



POLSKO-JAPONSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Filia w Gdańsku

Langmesser Adam

Nr albumu s27119

Nazwa specjalizacji: Aplikacje Internetowe

Redosz Mateusz

Nr albumu s27094

Nazwa specjalizacji: Aplikacje Internetowe

Oziemczuk Stanisław

Nr albumu s26982

Nazwa specjalizacji: Aplikacje Internetowe

Badek Kacper

Nr albumu s29168

Nazwa specjalizacji: Aplikacje Internetowe

Aplikacja webowa: spoty-na-drony.pl

Rodzaj pracy

inżynierska

Imię i nazwisko promotora

mgr Adam Urbanowicz

Gdańsk, miesiąc, 2100 obrony

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

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

Słowa kluczowe: — brak —



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Karta projektu

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

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

Data ukończenia projektu: 12 listopada 2025	Recenzent: dr Elżbieta Puźniakowska-Gałuch
---	--

Spis treści

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

3.3.2	Narzędzia	14
3.4	Zasoby i ograniczenia	17
3.4.1	Zasoby	17
3.4.2	Ograniczenia	17
3.4.3	Usługi zewnętrzne	17
3.5	Analiza ryzyka	20
4	Analiza wymagań	21
4.1	Przypadki użycia	21
4.1.1	Aktorzy	21
4.1.2	Diagram przypadków użycia	23
4.1.3	Scenariusz przypadków użycia	23
4.2	Wymagania ogólne i dziedzinowe	23
4.3	Wymagania funkcjonalne	23
4.3.1	Funkcjonalności dla mapy	23
4.3.2	Funkcjonalności dla chatu	23
4.3.3	Funkcjonalności dla forum	23
4.3.4	Funkcjonalności dla konta użytkownika	23
4.3.5	Funkcjonalności dla logowania i rejestracji	23
4.3.6	Funkcjonalności dla wyszukiwarki spotów	23
4.3.7	Funkcjonalności dla motywu	23
4.4	Wymagania pozafunkcjonalne	23
4.5	Wymagania interfejs z otoczeniem	23
4.6	Wymagania na środowisko docelowe	23
5	Projekt	24
5.1	Wzorce projektowe	24
5.2	Architektura systemu	24
5.2.1	Diagram architektury	24
5.2.2	Komponenty systemu	24
5.3	Projekt bazy danych	24
5.3.1	Model danych	24

5.3.2	Diagram ERD	24
5.4	Architektura interfejsu użytkownika	24
5.4.1	Projekt strony głównej	24
5.4.2	Projekt panelu logowania	24
5.4.3	Projekt mapy	24
5.4.4	Projekt chatu	24
5.4.5	Projekt forum	24
5.4.6	Projekt konta użytkownika	24
6	Przebieg realizacji projektu	25
6.1	Sprint 1	25
6.2	Sprint 2	25
7	Realizacja Projektu	26
7.1	Implementacja backendu	26
7.1.1	Struktura projektu	26
7.1.2	Endpointy systemu	26
7.1.3	Integracja z bazą danych	29
7.1.4	Obsługa uwierzytelnienia	29
7.1.5	Konteneryzacja	29
7.2	Implementacja frontendu	29
7.2.1	Struktura aplikacji	29
7.2.2	Zarządzanie stanem i przepływ danych	34
7.2.3	Integracja i komunikacja z backendem	36
7.2.4	Style	39
7.2.5	Strona główna	43
7.2.6	Mapa	43
7.2.7	Chat	43
7.2.8	Forum	43
7.2.9	Konto użytkownika	43
7.2.10	Panel logowania	43
7.3	Implementacja CI/CD	43

