



**WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ**
Uniwersytet Łódzki

Bartłomiej Płóciennik

.....
(imię i nazwisko)

Kierunek: informatyka
Specjalność: informatyka stosowana
Specjalizacja: Systemy i aplikacje mobilne
Numer albumu: 402150

"Re(me)mber" - multiplatformowa aplikacja do gromadzenia wspomnień.

.....
(tytuł pracy dyplomowej)

Praca inżynierska

wykonana pod kierunkiem
dr. Grzegorza Wieczorka
w Katedrze Systemów Inteligentnych
WFiIS UŁ

Łódź2025
(rok kalendarzowy)

Spis treści

Wstęp	4
Rozdział 1 Omówienie projektu	6
1.1 Geneza	6
1.2 Opis aplikacji	6
1.3 Opis działania	6
1.4 Przegląd istniejących rozwiązań	7
1.4.1 Zdjęcia Google	7
1.4.2 Journey: Diary, Journal, Notes.....	8
Rozdział 2 Analiza wymagań.....	10
2.1 Przypadki użycia	10
2.2 Wymagania funkcjonalne	10
2.3 Wymagania niefunkcjonalne	11
Rozdział 3 Fundamenty Projektu	13
3.1 Wykorzystane technologie i narzędzia.....	13
3.1.1 Visual Studio Code.....	13
3.1.2 Android Studio	13
3.1.3 Git	13
3.1.4 Google Maps Platform.....	14
3.1.5 Firebase	14
3.1.6 Google Forms	16
3.1.7 Gmail	16
3.1.8 Flutter SDK	17
3.2 Struktura plików w projekcie	17
3.2.1 Ogólna struktura plików	17
3.2.2 Struktura katalogu lib	18
Rozdział 4 Flutter	19
4.1 Tworzenie aplikacji z wykorzystaniem Flutter	19
4.2 Widgety i ich stan	19
4.3 Platforma pub.dev	20
4.4 Popularność	20
4.5 Ograniczenia Flutter dla aplikacji webowych	21
Rozdział 5 Użytkownik i interfejs	22

5.1	Idea tworzenia interfejsu	22
5.2	Motywy i schematy kolorystyczne	26
Rozdział 6 Struktura aplikacji		27
6.1	Ekran ładowania	27
6.2	Ekran Startowy	27
6.3	Ekran logowania/rejestracji	28
6.4	Ekran zawartości	29
6.4.1	Ekran galerii wspomnień	30
6.4.2	Ekran dodawania wspomnienia	32
6.4.3	Ekran mapy wspomnień	34
6.4.4	Szuflada użytkownika	35
6.5	Ekran mapy	37
6.5.1	Ekran mapy w trybie wyboru lokalizacji	37
6.5.2	Ekran mapy w trybie mapy wspomnień.....	38
6.5.3	Ekran mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia.....	38
6.6	Ekran szczegółów wspomnienia	38
6.7	Pozostałe elementy	41
6.7.1	Informacyjne okna dialogowe	41
6.7.2	Pola tekstowe	41
6.7.3	Panel informacyjny.....	42
6.7.4	Widget Infotext.....	42
6.7.5	Indykator ładowania	43
Rozdział 7 Wybrane elementy implementacji		44
7.1	Następowanie stanu uwierzytelnienia użytkownika	44
7.2	Obsługa wyjątków związanych z Firebase	44
7.3	Mapa nazw dla danych z Firebase	46
7.4	Cached Network Image.....	47
7.5	Zarządzenie stanem	48
7.6	Rozróżnianie platform.....	50
7.7	Implementacja ekranu galerii wspomnień	50
7.8	Dodawanie i usuwanie wspomnienia z ulubionych.....	54
7.9	Dodawanie szczegółów wspomnienia	55
7.9.1	Dodawanie zdjęcia	55
7.9.2	Dodawanie daty	56
7.9.3	Dodawanie lokalizacji	58
7.10	Wielokrotne wykorzystanie ekranu mapy	60

7.10.1	Implementacja ekranu mapy w trybie wyboru lokalizacji	61
7.10.2	Implementacja ekranu mapy wspomnień.....	63
7.10.3	Implementacja ekranu mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia	65
Rozdział 8 Problemy w Implementacji.....		66
8.1	Implementacja mapy	66
8.1.1	Brak ładowania mapy	66
8.1.2	Niewykrywanie włączonej lokalizacji	66
8.1.3	Brak trybu ciemnego dla interaktywnej mapy.....	66
8.2	Konieczność rozdzielenia logiki pobierania zdjęć między platformami.....	67
8.3	Ukrycie klucza API Map Google dla wersji webowej w repozytorium	68
Rozdział 9 Testy i Diagnostyka		69
9.1	Flutter DevTools	69
9.2	Testowanie aplikacji	70
9.2.1	Wersja na system Android	70
9.2.2	Wersja webowa.....	71
9.2.3	Wyniki testów	71
9.3	System automatycznego wykrywania błędów	71
Podsumowanie		73
Bibliografia		75
Spis rysunków.....		81

Wstęp

Wspomnienia stanowią jeden z najważniejszych elementów świadomości każdego człowieka. Pozwalają budować poczucie własnej tożsamości i samoświadomości, będąc przy tym kluczem nie tylko do przeszłości, ale także teraźniejszości i przyszłości.

Jednocześnie ciężko wyobrazić sobie współczesny świat bez nowoczesnych technologii. Komputery i urządzenia mobilne stały się nierozdzielną częścią ludzkiej egzystencji. Szacuje się, że aktualnie na świecie jest niespełna 4,5 miliarda użytkowników samych smartfonów [1], a prognozy przewidują dalsze wzrosty w przyszłości. Naturalnym wydaje się więc połączenie dwóch omawianych tematów.

W związku z tym cel niniejszej pracy stanowi stworzenie multiplatformowej aplikacji umożliwiającej gromadzenie wspomnień. Aplikacja ta powstawać będzie z myślą o systemie **Android** oraz wersji **webowej**. Podstawę jej implementacji stanowić będzie technologia **Flutter**, umożliwiająca tworzenie multiplatformowych aplikacji z pojedynczej bazy kodu. Podstawę przechowywania wspomnień i zarządzania użytkownikami stanowić będzie platforma **Firestore**.

Na pracę składać się będzie dziewięć głównych rozdziałów. Pierwszy z nich to dokładniejsze zaznajomienie z tworzoną aplikacją. Jej genezę, opisem, działaniem, a także podobnymi rozwiązaniami. Drugi rozdział to zapoznanie z przykładami użycia oraz wymaganiami funkcjonalnymi i niefunkcjonalnymi. Kolejny rozdział poświęcony zostanie przedstawieniu fundamentów projektu, uwzględniając najważniejsze wykorzystane technologie, narzędzia oraz strukturę plików. Rozdział czwarty w całości skupiać się będzie na opisanu technologii Flutter. Zostanie przedstawiona pod kątem jej podstawowych założeń, zestawiona z innymi rozwiązaniami oraz omówione zostaną niedogodności związane z tworzeniem aplikacji webowych przy jej pomocy. W piątym rozdziale omówione zostaną ogólne informacje na temat interfejsu aplikacji, związane z ideą jego projektowania i schematami kolorystycznymi. Szósty rozdział to zagłębienie się w strukturę aplikacji. Poszczególne ekrany oraz najważniejsze elementy aplikacji zostaną dokładniej omówione pod kątem interfejsu oraz dostępnych funkcjonalności. Dla większej czytelności opisy te dotyczyć będą wersji aplikacji na system Android, wykorzystującej tryb ciemny. Siódmy rozdział poświęcony zostanie analizie najciekawszych aspektów związanych z implementacją aplikacji od strony programistycznej. Rozdział ósmy opisywać będzie wybrane problemy napotkane w czasie implementacji oraz to, jak zostały one rozwiązane. Z kolei dziewiąty rozdział poświęcony zostanie metodyce testowania i diagnozowania aplikacji. Omówione zostaną wykorzystane platformy testowe, wyniki testów i zaimplementowane systemy

radzenia sobie z błędami aplikacji. Wymienione rozdziały zostaną zwieńczone podsumowaniem zawierającym obserwacje powstałe w czasie implementacji oraz dalsze plany rozwoju omawianej aplikacji.

Rozdział 1

Omówienie projektu

1.1 Geneza

Geneza pracy związana jest bezpośrednio z projektem **StartIT**, odbywającym się w Antwerpii w marcu 2024 roku. Był to czas, w którym nieustannie powstawały nowe, niezwykle wspomnienia. Tworzona była także niezwykle rozbudowana fotorelacja z wydarzenia. Pojawiło się jednak ryzyko, że wiele z tych wspaniałych chwil może z czasem zwyczajnie utracić swoje dawne rezonowanie. Logicznym stała się zatem chęć cyfrowego ich utrwalenia.

1.2 Opis aplikacji

Aplikacja **Re(me)mber** powstaje z myślą o urządzeniach z systemem Android oraz platformie webowej. Ma to być narzędzie dla osób w każdym wieku, które umożliwi im łatwe gromadzenie wspomnień z wykorzystaniem najnowszych technologii. Skupiając się początkowo na rodzimych użytkownikach, aplikacja będzie wykorzystywać polską wersję językową.

Pełnię funkcjonalności oferować będzie wersja aplikacji na system Android. Ma umożliwiać przeglądanie i zarządzanie wspomnieniami oraz, co najważniejsze, ich zapisywanie. Podawane przy uwiecznianiu wspomnienia informacje (**tytuł, opis, data, zdjęcie i lokalizacja**) mają zapewnić jak najdokładniejsze utrwalenie tego ulotnego momentu. Wersja webowa ma pełnić funkcję swego rodzaju klienta, który umożliwi przeglądanie i zarządzanie zgromadzonymi wspomnieniami z poziomu przeglądarki internetowej.

