługę TODO: uzupełnić nazwę docelowej usługi mapowej.
- Poprawki konfiguracji narzędzia ESLint po stronie frontendu.

Mateusz Redosz

- Dalsze poprawki obsługi JWT na backendzie oraz logowania błędów związanych z procesem logowania i rejestracji.

Stanisław Oziemczuk

- Poprawki działania logowania użytkownika po stronie backendu i frontendu, obejmujące obsługę błędów oraz komunikaty dla użytkownika.

Kacper Badek

- Implementacja logiki cyklicznego usuwania przeterminowanych tokenów resetu hasła.

Cały zespół

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

TODO: odwołanie do rozdziału z harmonogramem projektu.
TODO: odwołanie do rozdziału z analizą wymagań.

Grudzień 2024

Adam Langmesser

- Dalsze poprawki konfiguracji Spring Security na backendzie, w tym doprecyzowanie ról i uprawnień.
- Rozszerzenie encji użytkownika o dane deweloperskie oraz przygotowanie inicjalnych danych w bazie (np. konta testowe).
- 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 asercji.

Mateusz Redosz

- Rozszerzenie stanu Redux o informację o tym, czy użytkownik jest aktualnie zalogowany.
- Implementacja testów Testy E2E dla procesów logowania i rejestracji użytkownika.
- Dodanie uruchamiania testów frontendu do potoku CI/CD.

- Poprawa sposobu wyświetlania szczegółów spota na froncie.

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.

Kacper Badek

- Dodanie danych deweloperskich dla mapy (przykładowe spoty, 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. TODO: odwołanie do rozdziału z analizą wymagań oraz do rozdziału opisującego projekt architektury.

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 stylistyki komponentów oraz zachowania w trybie ciemnym i jasnym.
- Poprawa logiki wylogowywania użytkownika na froncie (m.in. czyszczenie stanu Redux, przekierowania).
- Poprawa logiki dodawania komentarzy do spotów na mapie po stronie frontendu.

Stanisław Oziemczuk

- Implementacja logiki dodawania spota do ulubionych po stronie frontendu oraz integracja z backendem.
- Implementacja integracji z zewnętrznym API pogodowym służącym do pobierania podstawowych danych pogodowych dla danego spota.

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.

TODO: uzupełnić nazwę i dostawcę API pogodowego oraz odwołać się do kart usług zewnętrznych i bibliografii.

Marzec 2025

W marcu 2025 roku skoncentrowano się na dopracowaniu systemu powiadomień e-mail oraz rozbudowie komentarzy, a także na ułatwieniu startu środowiska deweloperskiego.

Kacper Badek

- Refaktoryzacja szablonów wiadomości e-mail związanych z rejestracją i resetowaniem hasła – zastąpienie układu opartego na *flexboxie* layoutem tabelarycznym z *inline-stylami*, ujednolicenie struktury HTML, metadanych oraz stylu logo, co poprawiło kompatybilność z różnymi klientami pocztowymi.
- Rozbudowa systemu komentarzy do spotów: dodanie możliwości edycji, usuwania i głosowania na komentarze, wydzielenie dedykowanego kontrolera po stronie backendu oraz lepsza integracja z widokiem szczegółów spota (paginacja, powiadomienia, prezentacja ocen).

Stanisław Oziemczuk

- Przygotowanie skryptu uruchamiającego wszystkie wymagane kontenery Docker (bazy danych i usługi pomocnicze) na potrzeby środowiska deweloperskiego, co uprościło proces uruchamiania projektu na nowych stanowiskach.

Mateusz Redosz

- Migracja wszystkich adresów obrazów (galerie spotów oraz logotypy w szablonach e-mail) z usługi Google Drive do CDN ucarecdn.com, co ujednoliciło sposób serwowania grafik i poprawiło niezawodność ich wczytywania.

Kwiecień 2025

W kwietniu 2025 roku uporządkowano projekt LaTeX pracy dyplomowej oraz znacząco wzmocniono integrację aplikacji z TypeScript i Tailwind CSS, jednocześnie przebudowując layout i wdrażając pierwszą pełną wersję modułu czatu.

Adam Langmesser

- Konfiguracja projektu dokumentacji w LaTeX: dodanie pliku `.gitignore` z typowymi plikami pomocniczymi LaTeXa, przygotowanie `Makefile` automatyzującego proces budowania, utworzenie pliku bibliografii oraz głównego dokumentu `engineer.tex` z przykładową strukturą rozdziałów i klasą `sprz.cls`.

- Implementacja pełnej funkcjonalności czatu w warstwie backendowej i frontendowej (podstawowe API, integracja z Redux, widoki ekranów czatu), a także dostosowanie layoutu i paska bocznego do nowej sekcji.

Mateusz Redosz

- Wzmocnienie integracji projektu z TypeScript: aktualizacja wersji kompilatora, dodanie pliku `tsconfig`, zdefiniowanie typowanego store Redux oraz pomocniczych hooków `useDispatchTyped` i `useSelectorTyped`, a także przygotowanie zestawu interfejsów modelu dla komentarzy, spotów i użytkowników (wraz z odpowiadającymi im typami wyliczeniowymi).

- Integracja Tailwind CSS z projektem frontendu, w tym konfiguracja pluginów i ustawień Prettiera, aktualizacja zależności, zastąpienie dotychczasowych klas i styli komponentów utilitami Tailwinda oraz usunięcie starej konfiguracji `PostCSS`.

- Przebudowa layoutu aplikacji: zastąpienie nagłówka (*Header*) nowym panelem bocznym (*Sidebar*) z linkami nawigacyjnymi, opcjami konta oraz przełącznikiem trybu ciemnego, a także aktualizacja testów integracyjnych i jednostkowych tak, aby obejmowały nowy układ i zachowanie sidebaru.

- Integracja API profilu użytkownika z nową stroną *Profile* (statystyki oraz najpopularniejsze zdjęcia) oraz przepisanie komponentów *Sidebar* i *Layout* na TypeScript z lepszą obsługą urządzeń mobilnych (m.in. komponent *MobileBar* i typowane selektory).

Maj 2025

W maju 2025 roku kluczowe prace dotyczyły migracji mapy do MapLibre, rozwoju forum oraz panelu konta użytkownika, a także uporządkowania warstwy API i formularzy.

Stanisław Oziemczuk

- Migracja mapy z biblioteki Leaflet na MapLibre, w tym dostosowanie komponentów frontendu, routingów oraz obsługi interakcji ze spotami.

- Zmiana wyglądu znacznika lokalizacji użytkownika na mapie, co poprawiło czytelność i spójność z nową paletą kolorów oraz warstwami mapy.
- 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 MapLibre (wygodniejsze otwieranie modali ze szczegółami, dopracowany wygląd markerów oraz layout sekcji ulubionych spotów).