8 Testy	44
8.1 Testy jednostkowe	44
8.2 Testy integracyjne	44
8.3 Testy E2E	44
8.4 Wyniki testów i wnioski	44
9 Prezentacja systemu	45
9.1 Strona główna	45
9.2 Strona mapy	45
9.3 Strona chatu	45
9.4 Strona forum	45
9.5 Panel logowania	45
9.6 Panel konta użytkownika	45
10 Nakład pracy	46
10.1 Ogólny nakład pracy	46
10.2 Indywidualne nakłady pracy	46
10.2.1 Adam Langmesser	46
10.2.2 Mateusz Redosz	46
10.2.3 Stanisław Oziemczuk	49
10.2.4 Kacper Badek	49
11 Podsumowanie	50
11.1 Osiągnięte rezultaty	50
11.2 Napotkane wyzwania	50
11.3 Plany na przyszłość	50
12 Słownik pojęć i skrótów	51
Spis tabel	57
Bibliografia	58
Załączniki	60

Rozdział 1

Wstęp

1.1 O projekcie

1.2 Cel i zakres prac

1.3 Geneza pomysłu

Rozdział 2

Opis problemu

2.1 Rich picture

2.2 Udziałowcy

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

Tabela 2.1: Zespół projektowy

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

Tabela 2.2: Promotor

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

Tabela 2.3: Droniarze

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

Rozdział 3

Planowanie

3.1 Metodologia pracy

3.1.1 Przegląd rozważanych podejść

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

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

3.1.2 Odrzucone podejścia

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

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

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

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

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

Kluczowe argumenty wyboru:

- **Brak sprintów.** Zastosowano przepływ ciągły, co pozwala dopasować tempo do zmiennej dostępności zespołu i unikać sztucznego „domykania” iteracji.
- **Rozbudowana faza startowa.** Na początku przewidziano większy wysiłek planistyczny: doprecyzowanie zakresu, wstępna wizja architektury, identyfikacja ryzyk, plan publikacji oraz kryteria jakości – bez zamrażania szczegółów.
- **Ciągła weryfikacja wymagań.** W trakcie realizacji przewidziano bieżące doprecyzowywanie backlogu, regularny feedback promotora oraz możliwość korygowania kierunku bez kosztów „przeskakiwania” między fazami.
- **Praktyki Lean i koncentracja na wartości.** Priorytetyzacja wartości biznesowej, wizualizacja pracy, małe partie dostaw.
- **Lekka governance i kamienie milowe.** Zastosowano lekkie mechanizmy nadzoru (peer review, prezentacje postępów) zapewniające przejrzystość bez nadmiernej biurokracji.

3.1.4 Narzędzia i komunikacja

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

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

3.1.5 Podział ról w zespole

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

Każdy z członków zespołu uczestniczy również w przygotowaniu dokumentacji.

3.2 Harmonogram projektu

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

Rok 2024

Czerwiec • Zebranie zespołu.

- Rozważenie potencjalnych pomysłów.

Lipiec • Wybór technologii.

- Wstępne założenia architektoniczne.

Sierpień • *(do uzupełnienia)*

- *(do uzupełnienia)*

Wrzesień • *(do uzupełnienia)*

- *(do uzupełnienia)*

Październik • *(do uzupełnienia)*

- *(do uzupełnienia)*

Listopad • (do uzupełnienia)

- (do uzupełnienia)

Grudzień • (do uzupełnienia)

- (do uzupełnienia)

Rok 2025

Styczeń • (do uzupełnienia)

- (do uzupełnienia)

Luty • (do uzupełnienia)

- (do uzupełnienia)

Marzec • (do uzupełnienia)

- (do uzupełnienia)

Kwiecień • (do uzupełnienia)

- (do uzupełnienia)

Maj • (do uzupełnienia)

- (do uzupełnienia)

Czerwiec • (do uzupełnienia)

- (do uzupełnienia)

Lipiec • (do uzupełnienia)

- (do uzupełnienia)

Sierpień • (do uzupełnienia)

- (do uzupełnienia)

Wrzesień • (do uzupełnienia)

- (do uzupełnienia)

Październik • (*do uzupełnienia*)

- (*do uzupełnienia*)

Listopad • (*do uzupełnienia*)

- (*do uzupełnienia*)

Grudzień • (*do uzupełnienia*)

- (*do uzupełnienia*)

Rok 2026

Styczeń • (*do uzupełnienia*)

- (*do uzupełnienia*)