Dzięki przypisaniu wspomnień do konta i przechowywaniu ich w chmurze użytkownicy będą mogli uzyskać do nich dostęp niezależnie od urządzenia i platformy, na której używają aplikacji. Dodatkowo użytkownicy będą mieli do dyspozycji mapę zawierającą znaczniki reprezentujące zapisane wspomnienia, która będzie stanowić swego rodzaju historię odwiedzonych miejsc.

1.3 Opis działania

Do rozpoczęcia gromadzenia wspomnień użytkownik musi utworzyć konto. Może to zrobić z wykorzystaniem nazwy użytkownika, adresu e-mail i hasła. Po pomyślnym zalogowaniu/rejestracji otrzymuje on dostęp do wszystkich funkcjonalności aplikacji charakterystycznych dla platformy, z której aktualnie korzysta.

Wybierając odpowiednią pozycję z paska nawigacyjnego użytkownik może:

1. Przeglądać zgromadzone wspomnienia oraz wyświetlać ich szczegóły. Z poziomu szczegółów wspomnienia może dodać wspomnienie do ulubionych, wymazać je lub pobrać powiązane ze wspomnieniem zdjęcie do pamięci urządzenia.
2. Przeglądać mapę zgromadzonych wspomnień, zawierającą znaczniki reprezentujące zapisane przez użytkownika wspomnienia.
3. **W wersji aplikacji na system Android** – zapisać nowe wspomnienie przy użyciu jego tytułu, opisu, lokalizacji (wybór miejsca na mapie lub pobranie aktualnej lokalizacji), nowo wykonanego lub wczytanego z pamięci urządzenia zdjęcia oraz daty. Domyślnie podejmowana jest próba odczytania daty z danych **EXIF** [2] zdjęcia, a w przypadku niepowodzenia można wybrać datę ręcznie, przy użyciu odpowiedniego okna dialogowego.

Dodatkowo, wykorzystując specjalną szufladę, wysuwaną przy pomocy przycisku umieszczonego w lewym górnym rogu ekranu, użytkownik może:

1. Zmienić swoje hasło.
2. Zgłosić błąd przy użyciu specjalnego formularza.
3. Skontaktować się z pomocą techniczną poprzez wiadomość e-mail.
4. Wylogować się z aplikacji.

1.4 Przegląd istniejących rozwiązań

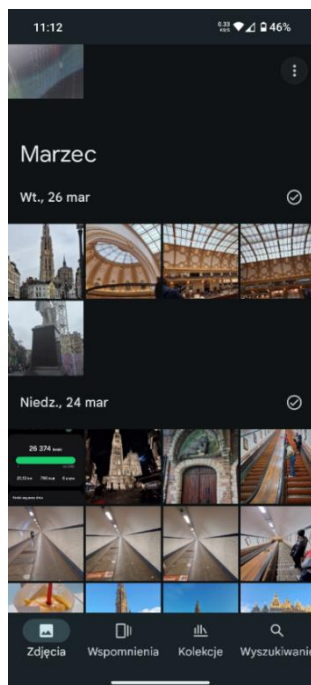
1.4.1 Zdjęcia Google

Widoczne na **Rysunku 1 Zdjęcia Google (Google Photos)** [3] to usługa/aplikacja firmy Google, która umożliwia przechowywanie, organizowanie czy edytowanie zdjęć i filmów. Na niektórych urządzeniach z systemem Android stanowi ona domyślną aplikację galerii. Dotyczy to szczególnie urządzeń cechujących się zastosowaniem możliwie „czystej” (wolnej od rozbudowanych modyfikacji graficznych i nieobciążonej nadmiarowym oprogramowaniem producenta) wersji systemu Android, takich jak linie **Google Pixel** czy **Nothing Phone**. Zakładka **Wspomnienia** w wersji aplikacji na system Android umożliwia tworzenie wspomnień w oparciu o wybrane zdjęcia/filmy.

Lokalizacja danego wspomnienia musi zostać wybrana przez użytkownika poprzez wprowadzenie nazwy miejsca, a nie wybór go z mapy czy pobranie aktualnej lokalizacji. W gruncie rzeczy, dodanie wspomnienia przypomina utworzenie specjalnej kategorii dla wybranych zdjęć/filmów.

Problematyczną pozostaje również kwestia tego jak Google Photos kategoryzuje zdjęcia. Domyślnie nie są wyświetlane wszystkie zdjęcia zapisane w pamięci urządzenia. Niektóre z nich trzeba wyszukać ręcznie, przechodząc przez kolejne kategorie i kolekcje. Toteż utworzenie wspomnienia z danego pliku często nie jest prostym zadaniem. Dodatkowo

wersja webowa Google Photos posiada zgoła inny interfejs. Użytkownik ma w niej dostęp do wszystkich zdjęć, dla których podjął decyzję o utworzeniu kopii zapasowej. W tej wersji nie występuje bezpośrednio zakładka **Wspomnienia**. Aspekty te mogą potencjalnie utrudnić szybkie odnalezienie wcześniej zapisanych wspomnień.



Rysunek 1
Zdjęcia Google w wersji na system Android

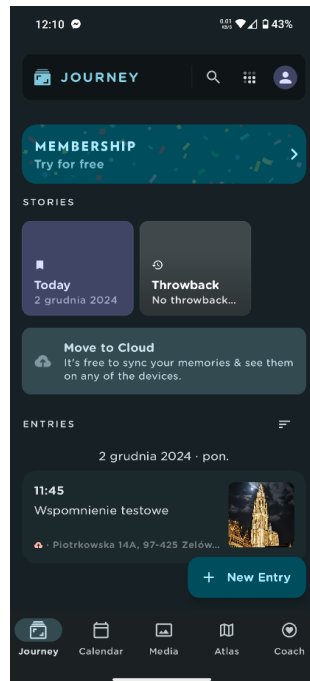
1.4.2 Journey: Diary, Journal, Notes

Widoczna na **Rysunku 2** aplikacja **Journey: Diary, Journal, Notes** [4] stanowi swego rodzaju cyfrowy dziennik/pamiętnik. Wpisy do niego mogą być urozmaicone o zdjęcia czy podanie lokalizacji. Dodatkowo aplikacja posiada system coachingowy, w którym to można skorzystać z różnego rodzaju, nastawionych na osobisty samorozwój, kursów, artykułów itp.

W wersji na system Android wpisy do dziennika przechowywane są domyślnie w pamięci lokalnej urządzenia. Opcję synchronizacji z chmurą należy skonfigurować ręcznie w ustawieniach aplikacji. W wersji webowej bez tej konfiguracji użytkownik nie może dodać nowego wpisu. Znaczna część funkcjonalności wymaga dodatkowego abonamentu, który to aplikacja uporczywie reklamuje użytkownikowi. Zgodnie z opisem abonamentów [5], pełna funkcjonalność aplikacji jest dostępna tylko dla użytkowników, którzy zdecydują się na wykupienie wyższych wersji abonamentu. Za barierą płatności zablokowane są, chociażby możliwość dowolnego formatowania zapisywanego tekstu, więcej motywów aplikacji czy, co najważniejsze, bardziej rozbudowane możliwości przechowywania i synchronizacji danych.

Ostatecznie aplikacja wydaje się chcieć robić zbyt wiele rzeczy na raz. Skupia się na zbyt dużej ilości oferowanych funkcjonalności, przez co przy pierwszym kontakcie odrzuca,

zamiast zachęcać do jej wypróbowania. Dodatkowo odrzucający jest także tak silny nacisk położony na mikropłatności.



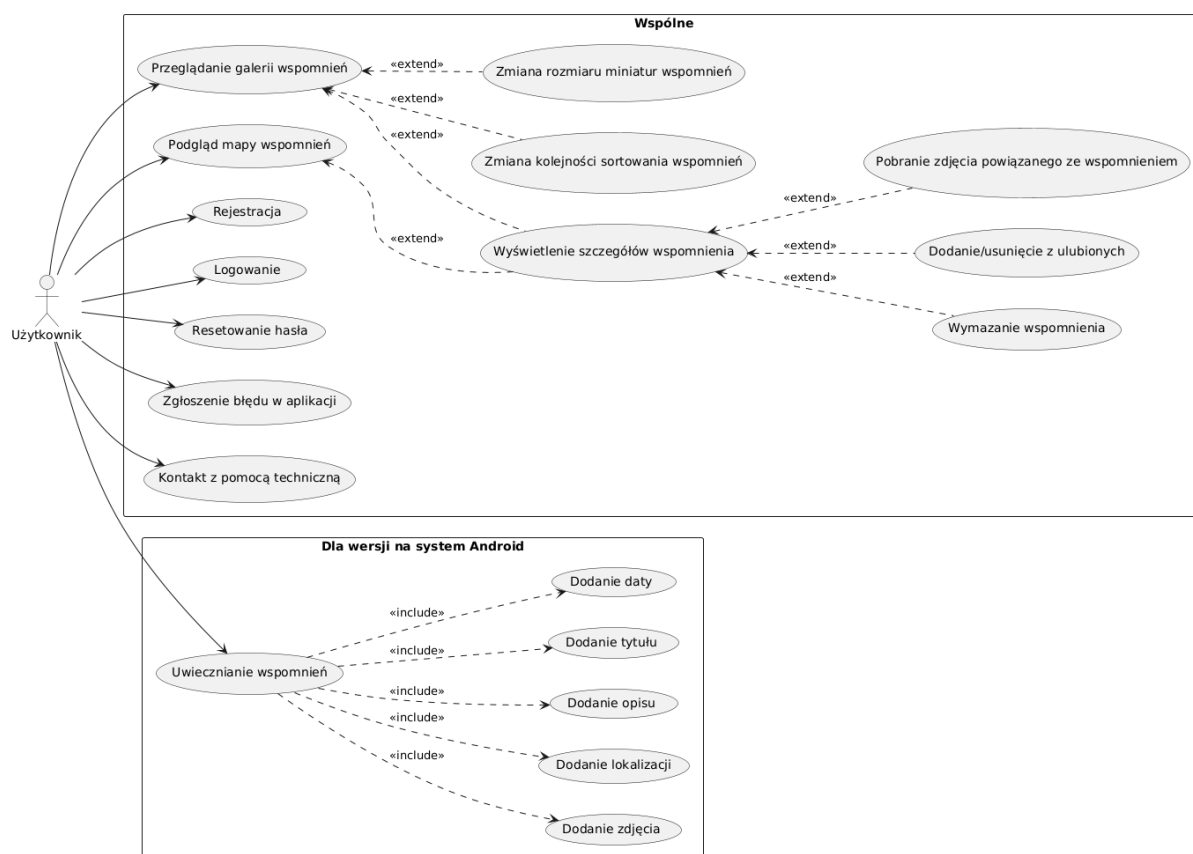
Rysunek 2
Journey: Diary, Journal, Notes w wersji na system Android