Mateusz Redosz

- Reorganizacja panelu bocznego (*Sidebar*) do zestawu modularnych komponentów z dynamicznie generowanymi linkami, dodanie ogólnych hooków do zarządzania stanem (w tym trybem ciemnym) oraz wprowadzenie zależności do biblioteki `motion` na potrzeby animacji.
- Przeniesienie kluczowych komponentów formularzy (*FormContainer*, *FormInput*, *OauthButton*, *OauthForm*) z JavaScriptu na TypeScript, zastąpienie starego komponentu *Input* nowym *FormInput* z lepszą walidacją i stylami, poprawa zmiennych CSS oraz responsywności widoków formularzy.
- Dodanie sekcji *„Znajomi”* w panelu konta, obejmującej obsługę znajomych, obserwowanych i obserwujących, wprowadzenie nowych modeli i endpointów do zarządzania relacjami oraz dodanie testów jednostkowych dla komponentu *FriendCard*.
- Refaktoryzacja endpointów użytkownika tak, aby login był pobierany z kontekstu uwierzytelnienia zamiast z parametru ścieżki, dostosowanie do tego wywołań na froncie oraz przeniesienie slice'a konta na TypeScript z uproszczonym stanem i uporządkowanymi importami.

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, wraz z dostosowaniem layoutu, routingu i podstawowych styli.

Czerwiec 2025

W czerwcu 2025 roku rozwijano przede wszystkim panel konta użytkownika, w tym ulubione spoty, oraz refaktoryzowano sekcję komentarzy do spotów.

Mateusz Redosz

- Przeniesienie stanu sidebaru do osobnego slice'a w Redux, co poprawiło spójność zachowania panelu bocznego między różnymi widokami i na urządzeniach o różnej rozdzielczości.

- Przebudowa profilu użytkownika poprzez rozdzielenie widoków własnego profilu i profili innych użytkowników (nowe endpointy *public/private profile*, rozszerzony model danych), dodanie przycisków akcji (follow/unfollow, zaproszenie do znajomych), usprawnienie nawigacji z kart znajomych i sidebaru oraz aktualizacja routingu i testów do nowego podziału.

- 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.

Stanisław Oziemczuk

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

Lipiec 2025

W lipcu 2025 roku intensywnie rozwijano część społecznościową, panel konta oraz czat, a także usprawniano obsługę mapy i mediów w systemie.

Mateusz Redosz

- 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'a Redux do zarządzania aktywnym typem listy social.

- Dodanie nowej sekcji *„Zdjęcia"* w panelu użytkownika, z możliwością sortowania i filtrowania po dacie, wyodrębnienie wspólnego wrappera layoutu dla podstron konta, aktualizacja routingu i styli oraz dopisanie testów dla wyboru daty.

- Dodanie sekcji komentarzy w panelu konta (z grupowaniem po dacie i spocie oraz filtrowaniem/sortowaniem po dacie), co ułatwiło użytkownikom przeglądanie własnej aktywności.

- Dodanie strony ustawień konta umożliwiającej edycję nazwy użytkownika, adresu e-mail i hasła (z walidacją w oparciu o `zod` i `react-hook-form`), aktualizacja routingu i modeli (nowe enumy/interfejsy, wariant przycisku), usunięcie starych endpointów edycji danych z *AccountController* 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

- Rozbudowa funkcjonalności czatu – dodanie szczegółowego pobierania danych czatu i wysyłania wiadomości po WebSocket, odświeżenie interfejsu

(spinner, skeletony list, okno rozmowy z separatorami dat i pustym stanem) oraz wprowadzenie nowych zależności i zmian w stylach poprawiających UX.

- Uporządkowanie formatowania kodu z wykorzystaniem narzędzia Prettier, dodanie skryptu `format:check` oraz sprawdzania formatowania w potokach CI/CD.
- Przeprowadzenie dużej refaktoryzacji czatu: ujednolicenie modelu danych do jednego *ChatDto*, uproszczenie powiązanego API i struktury store'a Redux, dostosowanie komponentów czatu do nowego modelu oraz poprawa wyglądu (tło, paski przewijania), połączona z czyszczeniem i formatowaniem kodu.
- Poprawa danych deweloperskich w bazie (m.in. przykładowych spotów i użytkowników), tak aby lepiej odzwierciedlały realistyczne scenariusze działania systemu.
- Usprawnienie mechanizmu nieskończonego przewijania i nazewnictwa listy czatów oraz redukcja drobnego długu technicznego (lepsze wartości domyślne, uproszczone typy).
- Stworzenie ogólnej logiki i komponentów pomocniczych umożliwiających wszystkim członkom zespołu wygodne korzystanie z WebSocketów na froncie (uogólnione hooki i funkcje narzędziowe).
- Poprawa UX czatu poprzez dodanie grupowania wiadomości w czasie z podpowiedzią pełnej daty, wielowierszowego pola tekstowego z obsługą Enter/Shift+Enter, uproszczonych klas ikon oraz dodatkowych poprawek layoutu i efektów najechania.
- Wprowadzenie drobnych poprawek w potokach CI/CD (konfiguracja kroków, stabilność zadań).

Stanisław Oziemczuk

- Przebudowa wyszukiwania spotów na mapie – zastąpienie dotychczasowych filtrów nowym paskiem wyszukiwania po nazwie z bocznym panelem wyników i paginacją oraz dopracowanie interfejsu (gwiazdki ocen, warstwy sidebarów) i logiki cache'owania zapytań.

- Ujednolicenie obsługi mediów dla spotów i komentarzy (wspólny model dla zdjęć i wideo), aktualizacja powiązanych struktur danych i galerii w UI oraz dodanie zależności i styli potrzebnych do odtwarzania oraz wygodnego wyświetlania multimediów.

- Refaktoryzacja struktury plików na backendzie związanych z modułem mapy, co uprościło nawigację po kodzie i ułatwiło dalszy rozwój funkcjonalności.

Kacper Badek

- Ujednolicenie modelu tagów dla spotów i forum, rozbudowa funkcjonalności postów na forum, aktualizacja API kategorii i tagów oraz dodanie edytora typu *rich text* i lepiej wystylizowanych komponentów wyboru, a także usprawnienie hooka do wykrywania kliknięć poza aktywnymi elementami.

Sierpień 2025

W sierpniu 2025 roku kontynuowano rozwój czatu, wyszukiwania oraz panelu użytkownika, a także rozbudowano stronę główną i usprawniono forum.

Adam Langmesser

- Dodanie do czatu możliwości wyszukiwania i wysyłania animowanych obrazów (GIF) oraz wstawiania emoji z dedykowanego okna przy polu wpisywania wiadomości, co znacząco poprawiło komfort rozmowy.