3.3 Technologie i narzędzia

Do realizacji projektu wykorzystano wiele technologii oraz narzędzi informatycznych. Przy wyborze technologii kierowaliśmy się ich popularnością, dostępnością dokumentacji oraz artykułów, a także łatwością użycia. Ponadto staraliśmy się korzystać z rozwiązań, które poznaliśmy w trakcie studiów. Narzędzia zostały dopasowane do wybranych technologii. 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 wielu dostępnym pluginom oferuje obsługę wielu języków programowania oraz innych składni. Pozwala również na integrację z repozytorium. Używamy 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. Wykorzystujemy je do lokalnego uruchamiania bazy danych oraz serwisu do cachowania.

- **One Drive**

Usługa dysku chmurowego oferowana przez firmę Microsoft. Przechowujemy 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 metodach 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 łączymy ją do głównej gałęzi deweloperskiej.

- **GitHub Actions**

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

- **GitHub Copilot**

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

- **Discord**

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

3.4 Zasoby i ograniczenia

3.4.1 Zasoby

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

3.4.2 Ograniczenia

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

3.4.3 Usługi zewnętrzne

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

Limit	3000 min/mies.
--------------	----------------

Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)

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

Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage

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

Tabela 3.3: Usługa zewnętrzna: Mailtrap

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

Tabela 3.4: Usługa zewnętrzna: LocationIQ

Usługa	Google Maps (Maps URLs) [7]
---------------	-----------------------------

Opis	Otwieranie nawigacji w aplikacji Map Google (deep link/URL).
Limit	Brak limitu w ramach dokumentowanego sposobu użycia.

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

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

Tabela 3.6: Usługa zewnętrzna: OpenFreeMap

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

Tabela 3.7: Usługa zewnętrzna: Open-Meteo

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

Tabela 3.8: Usługa zewnętrzna: Tenor GIF API

Usługa	Where the ISS at? [12]
---------------	------------------------

Opis	HTTP API z bieżącą pozycją satelity, używane pomocniczo.
Limit	1 zapytanie na sekundę; brak ogólnego limitu dziennego.

Tabela 3.9: Usługa zewnętrzna: Where the ISS at?

3.5 Analiza ryzyka

Rozdział 4

Analiza wymagań

4.1 Przypadki użycia

4.1.1 Aktorzy

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

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

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

Moderator Moderacja treści: posty na forum, komentarze spotów.

Deweloper Rozwija i utrzymuje system.

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

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

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

4.1.2 Diagram przypadków użycia

4.1.3 Scenariusz przypadków użycia

4.2 Wymagania ogólne i dziedzinowe

4.3 Wymagania funkcjonalne

4.3.1 Funkcjonalności dla mapy

4.3.2 Funkcjonalności dla chatu

4.3.3 Funkcjonalności dla forum

4.3.4 Funkcjonalności dla konta użytkownika

4.3.5 Funkcjonalności dla logowania i rejestracji

4.3.6 Funkcjonalności dla wyszukiwarki spotów

4.3.7 Funkcjonalności dla motywu

4.4 Wymagania pozafunkcjonalne

4.5 Wymagania interfejs z otoczeniem

4.6 Wymagania na środowisko docelowe

Rozdział 5

Projekt

5.1 Wzorce projektowe

5.2 Architektura systemu

5.2.1 Diagram architektury

5.2.2 Komponenty systemu

5.3 Projekt bazy danych

5.3.1 Model danych

5.3.2 Diagram ERD

5.4 Architektura interfejsu użytkownika

5.4.1 Projekt strony głównej

5.4.2 Projekt panelu logowania

5.4.3 Projekt mapy

5.4.4 Projekt chatu

5.4.5 Projekt forum

5.4.6 Projekt konta użytkownika

Rozdział 6

Przebieg realizacji projektu

6.1 Sprint 1

6.2 Sprint 2

Rozdział 7

Realizacja Projektu

7.1 Implementacja backendu

7.1.1 Struktura projektu

7.1.2 Endpointy systemu

GET /user-dashboard/profile

Opis: Zwraca profil aktualnie zalogowanego użytkownika.