Rozdział 2

Analiza wymagań

2.1 Przypadki użycia

Rysunek 3 przedstawia utworzony diagram przypadków użycia.



Rysunek 3
Diagram przypadków użycia

2.2 Wymagania funkcjonalne

Poniższa lista przedstawia wymagania funkcjonalne, które definiują jakie funkcjonalności powinna zawierać tworzona aplikacja.

1. Dostęp do głównych funkcji aplikacji jedynie po utworzeniu konta i zalogowaniu się na nie.
2. Utworzenie konta powinno odbywać się w oparciu o wykorzystanie adresu e-mail (z jednym adresem e-mail może być powiązany tylko jeden użytkownik), nazwy użytkownika i hasła (co najmniej 8 znaków).

3. Logowanie do aplikacji powinno odbywać się w oparciu o wykorzystanie adresu e-mail oraz hasła.
4. Użytkownik powinien mieć możliwość zmiany swojego hasła do aplikacji.
5. Użytkownik powinien mieć dostęp do zapisanych wspomnień z różnych urządzeń, dzięki zapisywaniu danych w chmurze.
6. Użytkownik powinien mieć możliwość zmiany kolejności sortowania wspomnień w galerii (od najnowszych lub od najstarszych).
7. Użytkownik powinien mieć możliwość zmiany rozmiaru kafelków wspomnień w galerii (małe lub duże).
8. Użytkownik powinien mieć możliwość dodania lub usunięcia danego wspomnienia z ulubionych.
9. Użytkownik powinien mieć możliwość wymazania zapisanego przez siebie wspomnienia.
10. Użytkownik powinien mieć możliwość pobrania zdjęcia powiązanego ze wspomnieniem na swoje urządzenie.
11. Użytkownik powinien mieć możliwość wyświetlenia mapy zapisanych przez siebie wspomnień.
12. W wersji aplikacji na system Android możliwość zapisania nowego wspomnienia poprzez dodanie:
 - a. **Tytułu wspomnienia**
 - b. **Opisu wspomnienia**
 - c. **Daty wspomnienia** – próba odczytania informacji z danych **EXIF** zdjęcia, dodatkowo opcja ręcznego wyboru daty, z użyciem odpowiedniego okna dialogowego.
 - d. **Lokalizacji** – wybór miejsca na mapie lub pobranie aktualnej lokalizacji użytkownika
 - e. **Zdjęcia** – wykonanie nowego lub wybór zdjęcia z pamięci urządzenia
13. Użytkownik powinien mieć możliwość kontaktu z pomocą techniczną.
14. Użytkownik powinien mieć możliwość zgłoszenia napotkanego w aplikacji błędu poprzez odpowiedni formularz.
15. Motyw aplikacji powinien dostosowywać się do motywu systemu/przeglądarki.
16. Orientacja wersji aplikacji na system Android powinna zostać zablokowana w trybie portretowym.

2.3 Wymagania niefunkcjonalne

Poniższa lista prezentuje wymagania niefunkcjonalne, które definiują warunki, przy których aplikacja ma działać prawidłowo oraz określają wytyczne związane z jej jakością.

1. Prostota obsługi – interfejs aplikacji powinien być czytelny i responsywny.
2. Spójność doświadczenia – aplikacja powinna wyglądać i działać w podobny sposób zarówno w wersji na system Android jak i w wersji webowej.
3. Połączenie z Internetem wymagane do prawidłowego działania wersji webowej oraz dostępu do pełnej funkcjonalności wersji na system Android.

4. Przy dodawaniu wspomnienia, zezwolenie na dostęp do lokalizacji i włączenie usług lokalizacyjnych urządzenia są niezbędne do uzyskania aktualnej lokalizacji użytkownika.
5. Przy dodawaniu wspomnienia, aby wykonać nowe zdjęcie, niezbędne jest zezwolenie na dostęp aplikacji do kamery.
6. W wersji na system Android działanie na urządzeniach o wersji systemu co najmniej **Android 10**.
7. W wersji webowej działanie w przeglądarkach internetowych:
 - **Microsoft Edge** – wersje od 131.0.2903.70
 - **Opera** – wersje od 115.0.5322.77
 - **Mozilla Firefox** – wersje od 133.0
 - **Google Chrome** – wersje od 131.0.6778.134
8. Skalowalność – przygotowanie aplikacji w sposób umożliwiający jej dalszy rozwój i dodawanie nowych funkcjonalności.
9. Niezawodność – stworzenie aplikacji dopracowanej pod kątem występowania błędów oraz wprowadzenie mechanizmów mających na celu wykrywanie problemów w przyszłości.

Rozdział 3

Fundamenty Projektu

3.1 Wykorzystane technologie i narzędzia

3.1.1 Visual Studio Code

Visual Studio Code [6] jest edytorem kodu stworzonym przez firmę **Microsoft**. Oferuje szerokie możliwości dostosowania go do swoich potrzeb, umożliwiając instalację przydatnych rozszerzeń czy motywów. Obsługuje przy tym wiele języków programowania, stawiając na szybkość działania i wspieranie programisty narzędziami takimi jak wbudowany debugger czy podświetlanie i podpowiadanie składni. W czasie tworzenia aplikacji będącej przedmiotem pracy stanowił podstawowe narzędzie do pisania i testowania kodu.

3.1.2 Android Studio

Android Studio [7] jest oficjalnym **IDE** (zintegrowanym środowiskiem programistycznym) wykorzystywanym do tworzenia aplikacji na system Android. Umożliwia tworzenie ich z wykorzystaniem języka **Java** lub **Kotlin** (język preferowany). Dodatkowo oferuje opcję tworzenia emulatorów urządzeń z systemem Android, co pozwala na testowanie aplikacji bez dostępu do fizycznego urządzenia. IDE to jest wykorzystywane także w trakcie tworzenia aplikacji na system Android z użyciem Flutter. Stanowi źródło komponentów [8] niezbędnych do prawidłowego utworzenia i działania aplikacji. Przy tworzeniu opisywanej aplikacji wykorzystano Android Studio w wersji **Koala Feature Drop | 2024.1.2**. Nowsza wersja (**Ladybug | 2024.2.1**) potrafiła powodować problemy z prawidłowym kompilowaniem się aplikacji, dlatego podjęta została decyzja o pozostaniu przy starszej wersji oprogramowania.

3.1.3 Git

Git [9] to darmowy system kontroli wersji. Ułatwia śledzenie i wprowadzanie zmian w rozwijanym projekcie. Daje przy tym możliwość powrotu do jego wcześniejszej wersji, w sytuacji, w której wprowadzone przez twórcę zmiany mogą nie działać w sposób prawidłowy.

3.1.4 Google Maps Platform

Google Maps Platform [10] to zbiór narzędzi i interfejsów programistycznych, które umożliwiają korzystanie z usług **Map Google** w tworzonej aplikacji.

Do stworzenia aplikacji będącej przedmiotem pracy wykorzystano:

1. **Maps SDK for Android** [11] – umożliwia wyświetlanie i interakcję z mapą w aplikacji na system Android.
2. **Maps JavaScript API** [12] – umożliwia wyświetlanie i interakcję z mapą z użyciem strony internetowej.
3. **Maps Static API** [13] – umożliwia wygenerowanie statycznego obrazu mapy na podstawie podanych parametrów, takich jak współrzędne geograficzne czy poziom przybliżenia.
4. **Geocoding API** [14] – umożliwia przede wszystkim wzajemną konwersję między współrzędnymi geograficznymi i adresami.