- Usprawnienie czatu poprzez wprowadzenie optymistycznego wysyłania wiadomości z natychmiastową aktualizacją okna rozmowy i listy czatów, paginację oraz wygodniejsze wybieranie konwersacji (z obsługą nieprzeczytanych), dodanie nowych endpointów do stronicowanego pobierania wiadomości, wzmocnienie typowania i refaktoryzacja komponentów oraz przygotowanie zależności pod wirtualizowane listy i Infinite scroll.

Stanisław Oziemczuk

- Uporządkowanie tematu współrzędnych spotów: dodanie jednoznacznego „punktu środka" spota i konsekwentne wykorzystywanie go w backendzie i frontendzie, co poprawiło dokładność pozycjonowania markerów na mapie.

Mateusz Redosz

- Rozbudowa strony głównej: dodanie karuzeli z najpopularniejszymi spotami, wyszukiwarki po lokalizacji (kraj/region/miasto) z listą wyników i dystansem od użytkownika, wprowadzenie odpowiednich endpointów i modeli danych po stronie API oraz poprawa zachowania i wyglądu sidebaru, co ułatwiło odkrywanie nowych miejsc do latania dronem.

- Dodanie zaawansowanego wyszukiwania spotów na stronie głównej (osobny widok z filtrami po mieście i tagach oraz przełącznik między prostym a zaawansowanym trybem), dostosowanie modeli i endpointów API (m.in. obsługa sugestii tagów i pól opcjonalnych) oraz dopracowanie interfejsu listy wyników, w tym komunikatu o braku dopasowanych spotów.

- Wprowadzenie paginacji do endpointów panelu użytkownika i przebudowa powiązanych komponentów frontendu na nieskończone przewijanie oparte na React Query i Intersection Observer, dzięki czemu listy (znajomi, obserwujący, ulubione spoty, media, komentarze) ładują się porcjami z poprawioną obsługą stanów ładowania i pustych wyników.

- Dodanie sekcji *„Zdjęcia"* w panelu społecznościowym użytkownika (ze zdjęciami pogrupowanymi po dacie i stronicowaniem), uogólnienie i uproszczenie logiki nieskończonego przewijania dla różnych zakładek social (friends/followers/followed/photos) oraz poprawa drobnych błędów w backendzie i kontraktach komponentów.

- Dodanie do panelu użytkownika funkcji dodawania własnych spotów (z adresem, multimediami i wielokątem na mapie) wraz z nowymi endpointami i modelami danych oraz widokiem listy dodanych spotów z formularzem w modalu i nieskończonym przewijaniem.

Kacper Badek

- Ujednolicenie modelu tagów dla spotów i forum oraz dalsza rozbudowa funkcjonalności postów forum, aktualizacja API kategorii i tagów, dodanie edytora typu *rich text* oraz lepiej wystylizowanych pól wyboru, a także usprawnienie hooka do wykrywania kliknięć poza aktywnymi elementami.

- Usprawnienie forum poprzez wprowadzenie czytelniejszych adresów URL postów opartych na slugach tytułów, lepszą walidację tytułu oraz wygodniejszą nawigację (m.in. łatwe przejście z posta do profilu autora).

Wrzesień 2025

Adam Langmesser

- Wprowadzenie możliwości rozpoczynania lub kontynuowania prywatnych rozmów czatowych bezpośrednio z list znajomych i obserwujących poprzez dodanie przycisku *„Wiadomość”* na kartach społecznościowych oraz integrację z logiką wyszukiwania i tworzenia czatów prywatnych.

- Dodanie nowoczesnej obsługi emoji w komunikatorze – integracja komponentu wyboru emoji z polem wpisywania wiadomości oraz uporządkowanie wyglądu okna wyboru animowanych obrazów (GIF), co poprawiło ergonomię korzystania z czatu.

Mateusz Redosz

- Refaktoryzacja systemu powiadomień na froncie, umożliwiająca jednoczesne wyświetlanie wielu komunikatów oraz zwiększająca modularność i możliwość ponownego wykorzystania komponentów powiadomień.

- Wprowadzenie stronicowania (paginacji) dla endpointów wyszukiwania spotów na backendzie oraz dostosowanie komponentów stron głównych i list wyników wyszukiwania do pracy w trybie nieskończonego przewijania (Infinite scroll) z mechanizmem *„load more”*, z wykorzystaniem tzw. obserwatora przecięcia (Intersection Observer) oraz przekazywaniem referencji i stanów ładowania.

- Dodanie walidacji formularza dodawania spota w oparciu o schemat walidacyjny, w tym sprawdzanie kompletności danych, wymaganego zestawu multimediów oraz punktów poligonu, a także wyświetlanie komunikatów błędów bezpośrednio pod sekcjami mediów i obszaru na mapie.

- Rozszerzenie komponentu przesyłania multimediów o podgląd wybranych obrazów i materiałów wideo oraz poprawę zarządzania cyklem życia adresów URL podglądu; jednocześnie podniesiono priorytet wyświetlania listy powiadomień (warstwa z-index), aby były lepiej widoczne.

- Wprowadzenie pełnej obsługi zmiany zdjęcia profilowego użytkownika, obejmującej wysyłanie plików do backendu, usuwanie poprzednich zdjęć z magazynu obiektowego oraz aktualizację adresu zdjęcia w profilu użytkownika.

- Refaktoryzacja struktury projektu LaTeX (LaTeX): zastąpienie dotychczasowej treści demonstracyjnej rzeczywistymi rozdziałami opisującymi projekt *„spoty-na-drony.pl”*, aktualizacja metadanych (tytuł, autorzy, promotor, cele projektu) oraz dodanie osobnych plików rozdziałów dla głównych części pracy (wstęp, opis problemu, kontekst projektu, analiza wymagań, decyzje projektowe, projekt, planowanie, implementacja, testowanie, prezentacja systemu, podsumowanie) i włączenie ich do pliku głównego.

- Rozszerzenie zaawansowanych możliwości wyszukiwania spotów o sortowanie wyników według popularności lub oceny oraz filtrowanie po minimalnej ocenie, wraz z odpowiednimi listami rozwijanymi w interfejsie użytkownika i obsługą nowych parametrów po stronie backendu.

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, z wprowadzeniem dedykowanych struktur DTO oraz dostosowaniem komponentów prezentujących dane pogodowe do nowego modelu danych i wymagań responsywności.

6.4 Etap 3 (październik 2025 – styczeń 2026)

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

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

Październik 2025

Adam Langmesser

- Wprowadzenie możliwości tworzenia czatów grupowych, w tym obsługi wyboru uczestników, komunikacji z backendem oraz integracji z istniejącą listą czatów.

- Dodanie funkcjonalności wysyłania i wyświetlania załączników w wiadomościach czatu (pliki oraz obrazy), wraz z logiką wyboru, podglądu i wysyłania samych plików bez treści tekstowej.

- Rozszerzenie istniejącej funkcjonalności czatów grupowych o możliwość edycji ich parametrów (zmiana nazwy oraz obrazu czatu) po stronie backendu i frontendu.