Metoda: GET /user-dashboard/profile

Zwraca (200 OK): application/json — obiekt UserProfileDto.

Błąd (404 Not Found): text/plain —

komunikat z wyjątku UserNotFoundException.

Przykładowa odpowiedź (200 OK):

```
1  {
2      "username": "john_doe",
3      "profilePhoto": "https://cdn.example.com/profiles/john_doe.
4          jpg",
5      "followersCount": 125,
6      "followedCount": 87,
7      "friendsCount": 32,
8      "photosCount": 58,
9      "mostPopularPhotos": [
10         {
11             "src": "https://cdn.example.com/photos/123.jpg",
12             "heartsCount": 240,
```

```
12      "viewsCount": 3400,
13      "title": "Sunset at the beach",
14      "id": 123
15    }
16  ]
17 }
```

Example response (404 Not Found):

Panel użytkownika

- GET /user-dashboard/profile
- GET /public/user-dashboard/profile/"targetUsername"
- PATCH /user-dashboard/profile
- GET /user-dashboard/friends
- GET /public/user-dashboard/friends/"targetUsername"
- PATCH /user-dashboard/friends
- PATCH /user-dashboard/friends/change-status
- GET /user-dashboard/followers
- GET /public/user-dashboard/followers/"targetUsername"
- GET /user-dashboard/followed
- GET /public/user-dashboard/followed/"targetUsername"
- GET /user-dashboard/friends/find
- GET /user-dashboard/friends/invites
- PATCH /user-dashboard/followed
- GET /user-dashboard/favorite-spots
- PATCH /user-dashboard/favorite-spots

- GET /user-dashboard/photos
- GET /user-dashboard/comments
- PATCH /user-dashboard/settings
- GET /user-dashboard/settings
- GET /user-dashboard/movies
- GET /user-dashboard/photos/"targetUsername"
- GET /user-dashboard/add-spot
- POST /user-dashboard/add-spot
- GET /user-dashboard/add-spot/coordinates

Strona główna

- GET /public/spot/most-popular
- GET /public/spot/search/home-page
- GET /public/spot/search/home-page/locations
- GET /public/spot/search/home-page/advance

Konto użytkownika

- POST /public/account/register
- POST /public/account/login
- GET /account/login-success
- POST /public/account/forgot-password
- POST /public/account/set-new-password
- GET /account/check