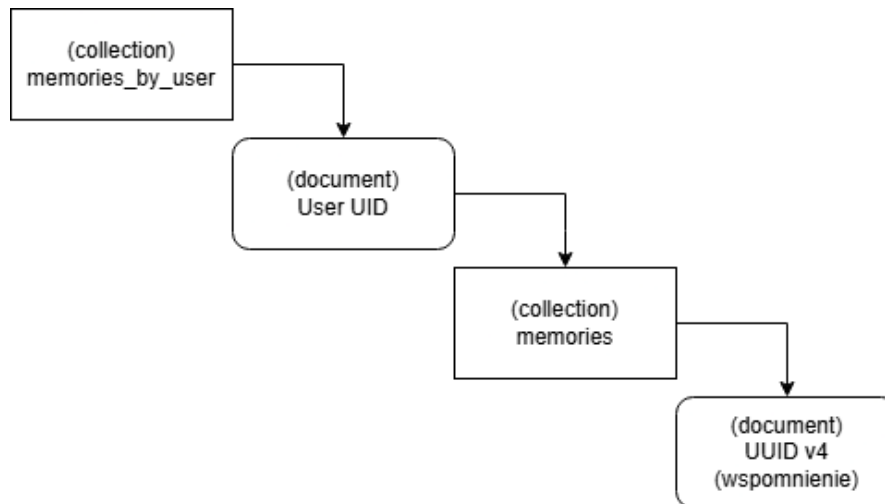
3.1.5 Firebase

Firebase [15] jest chmurową platformą firmy **Google**, która ułatwia tworzenie aplikacji mobilnych i webowych. Oferuje szeroki zakres dostępnych usług, obejmujący elementy takie jak baza danych, autoryzacja użytkowników czy analizowanie napotkanych przez aplikację błędów.

Do stworzenia aplikacji będącej przedmiotem pracy wykorzystano:

1. **Cloud Firestore** [16] – baza danych **NoSQL**, która umożliwia synchronizację danych w czasie rzeczywistym. Oferuje wsparcie trybu offline (przechowywanie danych lokalnie oraz synchronizacja zmian po połączeniu z Internetem) dla obsługiwanej aplikacji. Cloud Firestore opiera się na koncepcji kolekcji i dokumentów, które tworzą razem swego rodzaju hierarchiczną strukturę.

Rysunek 4 przedstawia strukturę bazy danych Cloud Firestore dla aplikacji będącej przedmiotem pracy.



Rysunek 4
Struktura bazy danych Cloud Firestore dla opisywanej aplikacji

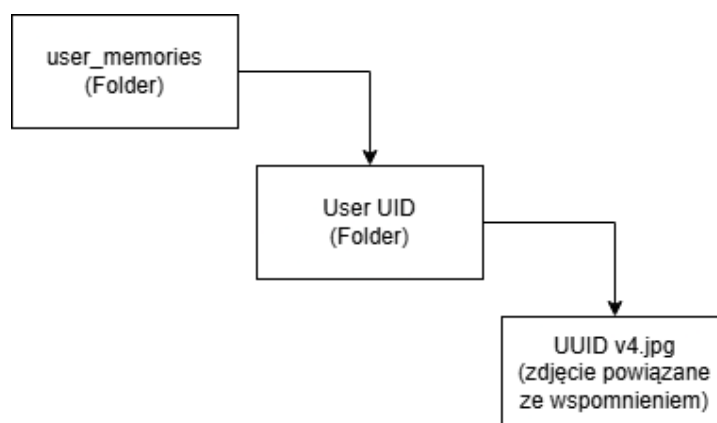
Rysunek 5 przedstawia strukturę dokumentu reprezentującego wspomnienie w Cloud Firestore. W celu zapewnienia jej unikalności, nazwa dokumentu jest generowana przez aplikację **Re(me)mber** podczas zapisywania wspomnienia, wykorzystując do tego **UUID version 4** [17]. Warto zauważyć, że pola **username**, **userId** oraz **email** nie są obecnie używane, ale mogą zostać wykorzystane do rozwoju aplikacji w przyszłości.

UUID v4
address (string)
description (string)
email (string)
geopoint (geopoint)
imageUrl (string)
isFavourite (boolean)
memoryDate (string)
title (string)
uploadTimeStamp (timestamp)
userId (string)
username (string)

Rysunek 5
Struktura dokumentu reprezentującego wspomnienie w Cloud Firestore

2. **Cloud Storage for Firebase** [18] – usługa pozwalająca na przechowywanie i zarządzanie plikami w chmurze. W aplikacji będącej przedmiotem pracy wykorzystano ją do przechowywania zdjęć powiązanych z zapisanymi wspomnieniami. Nazwa każdego zdjęcia odpowiada nazwie dokumentu w Cloud Firestore, który reprezentuje powiązane z tym zdjęciem wspomnienie.

Rysunek 6 przedstawia strukturę plików w Cloud Storage dla opisywanej aplikacji.



Rysunek 6

Struktura plików w Cloud Storage dla opisywanej aplikacji

3. **Firestore Authentication** [19] – zestaw usług i narzędzi umożliwiających uwierzytelnianie oraz obsługę użytkowników aplikacji. W zależności od konfiguracji użytkownik może logować się anonimowo, z wykorzystaniem adresu e-mail i hasła czy z użyciem zewnętrznych usług, takich jak konto **Google** lub **Facebook**. Na potrzeby aplikacji będącej przedmiotem pracy wybrano opcję opierającą się na użyciu adresu e-mail oraz hasła. Firestore Authentication umożliwia także konfigurację automatycznego wysyłania wiadomości e-mail, które służą do resetowania hasła czy potwierdzania rejestracji.
4. **Firestore Crashlytics** [20] – narzędzie umożliwiające nadzorowanie awarii aplikacji powiązanych z jej niespodziewanym działaniem czy nieobsłużonymi błędami. Udostępnia dokładne informacje o napotkanym problemie, które pozwalają na dogłębne poznanie jego charakterystyki. Niestety narzędzie to nie obsługuje błędów napotkanych w webowej wersji aplikacji.

3.1.6 Google Forms

Google Forms [21] to darmowe narzędzie umożliwiające tworzenie i udostępnianie internetowych ankiet, formularzy, quizów czy testów. W omawianej aplikacji umożliwia użytkownikom zgłaszanie napotkanych błędów oraz jest integralną częścią systemu ich automatycznego wykrywania i raportowania.

3.1.7 Gmail

Gmail [22] to darmowa usługa poczty elektronicznej firmy Google. Umożliwia wysyłanie, odbieranie i zarządzanie wiadomościami e-mail. Na potrzeby aplikacji **Re(me)mber** został utworzony i odpowiednio skonfigurowany specjalny użytkownik omawianej usługi. Umożliwiło to integrację z Firestore Authentication, dzięki czemu możliwe jest zautomatyzowane wysyłanie wiadomości e-mail związanych z resetowaniem hasła czy potwierdzeniem utworzenia konta.

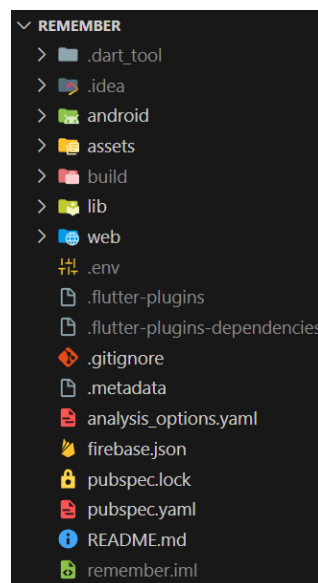
3.1.8 Flutter SDK

Flutter to technologia umożliwiająca tworzenie natywnie kompilowanych, multiplatformowych aplikacji z wykorzystaniem pojedynczej bazy kodu. Aplikacje te mogą działać na systemach **Android, iOS, Windows, Linux, macOS** oraz **w wersji webowej**. W skład **Flutter SDK** [23] wchodzi dedykowany framework oraz bogaty zestaw zasobów i narzędzi umożliwiających tworzenie aplikacji. Twórcą tej technologii jest **Google**, jednak dzięki swojej otwartoźródłowej naturze jest nieustannie rozwijana również przez skupioną wokół niej, ciągle rosnącą społeczność. Do stworzenia aplikacji wykorzystano Flutter SDK w wersji **3.24.5**. Dokładniejsze omówienie technologii Flutter zawarto w **Rozdziale 4**.

3.2 Struktura plików w projekcie

3.2.1 Ogólna struktura plików

Rysunek 7 przedstawia ogólną strukturę plików w projekcie.



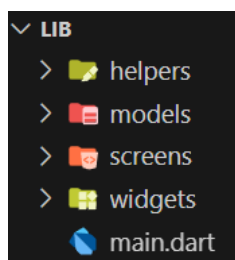
Rysunek 7
Ogólna struktura plików w projekcie

Najważniejsze elementy to:

1. **/android** – katalog zawierający pliki powiązane z platformą Android, takie jak pliki konfiguracyjne **gradle** czy **AndroidManifest.xml**.
2. **/assets** – katalog zawierający zewnętrzne zasoby wykorzystywane przez aplikację (np. obrazy).
3. **/build** – katalog zawierający pliki wynikowe powstałe jako rezultat kompilacji aplikacji na poszczególne platformy.

4. **/web** – katalog zawierający pliki konfiguracyjne powiązane z platformą webową, takie jak **index.html** czy **favicon.png**.
5. **pubspec.yaml** – plik konfiguracyjny Flutter. Z jego poziomu następuje zarządzanie nazwą projektu, zasobami z katalogu **assets** czy zewnętrznymi pakietami.
6. **/lib** – katalog zawierający główną część aplikacji w postaci plików **.dart**.