- Dodanie możliwości dołączania nowych użytkowników do istniejących czatów grupowych, wraz z odpowiednimi endpointami HTTP i modyfikacją interfejsu użytkownika.

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.

- Przebudowa struktury rozdziału *Implementacja*: wydzielenie osobnych sekcji dla backendu, frontendu i CI/CD, uzupełnienie dokumentacji backendu o listę endpointów z przykładami odpowiedzi oraz przygotowanie miejsc na opis implementacji pozostałych części.

- Przygotowanie struktury rozdziału *Analiza wymagań*, w tym włączenie plików dotyczących przypadków użycia, wymagań funkcjonalnych i pozafunkcjonalnych oraz wymagań dotyczących środowiska docelowego.

- Przygotowanie struktury rozdziału *Nakład pracy* wraz z podrozdziałami opisującymi indywidualny wkład każdego członka zespołu oraz włączenie go do głównej struktury dokumentu.

- Przebudowa i uporządkowanie struktury dokumentacji (m.in. „Ogólny nakład pracy", „Indywidualne nakłady pracy" oraz „Aspekty społeczne i biznesowe") poprzez wydzielenie ich do osobnych plików, usunięcie przestarzałych fragmentów oraz ujednolicenie poziomów nagłówków i oznaczeń sekcji.

Stanisław Oziemczuk

- Wprowadzenie rozszerzonej galerii multimediów 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, w tym korekta wysokości komponentów, układu galerii multimediów oraz przewijania list komentarzy.

Kacper Badek

- Przeprowadzenie dużej refaktoryzacji funkcjonalności forum, obejmującej obsługę komentarzy do postów (dodawanie, edycję, usuwanie oraz głosowanie) po stronie backendu i frontendu.
- Przebudowa stron forum w kierunku architektury z nieskończonym przewijaniem (Infinite scroll), z wykorzystaniem mechanizmów stronicowania i sortowania postów po stronie klienta i serwera.
- Zastąpienie klasycznych wskaźników ładowania (spinnerów) loaderami typu Skeleton loader dla listy postów oraz paneli kategorii i tagów, co poprawiło odbiór interfejsu w trakcie ładowania danych.

Listopad 2025

Adam Langmesser

- Opracowanie rozdziału *Analiza wymagań* – przygotowanie listy aktorów systemu, diagramu przypadków użycia oraz scenariuszy przypadków użycia.

Stanisław Oziemczuk

- Ujednolicenie logiki dodawania spota do ulubionych na froncie oraz dodanie API do sprawdzania, czy dany spot znajduje się już w ulubionych, co pozwoliło usunąć stare, rozproszone funkcje.
- Rozszerzenie obsługi dodawania mediów do spotów po stronie backendu i frontendu oraz aktualizacja walidacji i kontraktów API.
- Aktualizacja konfiguracji narzędzia ESLint pod kątem obsługi TypeScript, co poprawiło jakość statycznej analizy kodu frontendu.

TODO: odwołanie do odpowiedniego rozdziału i podrozdziałów poświęconych analizie wymagań.
TODO: dokończenie

Grudzień 2025

Adam Langmesser

- Opracowanie podrozdziału poświęconego wymaganiom dla modułu czatu, w tym wymagań funkcjonalnych i pozafunkcjonalnych.
- Dopracowanie kart usług zewnętrznych (m.in. usług mapowych, pogodowych, poczty e-mail) wykorzystywanych przez system.

TODO: odwołanie do numeru podrozdziału z wymaganiami dla czatu.
TODO: odwołanie do sekcji z kartami usług zewnętrznych.
TODO: dokończenie