7.1.3 Integracja z bazą danych

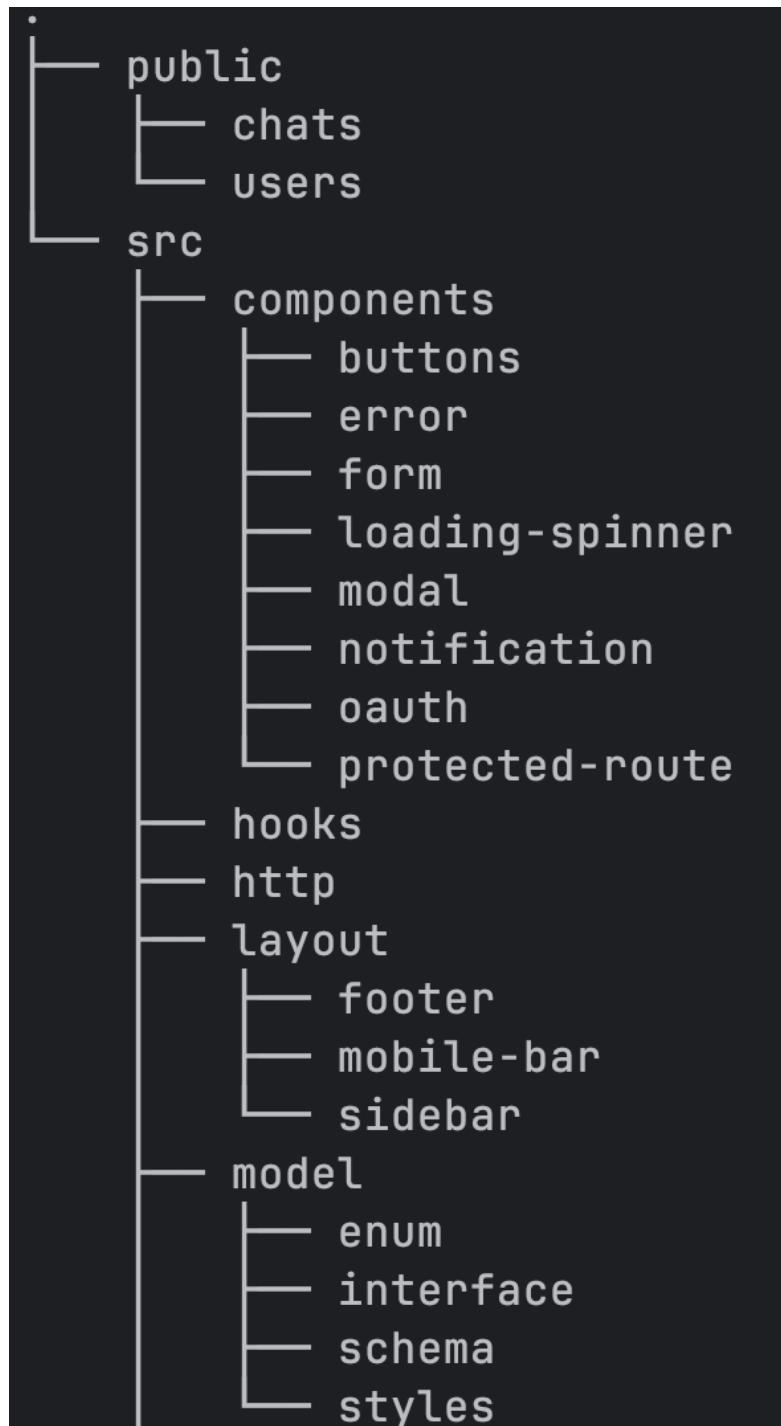
7.1.4 Obsługa uwierzytelnienia

7.1.5 Konteneryzacja

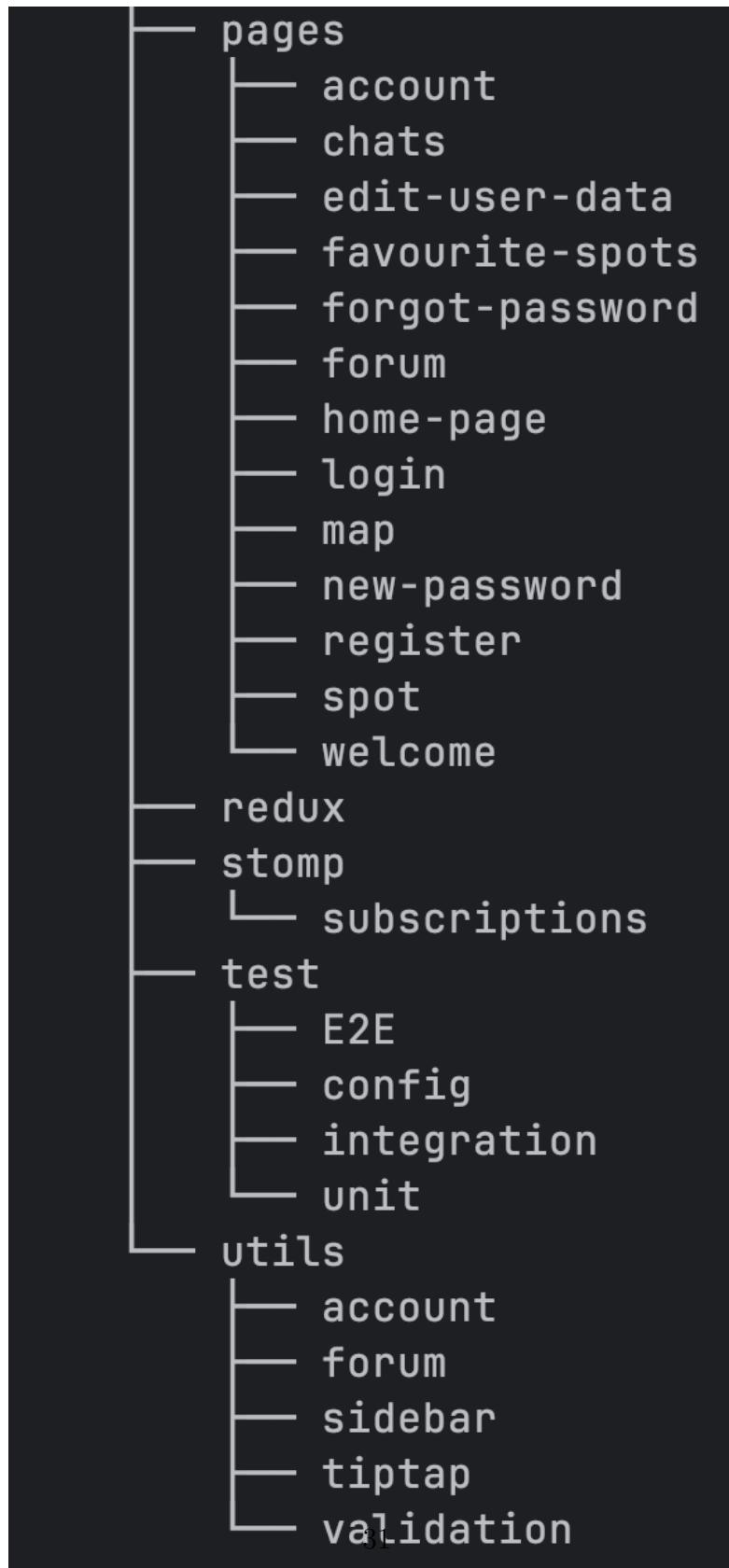
7.2 Implementacja frontendu

7.2.1 Struktura aplikacji

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



Rysunek 7.1: Struktura katalogów (1)



Rysunek 7.2: Struktura katalogów (2)

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

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

Rysunek 7.3: Implementacja routera (1)

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

export default router;
```

Rysunek 7.4: Implementacja routera (2)

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

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

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

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

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

Rysunek 7.5: Implementacja komponentu bramki (`ProtectedRoute`)

7.2.2 Zarządzanie stanem i przepływ danych

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

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

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

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

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

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

Rysunek 7.6: Konfiguracja sklepu (Redux store)

```

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

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

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

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

```

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

7.2.3 Integracja i komunikacja z backendem

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

logowaniem, rejestracją, zmianą hasła oraz wylogowaniem.

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

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

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

Rysunek 7.8: Implementacja modułu account (1)

```

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

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

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

```

Rysunek 7.9: Implementacja modułu `account` (2)

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

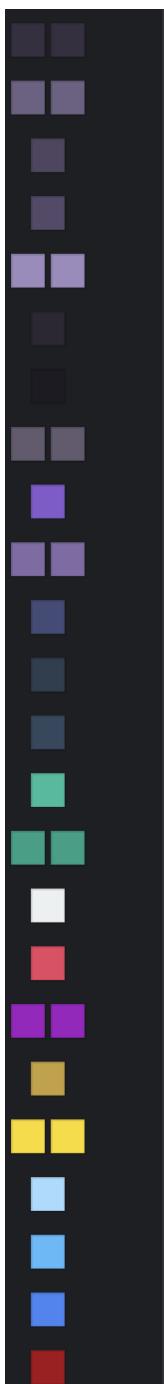
```
const { mutateAsync, isSuccess, error } = useMutation({
  mutationFn: loginUser,
});