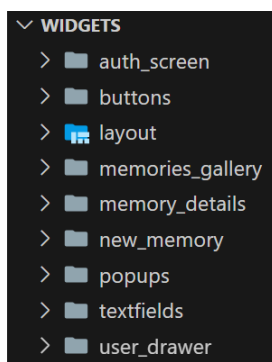
3.2.2 Struktura katalogu lib



Rysunek 8
Struktura katalogu lib

Dla lepszej czytelności i organizacji plików źródłowych projektu katalog **lib** podzielono (zgodnie ze strukturą widoczną na **Rysunku 8**) na:

1. **main.dart** – plik wejściowy aplikacji, który zawiera funkcję **main** oraz elementy związane z inicjalizacją aplikacji.
2. **/helpers** – katalog zawierający elementy pomocnicze dla aplikacji, takie jak wielokrotnie wykorzystywane funkcje, ustawienia motywów czy stałe i łańcuchy znakowe.
3. **/models** – katalog przechowujący zdefiniowane modele danych. Zawiera model wykorzystywany przy dodawaniu lokalizacji do wspomnienia.
4. **/screens** – katalog zawierający wykorzystywane w aplikacji ekrany. Część z nich posiada stosunkowo prostą konstrukcję, pełniąc przy tym rolę kontenerów, które opierają się na widgetach zdefiniowanych w innych plikach.
5. **/widgets** – katalog zawierający bardziej rozbudowane widgety oraz te, które są wielokrotnie wykorzystywane w aplikacji. Pogrupowano je względem kategorii, do której należą. Strukturę omawianego katalogu przedstawia **Rysunek 9**.



Rysunek 9
Struktura katalogu widgets

Rozdział 4

Flutter

4.1 Tworzenie aplikacji z wykorzystaniem Flutter

Jak opisano w poprzednim rozdziale, Flutter wykorzystuje się do tworzenia multiplatformowych aplikacji z użyciem pojedynczej bazy kodu. Podstawę tworzenia aplikacji stanowi język **Dart** [24]. Jest to obiektowy język programowania, udostępniony (podobnie jak Flutter) przez **Google**. W trakcie procesu deweloperskiego aplikacji, które nie są przeznaczone na platformę webową, Flutter wykorzystuje kompilację **JIT (Just in Time)** oraz specjalną maszynę wirtualną. Pozwala to na wygodne i dynamiczne testowanie zmian w kodzie. Można do tego wykorzystać mechanizm **hot reload**, który umożliwia odświeżenie aplikacji przy zachowaniu jej stanu lub mechanizm **hot restart**, który umożliwia całkowity restart aplikacji. Chcąc jednak skompilować aplikację do wersji produkcyjnej, wykorzystywana jest kompilacja **AOT (Ahead of Time)**, której wynikiem jest natywny kod maszynowy [25]. Jak wspomniano wcześniej, proces kompilacji jest inny [26] dla webowej wersji aplikacji. Dla procesu deweloperskiego, zamiast maszyny wirtualnej, wykorzystywany jest kompilator **dartdevc**, który obsługuje **hot restart**. Przy kompilacji wersji produkcyjnej wykorzystywany jest kompilator **dart2js**, który generuje zoptymalizowany kod **JavaScript**.

4.2 Widżety i ich stan

Podstawowym elementem tworzenia interfejsu z wykorzystaniem Flutter są **widżety** [27]. To wszystkie elementy widziane na ekranie aplikacji. Są one w sobie odpowiednio zagnieżdżane, tworząc hierarchiczną strukturę – tzw. **drzewo widżetów**. Ważnym pojęciem w kontekście widżetów jest także ich **stan (state)**. Są to informacje, na podstawie których tworzony jest widżet, a które mogą zmieniać się w czasie jego istnienia (np. reagując na akcje użytkownika). Przykładem stanu mogą być zmienne typu **bool**, które wpływają na to, jaka będzie zawartość danego widżetu.

W tym kontekście wyróżnia się dwa podstawowe rodzaje widżetów :

1. **StatelessWidget** – widżet, który nie posiada stanu. Jest zależny jedynie od swojej konfiguracji początkowej.
2. **StatefulWidget** – widżet, który posiada stan. Każda zmiana stanu przy użyciu metody **setState** wywołuje metodę **build** widżetu, która powoduje jego przebudowę w oparciu o nowy stan.

Flutter domyślnie [28] wykorzystuje **Material Design 3** [29]. Jest to zbiór wytycznych **Google** dotyczących tworzenia interfejsów użytkownika. Takie rozwiązanie zapewnia spójność doświadczenia podczas korzystania z aplikacji na różnych urządzeniach. Warto

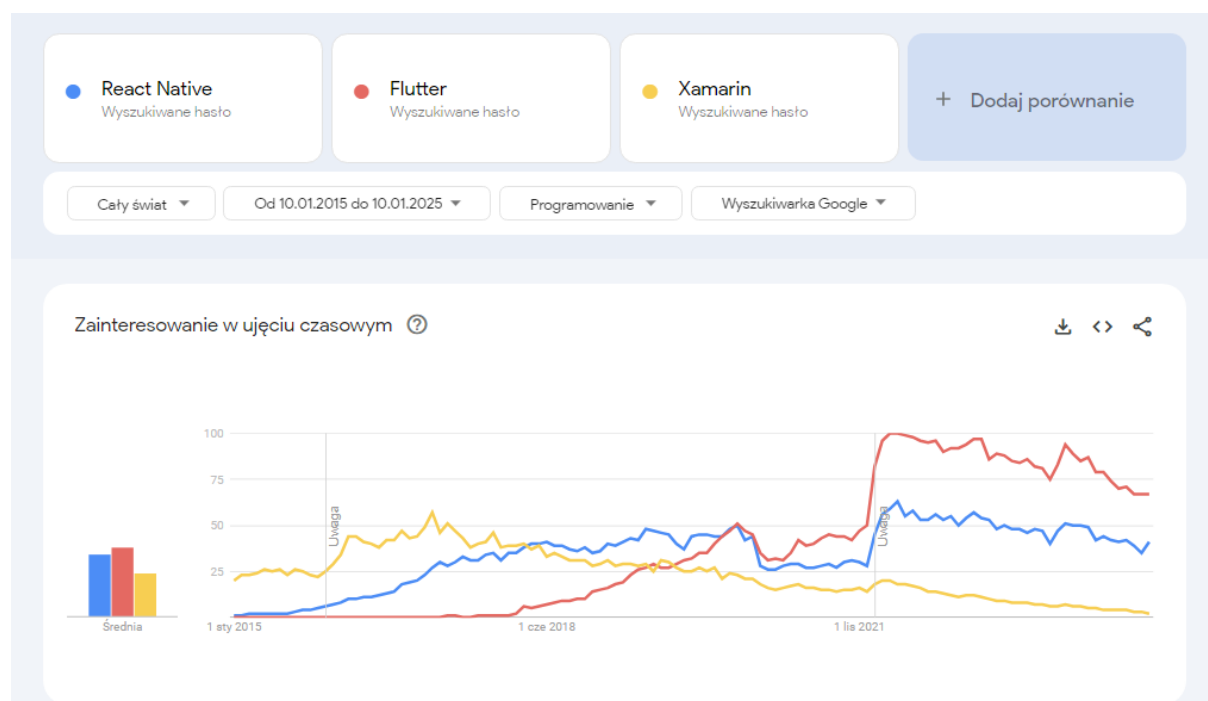
jednak zauważyć, że opcjonalnie można skorzystać z **Cupertino widgets** [30], a więc widgetów przygotowanych zgodnie z wytycznymi **Apple** dotyczącymi interfejsu lub stworzyć własne widżety.

4.3 Platforma pub.dev

Wspomniana otwartoźródłowa natura technologii Flutter oraz Dart jest wyraźnie widoczna w platformie **pub.dev** [31]. To oficjalne repozytorium pakietów dla Dart oraz Flutter, gromadzące tysiące pomocnych narzędzi i bibliotek, które znacznie poszerzają możliwości tworzonych aplikacji i ułatwiają ich rozwój. Wszystkie pakiety z platformy pub.dev, dodane do aktualnie tworzonego projektu, można znaleźć we wspomnianym w **Rozdziale 3.2.1** pliku **pubspec.yaml**.

4.4 Popularność

Główną konkurencją dla Flutter stanowią oparty na **JavaScript** – **React Native** i oparty na **C#** – **Xamarin**. Wykorzystując informacje z **Google Trends** [32] (**Rysunek 10**), łatwo zauważyć, że to właśnie Flutter, choć korzysta ze znacznie mniej popularnego języka, jakim jest Dart, cieszy się (według statystyk) największym zainteresowaniem. Od premiery wersji **1.0** w 2018 roku zainteresowanie omawianą technologią (pomimo niewielkich spadków) sukcesywnie rośnie.



Rysunek 10

Porównanie zainteresowania wybranymi narzędziami do tworzenia aplikacji multiplatformowych w Google Trends

Wedle informacji przedstawionych na oficjalnej stronie Flutter [33], technologia ta jest szeroko wykorzystywana przez Google, a także przez inne znane firmy, takie jak **Xiaomi**

czy **Wolt**. Zgodnie doceniane jest przyspieszenie procesu deweloperskiego dzięki wykorzystaniu pojedynczej bazy kodu, a tym samym większa spójność w działaniu aplikacji na różnych platformach.

4.5 Ograniczenia Flutter dla aplikacji webowych

Pomimo swoich licznych zalet, Flutter cechuje kilka potencjalnie problematycznych kwestii. Jedną z najważniejszych jest niedojrzałość webowej strony tej technologii. Różnica w ilości między pakietami z platformy pub.dev dostępnymi na system Android, a tymi dostępnymi jednocześnie na system Android i platformę webową wynosi kilkanaście tysięcy. Oferowane przez Flutter narzędzia debugowania w wersji webowej również są bardziej ograniczone niż te oferowane przykładowo dla aplikacji na system Android, a dodatkowo nie jest obsługiwany mechanizm **hot reload**. Innym poważnym problemem, adresowanym przez samych twórców Flutter [34], są problemy z pozycjonowaniem strony. Zapowiadają przy tym prace nad poprawieniem tej sytuacji [35].