Styczeń 2026

TODO: do uzupełnienia

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

Rozdział 7

Realizacja Projektu

7.1 Implementacja backendu

7.1.1 Struktura projektu

7.1.2 Integracja z bazą danych

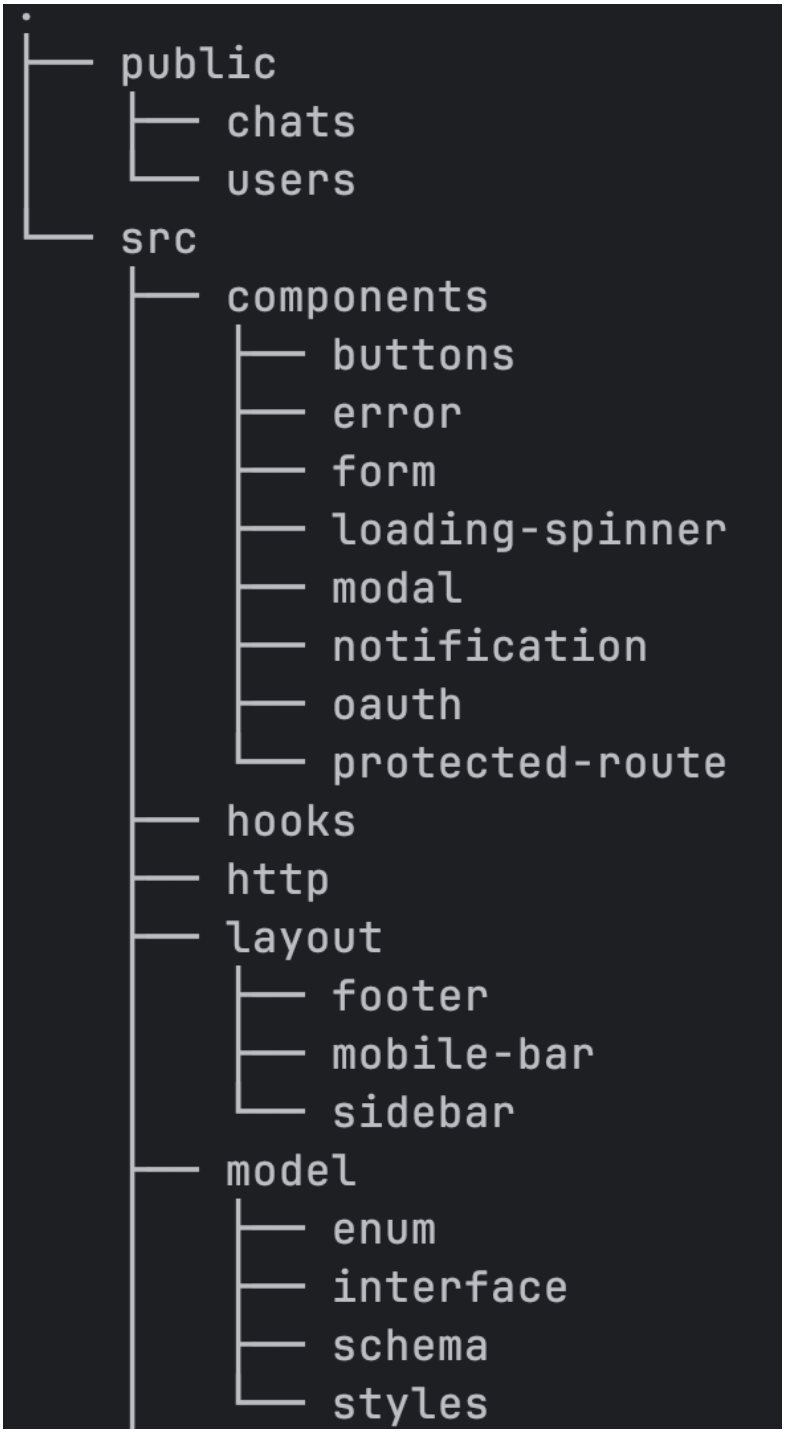
7.1.3 Obsługa uwierzytelnienia

7.1.4 Konteneryzacja

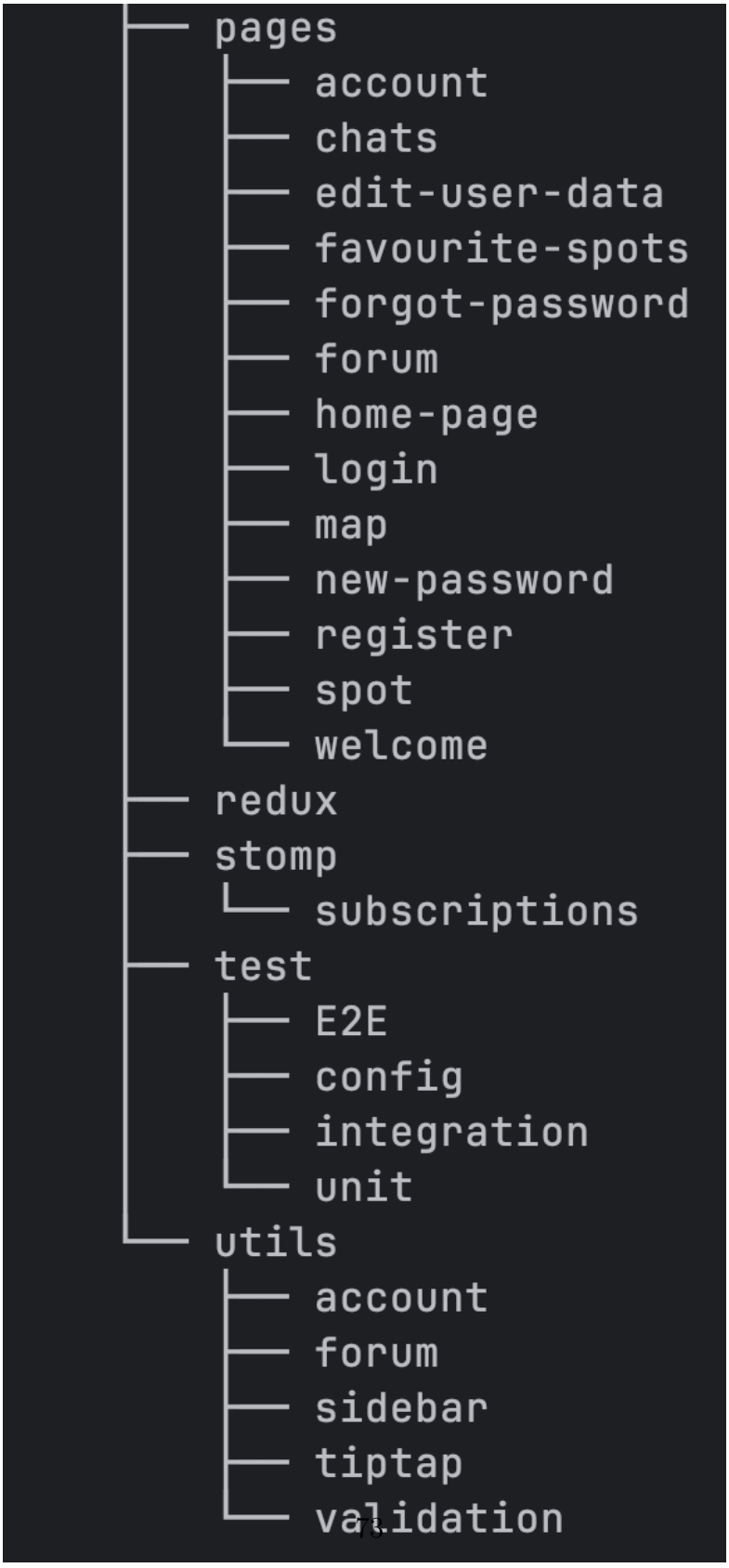
7.2 Implementacja frontendu

7.2.1 Struktura aplikacji

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



Rysunek 7.1: Struktura katalogów (1)



Rysunek 7.2: Struktura katalogów (2)

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

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

Rysunek 7.3: Implementacja routera (1)