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

Rysunek 7.10: Wykorzystanie TanStack Query przy logowaniu użytkownika

7.2.4 Style

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



	--color-violetDark: #363041;
	--color-violetLight: #6d6183;
	--color-violetLightDarker: #4f4660;
	--color-violetLightDark: #554a69;
	--color-violetLighter: #9b8cbd;
	--color-violetDarker: #2c2734;
	--color-violetHeavyDark: #1e1b23;
	--color-violetBtnBorderDark: #625b6e;
	--color-violetBright: #835ace;
	--color-darkVioletBtnOutline: #816ba6;
	--color-mediumDarkBlue: #424b77;
	--color-first: #2c3e50;
	--color-second: #34495e;
	--color-third: #1abc9c;
	--color-fourth: #16a085;
	--color-fifth: #ecf0f1;
	--color-sixth: #e94560;
	--color-magenta: #a01bc1;
	--color-darkYellow: #c5a03c;
	--color-ratingStarColor: #fadbd1;
	--color-locationMarkerDarkerBlue: #a3dcff;
	--color-locationMarkerLightBlue: #52bafb;
	--color-userLocationDot: #4285f4;
	--color-spotLocationMarker: #a8071a;

Rysunek 7.11: Implementacja zmiennych kolorystycznych (1)



Rysunek 7.12: Implementacja zmiennych kolorystycznych (2)

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

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

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

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

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

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

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

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

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

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

Rysunek 7.15: Implementacja animacji z wykorzystaniem Motion

7.2.5 Strona główna

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 rozpoczęłem 04.08.2024 a zakończyłem 08.09.2025. W projekcie pracowałem nad Rejestracją użytkownika, tokenem JWT, częściową implementacją CI/CD, stroną główną, zaimplementowaniem Sidebara oraz podstroną dla użytkownika. Moje wylistowane zadania z Jira:

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

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

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

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

4. CI/CD

- Dodanie testów z frontendu do github actions
- Poprawa github actions
- Poprawa pipeline od Javy i Reacta

5. Praca dyplomowa

- Uzupełnienie informacji o zespole i podział na rozdziały

10.2.3 Stanisław Oziemczuk

10.2.4 Kacper Badek

Rozdział 11

Podsumowanie

11.1 Osiągnięte rezultaty

11.2 Napotkane wyzwania

11.3 Plany na przyszłość

Rozdział 12

Słownik pojęć i skrótów

API

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

Backend

Część aplikacji odpowiedzialna za logikę biznesową, przetwarzanie danych i komunikację z bazą danych. Działa po stronie serwera i obsługuje żądania wysyłane przez frontend. 2, 15, 47

Backlog

Lista zadań, które należy wykonać w ramach projektu, używane w metodykach zwinnych.. 15

Biblioteka

Zewnętrzny lub wewnętrzny zestaw gotowych funkcji, klas, komponentów lub modułów, który można wielokrotnie wykorzystywać w projekcie zamiast pisać wszystko od zera. 32, 34, 36, 38, 41, 42

CI/CD

Skrót od *Continuous Integration/Continuous Deployment*. Praktyka programistyczna polegająca na automatyzacji procesu budowania, testowania i wdrażania oprogramowania. 16, 46, 49

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.. 41, 42

Design

Etap lub proces projektowania wyglądu i funkcjonalności aplikacji, obejmujący zarówno aspekty wizualne, jak i użytkowe (UX/UI). 46

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

Droniarz

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

Droniarz foto/video

Pilot wykorzystujący drony fotograficzne/filmowe do rejestracji materiałów wizualnych (zdjęcia, wideo), zwykle z naciskiem na stabilizację i jakość obrazu.. 17

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. 29

Framework

Zestaw narzędzi, bibliotek i struktur wspomagających tworzenie aplikacji. Ułatwia programowanie poprzez dostarczenie gotowych komponentów oraz określenie zasad organizacji kodu. 2, 39

Frontend

Warstwa aplikacji odpowiedzialna za interfejs użytkownika oraz interakcję z użytkownikiem. Zazwyczaj tworzona przy użyciu technologii takich jak HTML, CSS i JavaScript. 2, 15, 47

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`).. 34, 38

IDE

(ang. *integrated development environment*); to zintegrowane środowisko programistyczne, służące do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. 15

JWT

Skrót od *JSON Web Token*. Standard służący do bezpiecznego przekazywania informacji między stronami w formacie JSON, często używany w procedurach autoryzacji użytkowników. 36, 46