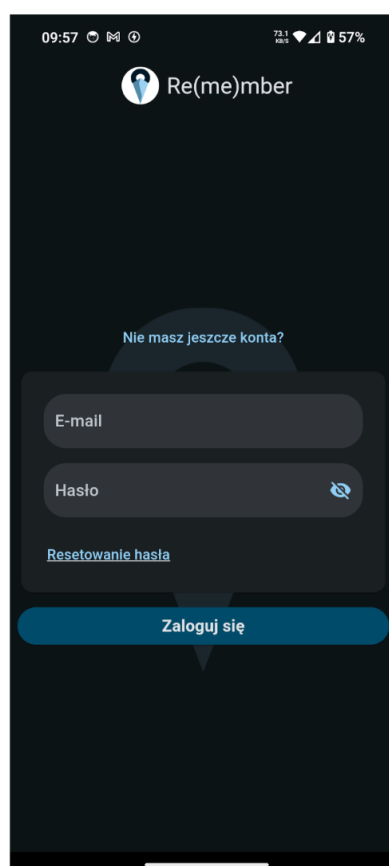
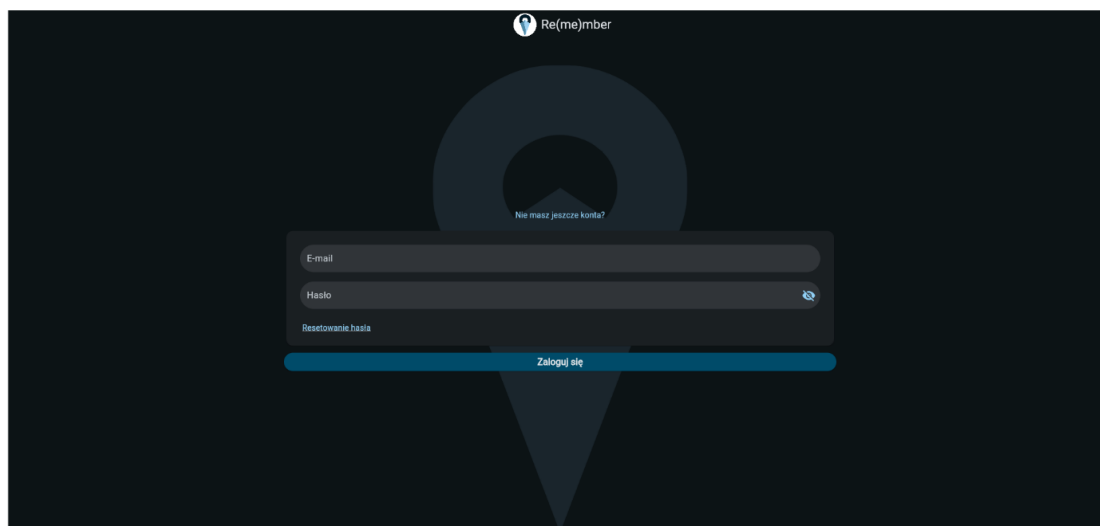
Rozdział 5

Użytkownik i interfejs

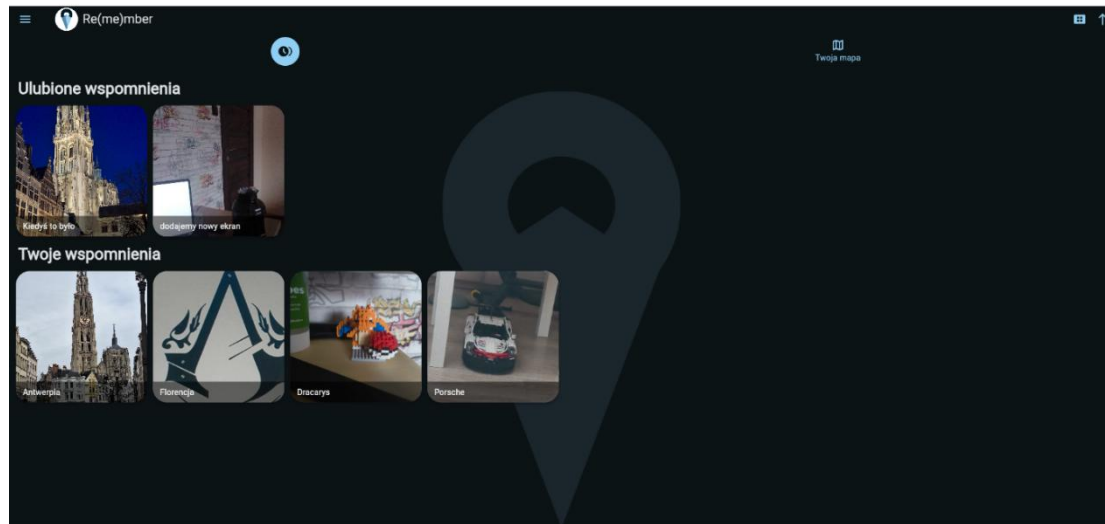
5.1 Idea tworzenia interfejsu

Tworząc interfejs aplikacji, należy pamiętać, że jest on projektowany przede wszystkim z myślą o użytkowniku końcowym. W związku z tym duży nacisk położono na czytelność, intuicyjność i responsywność. Podjęty został również wysiłek mający na celu ujednolicenie doświadczenia użytkownika korzystającego zarówno z wersji aplikacji na system Android, jak i jej webowego odpowiednika. Miało to na celu ograniczenie do minimum dysonansu wynikającego z obcowania z aplikacją na różnych platformach. Największą różnicą jest pasek nawigacyjny. W wersji na system Android znajduje się on na dole ekranu, oferując trzy opcje do wyboru, a w wersji webowej – na górze, oferując dwie opcje. Tam, gdzie było to potrzebne (np. podczas pobierania aktualnej lokalizacji użytkownika), zastosowano odpowiednie indykatory, które mają wskazywać na odbywający się proces ładowania. Dodatkowo wprowadzono odpowiednie mechanizmy i rozwiązania dbające o to, by interfejs dostosowywał się do wymiarów ekranu, na którym jest wyświetlany. Proces ten odbywa się przy zachowaniu czytelności i użyteczności interfejsu.

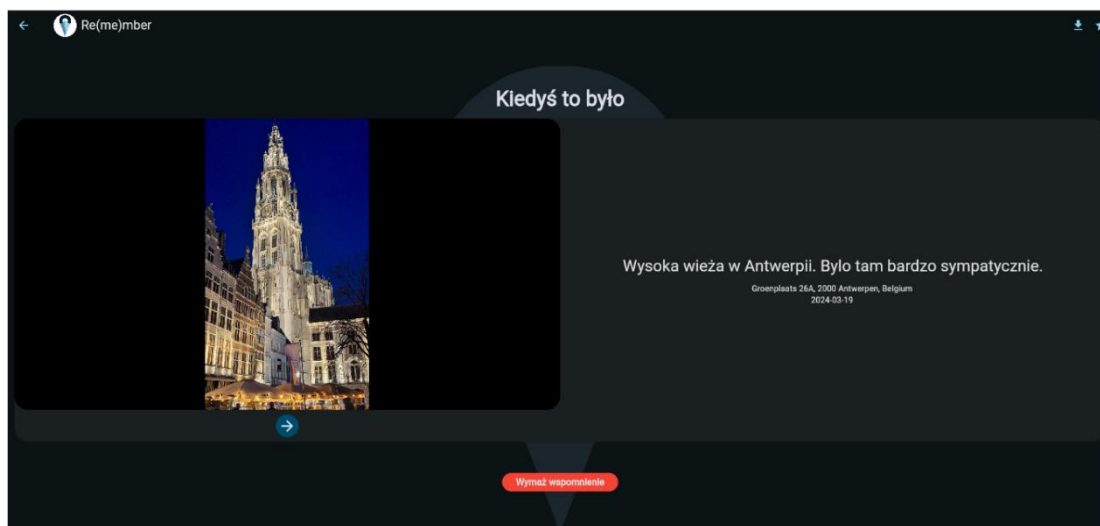
Rysunek 11, Rysunek 12 i Rysunek 13 przedstawiają przykładowe porównania interfejsu pomiędzy aplikacją w wersji webowej i na system Android.