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

export default router;
```

Rysunek 7.4: Implementacja routera (2)

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

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

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

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

    return isLogged ? 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)a `useState`) [14], jak i ze Stanu globalnego, utrzymywanego przez Bibliotekę React Redux [15]. Globalny Stan został wprowadzony po to, aby możliwie najbardziej ograniczyć przekazywanie Propsów w głąb drzewa komponentów oraz uniknąć niepotrzebnych ponownych renderów.

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

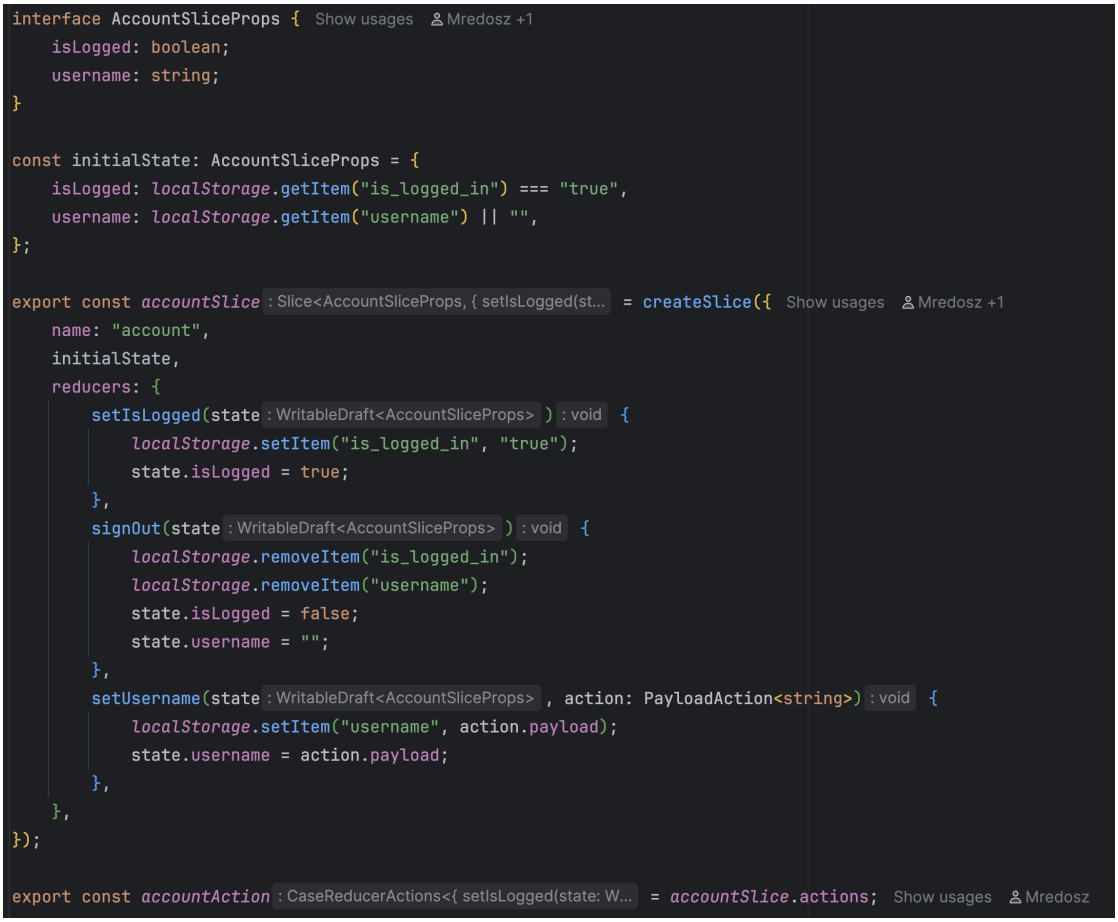
Z racji tego, że korzystamy z Reacta w połączeniu z TypeScriptem, przygotowaliśmy również własne Hook (React)i wspomagające typowanie, takie jak `useDispatchTyped` oraz `useSelectorTyped`. Pozwalają one na bezpieczne typo-

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`)



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

7.2.3 Integracja i komunikacja z backendem

Jest to kluczowy element aplikacji, ponieważ wymaga bezpiecznego przesyłania danych użytkownika. W celu uproszczenia komunikacji z serwerem skorzystaliśmy z biblioteki `axios` [16] oraz Biblioteki `TanStack Query` [17]. We wszystkich ścieżkach, które wymagają aby użytkownik był zalogowany, do zapytania dołączany jest token JWT. Token jest przekazywany w ciasteczku dzięki ustawieniu parametru `withCredentials` na wartość `true`. Przykładem pliku odpowiedzialnego za taką komunikację jest `account.js` (rys. 7.8 i 7.9), który obsługuje operacje związane z

logowaniem, rejestracją, zmianą hasła oraz wylogowaniem.

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

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

export async function 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)

Rysunek 7.9: Implementacja modułu `account` (2)

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

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

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

Rysunek 7.10: Wykorzystanie TanStack Query przy logowaniu użytkownika

7.2.4 Style

Do stylowania interfejsu wykorzystaliśmy Framework Tailwind CSS [18]. Dzięki gotowym klasom udostępnianym przez Tailwind mogliśmy definiować wygląd elementów bezpośrednio w kodzie komponentu, bez konieczności przechodzenia do osobnych plików ze stylami. Ułatwia to zarówno tworzenie widoków, jak i późniejsze modyfikacje — w przypadku zmiany stylu dokładnie wiadomo, gdzie należy jej dokonać. Korzystanie ze zdefiniowanych klas pozwoliło nam również zachować spójność wizualną w całej aplikacji. W pliku `index.css` zdefiniowaliśmy zmienne kolorystyczne (rys. 7.11 i 7.12). Dzięki temu zmiana motywu kolorystycznego w przyszłości sprowadza się do edycji wartości w jednym miejscu.

Rysunek 7.11: Implementacja zmiennych kolorystycznych (1)

Rysunek 7.12: Implementacja zmiennych kolorystycznych (2)

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

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

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

Rysunek 7.13: Przykładowe użycie klas Tailwind (w tym prefiksów responsywności)

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

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

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

Aby uzyskać płynniejsze i przyjemniejsze animacje, wykorzystaliśmy Bibliotekę Motion [19]. Pozwala ona w prosty sposób tworzyć animacje elementów interfejsu, bez potrzeby ręcznego pisania złożonych reguł CSS. W naszej aplikacji użyliśmy jej m.in. w polach formularza logowania i rejestracji (rys. 7.15). Na początku etykieta pola (np. „username") jest wyświetlana wewnątrz pola tekstowego, natomiast po

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

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

Rysunek 7.15: Implementacja animacji z wykorzystaniem Motion

7.2.5 Wyszukiwarka spotów

Niniejszy rozdział opisuje sposób implementacji wyszukiwarki spotów.

Jednym z głównych modułów aplikacji jest wyszukiwarka spotów, która umożliwia użytkownikowi szybkie odnalezienie interesujących lokalizacji. Funkcjonuje ona w dwóch wariantach: prostym i zaawansowanym (rys. 7.16 oraz 7.17).

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

Rysunek 7.16: Implementacja prostej wersji wyszukiwarki

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

Rysunek 7.17: Implementacja zaawansowanej wersji wyszukiwarki

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

```
<div className="dark:shadow-darkBgSoft flex rounded-full shadow-lg shadow-black/20">
    <NavLink
        to="/"
        className={({ isActive } : NavLinkRenderProps ) : string  =>
            `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 spoty po lokalizacji (kraj, region, miasto).

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

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

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

Rysunek 7.19: Implementacja karuzeli z najpopularniejszymi spotami

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

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

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

W skład zaawansowanej wersji wchodzą następujące komponenty:

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

Komponent `Switch` (rys. 7.18) zawiera dwa elementy `NavLink` z biblioteki React Router, co pozwala na przełączanie widoków bez konieczności przeładowywania całej strony.