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*).. 54

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.. . 21

Props

Właściwości przekazywane do komponentu React przez komponent nadzędny; służą do konfiguracji i przekazywania danych. Powinny być traktowane jako tylko do odczytu (read-only) wewnątrz komponentu potomnego.. 34

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). 33

React

Biblioteka JavaScript do budowy interfejsów użytkownika w oparciu o komponenty deklaratywne i wirtualny DOM. Zapewnia jednokierunkowy przepływ danych oraz zarządzanie stanem komponentów.. 34

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.. 34, 35

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.. 42

REST API

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

REST API musi spełniać następujące reguły:

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

Review kodu

Proces polegający na wzajemnym przeglądzie kodu źródłowego przez programistów w celu wykrycia błędów, poprawy jakości oraz zwiększenia spójności projektu. 15, 16, 46

Sidebar

Boczny panel w interfejsie użytkownika, zawierający menu nawigacyjne lub dodatkowe opcje funkcjonalne aplikacji. 46–48

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).. 34

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 „To do” lub „In progress”.. 15

TypeScript

Rozszerzenie do języka JavaScript dodający statyczne typowanie, interfejsy i narzędzia do większych projektów. Kompiluje się do czystego JavaScript, ułatwiając wykrywanie błędów w czasie kompilacji i refaktoryzacji.. 34

UI

Interfejs użytkownika (ang. *User Interface*); warstwa prezentacji odpowiedzialna za sposób wyświetlania danych oraz interakcji użytkownika z aplikacją.. 17, 34

Spis tabel

2.1 Zespół projektowy	7
2.2 Promotor	8
2.3 Droniarze	8
Tabela 3.1: Usługa zewnętrzna: GitHub Actions (CI)	18
Tabela 3.2: Usługa zewnętrzna: Azure Blob Storage	18
Tabela 3.3: Usługa zewnętrzna: Mailtrap	18
Tabela 3.4: Usługa zewnętrzna: LocationIQ	18
Tabela 3.5: Usługa zewnętrzna: Google Maps (Maps URLs)	19
Tabela 3.6: Usługa zewnętrzna: OpenFreeMap	19
Tabela 3.7: Usługa zewnętrzna: Open-Meteo	19
Tabela 3.8: Usługa zewnętrzna: Tenor GIF API	19
Tabela 3.9: Usługa zewnętrzna: Where the ISS at?	20

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] *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).
- [4] *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).
- [5] *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).
- [6] *LocationIQ Pricing*. LocationIQ. 1 stycznia 2024. URL: <https://locationiq.com/pricing> (dostęp 2.11.2025).
- [7] *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).
- [8] *OpenFreeMap Documentation*. OpenFreeMap. 1 stycznia 2024. URL: <https://openfreemap.org/docs> (dostęp 2.11.2025).
- [9] *OpenFreeMap Quick Start*. OpenFreeMap. 1 stycznia 2024. URL: <https://openfreemap.org/docs/quick-start> (dostęp 2.11.2025).
- [10] *Open-Meteo API Usage & Pricing*. Open-Meteo. 1 stycznia 2024. URL: <https://open-meteo.com/en/docs/usage-and-pricing> (dostęp 2.11.2025).
- [11] *Tenor API — Documentation*. Tenor. 1 stycznia 2024. URL: <https://tenor.com/gifapi/documentation> (dostęp 2.11.2025).

- [12] *Where the ISS at? API.* wheretheiss.at. 1 stycznia 2024. URL: <https://wheretheiss.at/> (dostęp 2.11.2025).
- [13] *React useState.* 1 stycznia 2025. URL: <https://react.dev/reference/react/useState> (dostęp 3.11.2025).
- [14] *Redux.* 1 stycznia 2025. URL: <https://redux.js.org/> (dostęp 3.11.2025).
- [15] *Axios.* 1 stycznia 2025. URL: <https://axios-http.com/> (dostęp 3.11.2025).
- [16] *Tanstack Query.* 1 stycznia 2025. URL: <https://tanstack.com/query/latest> (dostęp 3.11.2025).
- [17] *Tailwind.* 1 stycznia 2025. URL: <https://tailwindcss.com/> (dostęp 3.11.2025).
- [18] *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.