Rysunek 11
Porównanie ekranu logowania w wersji webowej (na górze) i na system Android (na dole)



Rysunek 12
 Porównanie ekranu galerii wspomnień w wersji webowej (na górze) i na system Android (na dole)



Rysunek 13

Porównanie ekranu szczegółów wspomnienia w wersji webowej (na górze) i na system Android (na dole)

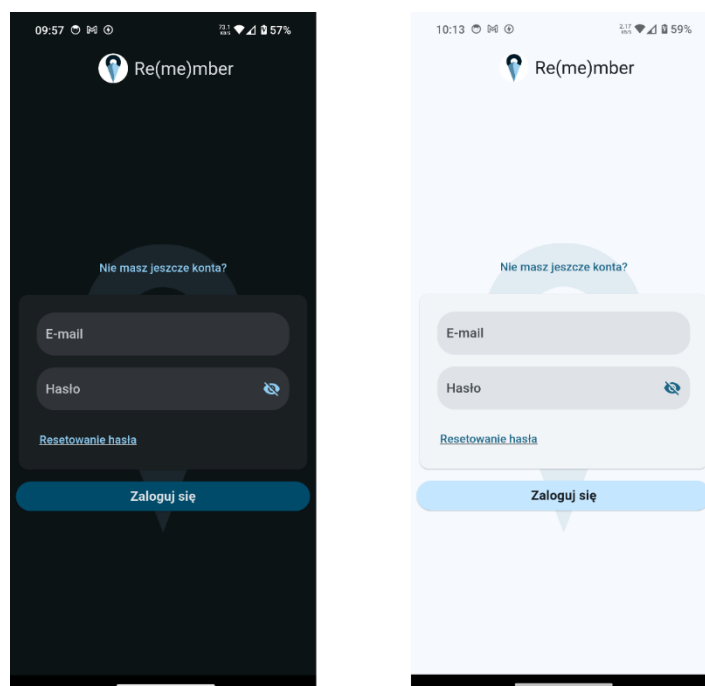
Przedstawiony na **Rysunku 13** ekran szczegółów wspomnienia jest ekranem, na którym domyślne różnice w interfejsie pomiędzy wersją aplikacji na system Android i wersją webową są najbardziej widoczne. Pozostałe ekrany są w zasadzie bliźniacze w obu przypadkach. Warto zauważyć, że zaimplementowane mechanizmy skalowania aplikacji mogą sprawić, że dla pewnych wymiarów okna przeglądarki internetowej, nawet na ekranie szczegółów wspomnienia, uzyskany zostanie układ taki jak w przypadku wersji na system Android.

Warto również zauważyć, że dla sytuacji, w których nie jest możliwe wyświetlenie całej zawartości danego ekranu jednocześnie, zastosowano mechanizm przewijania tej zawartości.

5.2 Motywy i schematy kolorystyczne

Aplikacja zawiera dwa podstawowe motywy – jasny i ciemny. To, który z nich jest aktualnie wyświetlany dostosowuje się do ustawień systemowych/ustawień przeglądarki.

Rysunek 14 przedstawia zestawienie motywów aplikacji na przykładzie ekranu logowania. Dla uproszczenia porównanie to dotyczyć będzie wersji aplikacji na system Android.



Rysunek 14

Zestawienie ciemnego (po lewej) i jasnego (po prawej) motywu aplikacji na przykładzie ekranu logowania w wersji na system Android

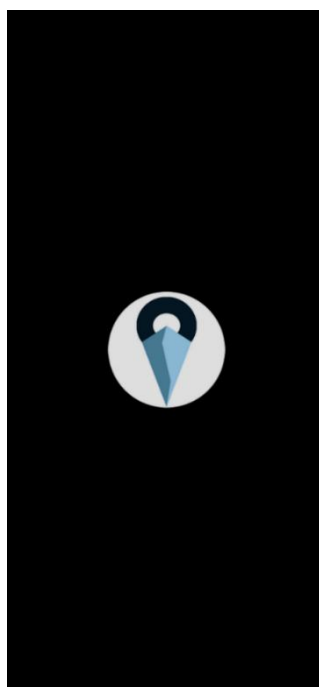
Schematy kolorystyczne zostały stworzone wykorzystując narzędzie **Material Theme Builder** [36]. Na podstawie wybranych kolorów wejściowych generuje ono odpowiedni schemat, który jest zgodny z regułami Material Design 3. Powstałe schematy zostały odpowiednio zaimportowane, a następnie wykorzystane do stworzenia właściwych motywów dopasowanych do potrzeb i założeń projektowych aplikacji. Starano się, aby tworzone motywy były możliwie zgodne z Material Design 3.

Rozdział 6

Struktura aplikacji

6.1 Ekran ładowania

Rysunek 15 przedstawia ekran ładowania, który pojawia się przy uruchomieniu wersji aplikacji na system Android. Zawiera on jedynie logo samej aplikacji i jest widoczny w czasie jej wstępnego ładowania.



Rysunek 15
Ekran ładowania aplikacji

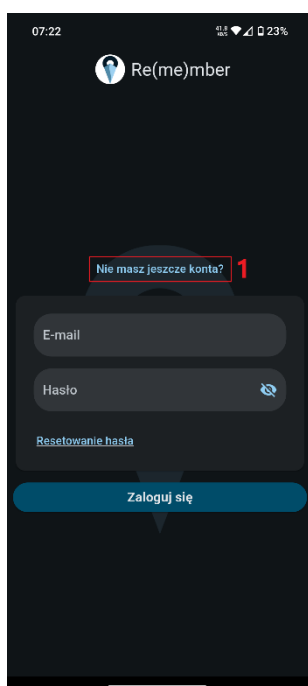
6.2 Ekran Startowy

Ekran wyświetlany po załadowaniu aplikacji jest zależny od tego, czy użytkownik jest do niej aktualnie zalogowany. Jeżeli tak nie jest, to wyświetlany jest ekran logowania/rejestracji, a gdy jest – ekran zawartości. Wprowadzono mechanizm, dzięki któremu przy każdym uruchomieniu aplikacji przez zalogowanego użytkownika następuje próba odświeżenia jego **ID token** [37][38]. Jeżeli próba ta się nie powiedzie, to użytkownik jest wylogowywany i przenoszony na ekran logowania/rejestracji. Warto zauważyć, że próby nieudane przez brak połączenia z Internetem nie wpływają na status zalogowania użytkownika. Co więcej, na krótko przed wygaśnięciem **ID token**, Firebase podejmuje analogiczne próby jego odświeżenia.

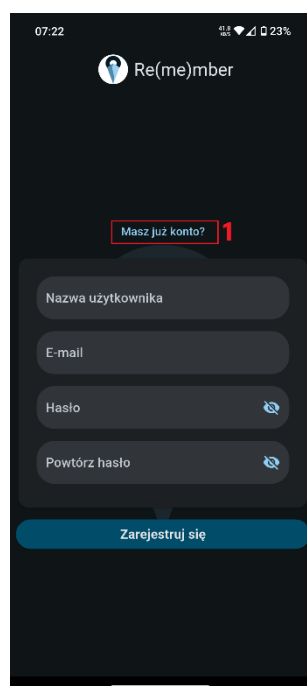
6.3 Ekran logowania/rejestracji

Widoczny na **Rysunku 16** ekran logowania i widoczny na **Rysunku 17** ekran rejestracji mogą być dynamicznie przełączane z użyciem przycisku tekstowego oznaczonego cyfrą **1** (kolorem czerwonym).

Ekran te zawierają przede wszystkim formularze umożliwiające użytkownikom zalogowanie się na swoje konto lub utworzenie nowego. Proces ten odbywa się z wykorzystaniem Firebase Authentication i wymaga połączenia z Internetem. W przypadku wystąpienia błędów powiązanych z tymi procesami użytkownik jest o nich informowany z użyciem stosownych okien dialogowych. Dodatkowo każde z pól formularzy jest odpowiednio opisane i walidowane. Informacyjne okna dialogowe omówiono w **Rozdziale 6.7.1**, a pola tekstowe w **Rozdziale 6.7.2**. Pomyślne utworzenie konta powoduje wysłanie do użytkownika wiadomości e-mail, która potwierdza rejestrację.

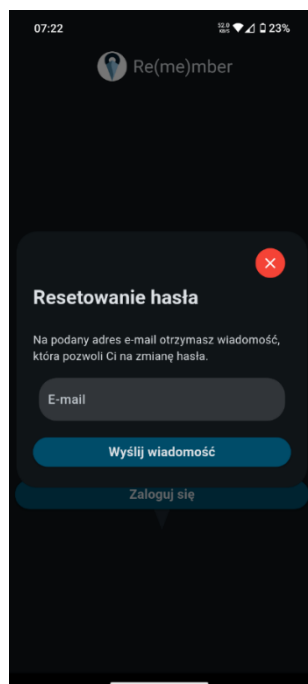


Rysunek 16
Ekran logowania



Rysunek 17
Ekran rejestracji

Z poziomu ekranu logowania użytkownik może także zresetować swoje hasło. Wykorzystywany do tego jest przycisk tekstowy „**Resetowanie hasła**”. Po jego użyciu pojawia się okno resetowania hasła, przedstawione na **Rysunku 18**. Resetowanie hasła z wykorzystaniem tego okna dialogowego odbywa się w oparciu o wprowadzenie powiązanego z kontem adresu e-mail. Po zatwierdzeniu użytkownik otrzymuje wiadomość e-mail, która umożliwia ustawienie nowego hasła. Wymagane jest połączenie z Internetem.

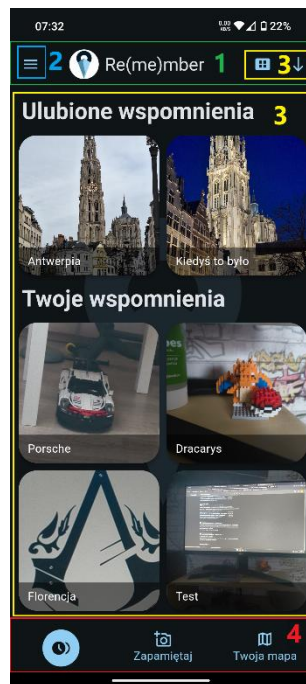


Rysunek 18
Okno resetowania hasła

Warto nadmienić, że Firebase Authentication domyślnie stosuje zabezpieczenie przed **email enumeration** [39]. W związku z tym, bez względu na to, czy z podanym adresem e-mail jest powiązane konto, czy też nie, użytkownik i tak zobaczy odpowiedni panel informacyjny (opisany w **Rozdziale 6.7.3**).