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

```
<div className="dark:bg-darkBgSoft bg-lightBgSoft flex w-full flex-col items-center justify-between
 space-y-3 rounded-md px-3 py-2 shadow-md md:flex-row md:space-y-0 lg:w-3/4 lg:space-x-3 xl:w-1/2
 dark:shadow-black">
    <div className="flex w-full flex-col space-y-2">
        <h1>Location</h1>
        <div className="flex w-full flex-col space-y-3 md:flex-row md:space-y-0 md:space-x-2">
            {inputList.map(({ id, label } : { readonly label: "Your Country"; readonl... ) : Element  => (
                <div key={id} className="relative w-full">
                    <SearchInput
                        label={label}
                        id={id}
                        value={searchLocation[id] ?? ""}
                        onChange={(e : ChangeEvent<HTMLInputElement> ) : void  =>
                            handleSetLocation(id, e.target.value)
                        }
                        onFocus={() : void  => setActiveInput(id)}
                    />
                    {activeInput === id && suggestions.length > 0 && (
                        <SearchSuggestions
                            suggestions={suggestions}
                            onClick={handleSuggestionClick}
                            id={id}
                            onClose={() : void  => setActiveInput(null)}
                        />
                    )}
                </div>
            ))}
        </div>
    </div>
    <button
        className="dark:bg-darkBgMuted dark:hover:bg-darkBgMuted/80 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>
```

Rysunek 7.20: Implementacja prostej wyszukiwarki

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

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

Rysunek 7.21: Implementacja listy do wyświetlania spotów

Komponent `SpotTile` zawiera następujące informacje:

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

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

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

7.2.6 Mapa

7.2.7 Chat

7.2.8 Forum

7.2.9 Konto użytkownika

7.2.10 Panel logowania

7.3 Implementacja CI/CD

Rozdział 8

Testy

8.1 Testy jednostkowe

8.2 Testy integracyjne

8.3 Testy E2E

8.4 Wyniki testów i wnioski

Rozdział 9

Prezentacja systemu

9.1 Strona główna

9.2 Strona mapy

9.3 Strona chatu

9.4 Strona forum

9.5 Panel logowania

9.6 Panel konta użytkownika

Rozdział 10

Nakład pracy

10.1 Ogólny nakład pracy

10.2 Indywidualne nakłady pracy

10.2.1 Adam Langmesser

10.2.2 Mateusz Redosz

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

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

- Propozycja wyglądu

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

- Strona główna z prostymi filtrami
- Strona główna z zaawansowanymi filtrami
- Sidebar
- Strona profilu
- Ustawienia
- Listy spotów
- Lista zdjęć
- Lista filmów
- Lista znajomych
- Dodanie spotów
- Lista komentarzy
- Strona główna profilu
- Listy
- Poprawa Sidebara
- Zmiana kropki na przyciemnienie tła na Sidebar
- Poprawa strony do logowania i rejestracji
- Usunięcie username z account Redux
- Dodanie zamknięcia Sidebara na małych ekranach po kliknięciu nav linka
- Poprawić tooltipa na sidebar
- Zmiana sposobu pobierania username na backendzie z tokena jwt
- Paginacja z infinity scrollem
- Lista zdjęć innego usera
- Walidacja i responsywność w dodaniu spotów
- Dodanie sortowania i filtrów na zaawansowanej stronie
- Zmiana na infinity scrola

- Zmiana zdjęcia profilowego użytkownika
- Czyszczenie formularza w dodawaniu spota
- Dodanie wyszukiwarki znajomych w Social
- Zatwierdzenie przez drugiego użytkownika dodania do znajomych
- Sprawdzenie czy wszystko działa i poprawki Mateusz

4. CI/CD
    - Dodanie testów z frontendu do github actions
    - Poprawa github actions
    - Poprawa pipeline od Javy i Reacta
5. Praca dyplomowa
    - Uzupełnienie informacji o zespole i podział na rozdziały

10.2.3 Stanisław Oziemczuk

10.2.4 Kacper Badek

Rozdział 11

Podsumowanie

11.1 Osiągnięte rezultaty

11.2 Napotkane wyzwania

11.3 Plany na przyszłość

Rozdział 12

Słownik pojęć i skrótów

API

(ang. *application programming interface*); zbiór reguł i operacji do komunikacji z oprogramowaniem.. 15, 16, 56

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.. 60

Backend

Część aplikacji odpowiedzialna za logikę biznesową, przetwarzanie danych i komunikację z bazą danych. Działa po stronie serwera i obsługuje żądania wysyłane przez frontend. 2, 14, 96

Backlog

Lista zadań, które należy wykonać w ramach projektu, używane w metodykach zwinnych.. 14

Biblioteka

Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. 74, 76, 78, 80, 83, 84, 89

BPMN

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

CDN

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

CI/CD

Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. 15, 49, 51, 54, 59, 62, 68, 95, 98

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. 52

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. 52

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.. 83, 84

Design

Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). 95

Disciplined Agile Delivery - Lean Life Cycle

Disciplined Agile Delivery w wariancie Lean Life Cycle to sposób prowadzenia projektu, który łączy elastyczność Agile z przewidywalnością Waterfalla, ale bez stałych sprintów — praca toczy się w ciągłym przepływie. Na starcie zakłada mocniejszą fazę przygotowawczą: doprecyzowanie zakresu, szkic architektury, identyfikację ryzyk i kryteria jakości. W realizacji następuje 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, 48

Docker

Platforma do konteneryzacji aplikacji. Pozwala uruchamiać oprogramowanie w lekkich, izolowanych kontenerach tworzonych na podstawie obrazów, co upraszcza wdrażanie i utrzymanie spójnego środowiska. 51, 57

Droniarz

Potoczne określenie osoby, która jest jednocześnie pilotem oraz operatorem drona. Zwykle entuzjasta dronów.. 8, 9, 111

Droniarz foto/video

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

DTO

Skrót od *Data Transfer Object*. Prosty obiekt przenoszący dane między warstwami systemu lub między usługami. Zawiera pola danych, zazwyczaj bez logiki biznesowej. 67

ESLint

Statyczny analizator kodu JavaScript/TypeScript. Umożliwia wykrywanie błędów, niespójności stylu oraz potencjalnych problemów poprzez zestaw reguł, które można dostosować do projektu. 52, 70

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. 71

Framework

Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. 2, 81

Frontend

Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. 2, 14, 96

GitHub

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

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`).. 76, 80

IDE

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

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. 63, 65, 69, 90, 104

Intersection Observer

API przeglądarkowe umożliwiające reagowanie na momenty, gdy dany element pojawia się w polu widzenia użytkownika (viewport) lub opuszcza je. Wykorzystywane m.in. do implementacji Infinite scroll i lazy loadingu. 64, 65

Jira

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

JWT

Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procesach autoryzacji użytkowników. 52–54, 78, 95

LaTeX

System składu tekstu wykorzystywany do przygotowywania profesjonalnych dokumentów technicznych i naukowych. Umożliwia precyzyjne formatowanie, zarządzanie odwołaniami, bibliografią i wzorami matematycznymi. 66

Leaflet

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

MapLibre

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

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*).. 107

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. 66

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. 51, 52, 55

PANSA

Polish Air Navigation Services Agency, pol. Polska Agencja Żeglugi Powietrznej. Instytucja ta zapewnia m.in. mapę z zaznaczonymi strefami lotów. Każda strefa ma swoje właściwości prawne. . 20

Prettier

Narzędzie do automatycznego formatowania kodu (np. JavaScript, TypeScript, CSS, HTML). Narzuca spójny styl formatowania, zastępując ręczne ustawianie wcięć i łamań linii. 50, 62

Props

Właściwości przekazywane do komponentu React przez komponent nadrzędny; służą do konfiguracji i przekazywania danych. Powinny być traktowane jako tylko do odczytu (read-only) wewnątrz komponentu potomnego.. 76

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). 75

React

Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. 49, 76

React Query

Dawna nazwa biblioteki TanStack Query. Biblioteka dla Reacta wspierająca pobieranie, cachowanie i synchronizację danych z API oraz obsługę stanów ładowania i błędów w komponentach. 64

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.. 54, 56, 58, 60, 76, 77

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.. 84

REST API

Architektura budowania usług sieciowych komunikujących się poprzez metody protokołu HTTP (GET, PUT, POST, DELETE, PATCH). Wymiana danych występuje często w formacie JSON lub XML.
REST API musi spełniać następujące reguły:

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

5. **Warstwowość** — klient nie wie czy komunikuje się bezpośrednio z serwerem, czy poprzez pośrednika (np. proxy) oraz nie wie z czym komunikuje się obsługująca go warstwa.
6. **Kod na żądanie (opcjonalnie)** — serwer może przesłać fragment kodu, który zostanie wykonany przez klienta.

14, 49

Review kodu

Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. 14, 15, 48, 95

Sidebar

Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. 46, 95–97

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. 69

Spot

Spotkanie zespołu projektowego, zazwyczaj krótkie i regularne, służące omówieniu postępów prac, problemów oraz planów na najbliższy okres. 87–92

Spring Boot

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

Spring Security

Moduł bezpieczeństwa w ekosystemie Spring odpowiedzialny za uwierzytelnianie i autoryzację użytkowników. Zapewnia obsługę różnych mechanizmów

logowania, ról i uprawnień oraz integrację z różnymi źródłami danych. 50, 53, 56

Stan

Aktualny zestaw danych przechowywanych przez aplikację lub komponent, na podstawie którego renderowany jest interfejs użytkownika. Stan może być lokalny (utrzymywany w pojedynczym komponencie) albo globalny (wspólny dla wielu komponentów).. 76

Tablica Kanban

Narzędzie do zarządzania przepływem pracy, które pomaga zespołom śledzić zadania oraz ich postępy. Składa się z kolumn reprezentujących stan etapu prac, na przykład „Do zrobienia" lub „W trakcie".. 14, 49

Tailwind CSS

Framework CSS typu *utility-first*, dostarczający gotowe klasy narzędziowe do określania wyglądu (kolory, odstępy, layout). Umożliwia szybkie prototypowanie i spójne stylowanie komponentów bez pisania rozbudowanych arkuszy CSS. 50, 57, 58

TanStack Query

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

Testy E2E

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

TypeScript

Rozszerzenie do języka JavaScript dodający statyczne typowanie, interfejsy i narzędzia do większych projektów. Kompiluje się do czystego JavaScript,

ułatwiając wykrywanie błędów w czasie kompilacji i refaktoryzację.. 49, 57–59, 70, 76

UI

Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. 15, 76

UML

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

WebSocket

Protokół komunikacyjny umożliwiający dwukierunkową komunikację w czasie rzeczywistym między przeglądarką a serwerem po pojedynczym, utrzymywanym połączeniu TCP. Często wykorzystywany m.in. w czatach i aplikacjach działających „na żywo”. 61, 62

z-index

Właściwość CSS określająca kolejność nakładania się elementów (oś Z). Wyższa wartość powoduje wyświetlenie elementu „nad” elementami o niższych wartościach. 66

Spis tabel

Bibliografia

[1] *Disciplined Agile Delivery*. PMI. 1 stycznia 2025. URL: https://www.pmi.org/disciplined-agile/process/introduction-to-dad/why (dostęp 30.10.2025).

[2] *Disciplined Agile Delivery — Lean Life Cycle*. PMI. 1 stycznia 2025. URL: https://www.pmi.org/disciplined-agile/lifecycle/lean-lifecycle (dostęp 30.10.2025).

[3] Stanisław Wrycza, Bartosz Marcinkowski i Krzysztof Wyrzykowski. „Język UML 2.0 w modelowaniu systemów informatycznych". Warszawa: Helion, 2006. ISBN: 83-736-1892-9, 8373618929.

[4] Michał Wolski. *10 wskazówek poprawiających modelowanie procesów biznesowych w notacji BPMN*. 14 maja 2024. URL: https://wolski.pro/2024/05/10-wskazowek-poprawiajacych-modelowanie-procesow-biznesowych-w-notacji-bpmn/ (dostęp 19.11.2025).

[5] *About billing for GitHub Actions*. GitHub Docs. 1 stycznia 2024. URL: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions (dostęp 2.11.2025).

[6] *Scalability and performance targets for Blob storage*. Microsoft Learn. 1 stycznia 2024. URL: https://learn.microsoft.com/azure/storage/blobs/scalability-targets (dostęp 2.11.2025).

[7] *What are the limitations in Mailtrap?* Mailtrap Docs. 1 stycznia 2024. URL: https://help.mailtrap.io/article/111-what-are-the-limitations-in-mailtrap/ (dostęp 2.11.2025).

[8] *LocationIQ Pricing*. LocationIQ. 1 stycznia 2024. URL: https://locationiq.com/pricing (dostęp 2.11.2025).

[9] *Google Maps (Maps URLs)*. Google Maps. 1 stycznia 2024. URL: https://developers.google.com/maps/documentation/urls/get-started?hl=pl (dostęp 2.11.2025).

[10] *OpenFreeMap Documentation*. OpenFreeMap. 1 stycznia 2024. URL: https://openfreemap.org/docs (dostęp 2.11.2025).

[11] *Open-Meteo API Usage & Pricing*. Open-Meteo. 1 stycznia 2024. URL: https://open-meteo.com/en/docs/usage-and-pricing (dostęp 2.11.2025).

[12] *Tenor API — Documentation*. Tenor. 1 stycznia 2024. URL: https://tenor.com/gifapi/documentation (dostęp 2.11.2025).

[13] *Where the ISS at? API*. wheretheiss.at. 1 stycznia 2024. URL: https://wheretheiss.at/ (dostęp 2.11.2025).

[14] *React useState*. 1 stycznia 2025. URL: https://react.dev/reference/react/useState (dostęp 3.11.2025).

[15] *Redux*. 1 stycznia 2025. URL: https://redux.js.org/ (dostęp 3.11.2025).

[16] *Axios*. 1 stycznia 2025. URL: https://axios-http.com/ (dostęp 3.11.2025).

[17] *Tanstack Query*. 1 stycznia 2025. URL: https://tanstack.com/query/latest (dostęp 3.11.2025).

[18] *Tailwind*. 1 stycznia 2025. URL: https://tailwindcss.com/ (dostęp 3.11.2025).

[19] *Motion*. 1 stycznia 2025. URL: https://motion.dev/ (dostęp 3.11.2025).

Załączniki

Płyta CD z następującą zawartością:

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