6.4 Ekran zawartości

Każde uruchomienie aplikacji bez dostępu do Internetu, a związane z wyświetleniem omawianego ekranu, powoduje wystąpienie stosownego okna dialogowego. Informuje ono użytkownika o korzystaniu z aplikacji w trybie offline, a tym samym o fakcie, iż część z jej funkcjonalności może nie działać prawidłowo.



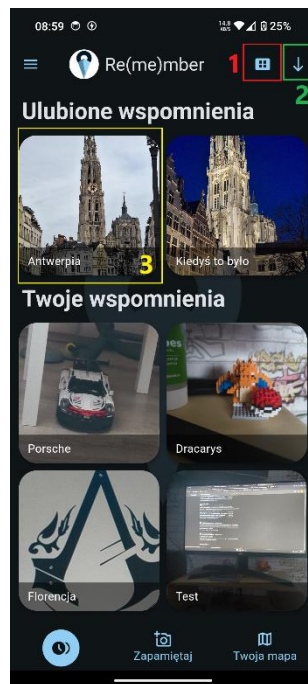
Rysunek 19
Struktura ekranu zawartości

Zgodnie z przedstawioną na **Rysunku 19** strukturą ekranu zawartości:

1. **Pasek aplikacji** (zielone zaznaczenie) – domyślnie zawiera przycisk szuflady użytkownika (opisany w **punkcie 2**), a także ikonę aplikacji i jej nazwę. Zależnie od wybranej zawartości ekranu może zawierać także dodatkowe przyciski.
2. **Przycisk szuflady użytkownika** (niebieskie zaznaczenie) – otwiera szufladę użytkownika (opisana w **Rozdziale 6.4.4**).
3. **Właściwa zawartość ekranu** (żółte zaznaczenie) – jest zależna od pozycji wybranej w pasku nawigacyjnym, opisanym w **punkcie 4**.
4. **Pasek nawigacyjny** (czerwone zaznaczenie) – umożliwia nawigację pomiędzy różnymi sekcjami aplikacji za pomocą odpowiednich, zawierających ikony, przycisków. Domyślnie wybrana jest pierwsza pozycja – **Wspomnienia**. Pozycja **Zapamiętaj** umożliwia dodanie nowego wspomnienia (wersja na system Android). Ostatnia pozycja – **Twoja mapa** – pozwala na wyświetlenie mapy zapisanych wspomnień.

6.4.1 Ekran galerii wspomnień

Ekran galerii wspomnień to domyślna konfiguracja ekranu zawartości. Zawiera zapisane przez użytkownika wspomnienia, pogrupowane względem tego, czy są one tymi ulubionymi. Dane są synchronizowane na bieżąco z Cloud Firestore. Co więcej, użytkownicy wersji aplikacji na system Android posiadają dostęp do zsynchronizowanych wspomnień w trybie offline. Domyślnie wspomnienia wyświetlane są od najnowszych datą (pole **memoryDate**). Dla pozycji o takiej samej dacie wspomnienia decydującym czynnikiem jest czas zapisania wspomnienia w aplikacji (pole **uploadTimeStamp**). Każde wspomnienie jest reprezentowane przez odpowiedni kafelek.



Rysunek 20
Struktura ekranu galerii wspomnień

Zgodnie z oznaczeniami przedstawionymi na **Rysunku 20**, użytkownik może:

1. **Zmienić rozmiar kafelków wspomnień** (czerwone zaznaczenie) – domyślnie kafelki są małe, ale mogą zostać ustawione na duże, jak na **Rysunku 21**.



Rysunek 21
Duże kafelki wspomnień

2. **Odwrócić kolejność sortowania wspomnień** (zielone zaznaczenie) – od najstarszych lub od najnowszych.

3. **Wybrać wspomnienie z listy** (żółte zaznaczenie) – wybór wspomnienia z listy poprzez kliknięcie w jego kafelek sprawia, że użytkownik przeniesie się na ekran szczegółów danego wspomnienia (opisany w **Rozdziale 6.6**).

Zawartość ekranu galerii wspomnień w przypadku braku zapisanych wspomnień lub wystąpienia błędu w czasie ich ładowania z Cloud Firestore opisano w **Rozdziale 6.7.4**.

6.4.2 Ekran dodawania wspomnienia

Ekran dodawania wspomnienia jest ekranem dostępnym jedynie w wersji aplikacji na system Android. **Rysunek 22** przedstawia strukturę omawianego ekranu.

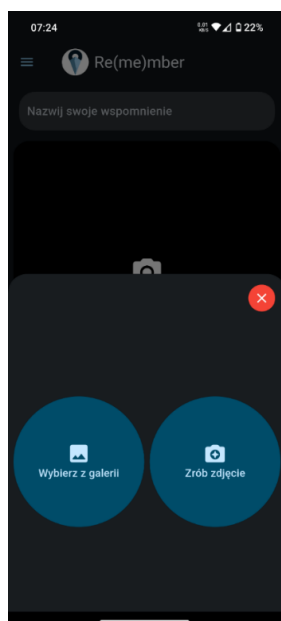


Rysunek 22
Struktura ekranu dodawania wspomnienia

Użytkownik, aby dodać wspomnienie, musi zgodnie z przedstawioną strukturą ekranu (kolejność uzupełniania informacji związanych ze wspomnieniem jest dowolna):

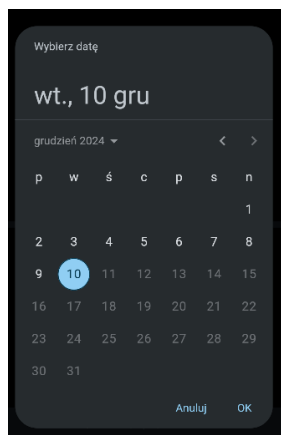
1. **Nazwać wspomnienie** (czerwone zaznaczenie).
2. **Wybrać zdjęcie do zapisania** (zielone zaznaczenie) – po wybraniu opisywanego w tym punkcie widgetu pojawia się przedstawiona na **Rysunku 23** dolna szuflada. Z jej pomocą użytkownik decyduje, czy chce wybrać zdjęcie zapisane w pamięci

urządzenia, czy zrobić nowe (w tym celu musi zezwolić aplikacji na dostęp do kamery).



Rysunek 23
Szuflada wyboru źródła zdjęcia

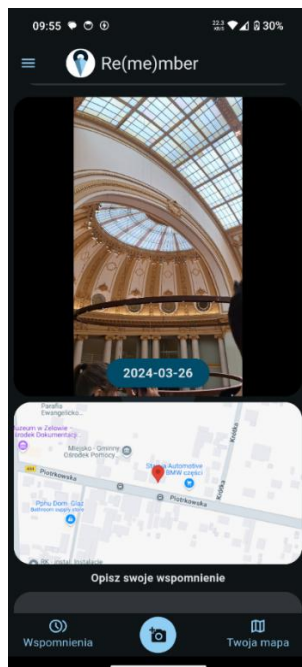
3. **Wybrać datę wspomnienia** (żółte zaznaczenie) – domyślnie podejmowana jest próba odczytania tej informacji z danych **EXIF** zdjęcia. O niepowodzeniu użytkownik jest informowany i musi wtedy wybrać datę ręcznie, z użyciem oznaczonego przycisku i korzystając z okna wyboru daty (**Rysunek 24**).



Rysunek 24
Okno wyboru daty

4. **Wybrać lokalizację wspomnienia** (niebieskie zaznaczenie) – wybór tego widgetu przenosi użytkownika na ekran mapy w trybie wyboru lokalizacji. Z jego poziomu może wybrać lokalizację ręcznie lub pobrać swoje aktualne położenie. Bardziej szczegółowy opis tego ekranu zawarto w **Rozdziale 6.5.1**
5. **Dodać opis wspomnienia** (fioletowe zaznaczenie).
6. **Zatwierdzić wprowadzone dane i zapisać wspomnienie za pomocą przycisku „Zapisz wspomnienie”** (białe zaznaczenie).

Wybór zdjęcia lub lokalizacji sprawia, że w miejscu informacji o ich niepodaniu pojawiają się odpowiednie reprezentacje (**Rysunek 25**). Bezpośrednio po zapisaniu lokalizacji podejmowana jest próba odczytania powiązanego z nią adresu (Geocoding API). Z tego powodu zapisanie wybranej lokalizacji bez dostępu do Internetu zakończy się informacją o błędzie. Przerywa to proces dodawania lokalizacji i skutkuje brakiem jej reprezentacji.

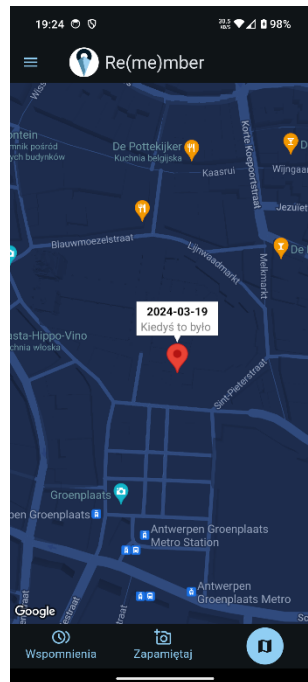


Rysunek 25
Reprezentacje wybranego zdjęcia i lokalizacji

Po wybraniu opcji „**Zapisz wspomnienie**” sprawdzane jest połączenie z Internetem i poprawność uzupełnienia wszystkich pól. Sukces oznacza wysłanie wybranego zdjęcia do Cloud Storage, a następnie utworzenie nowego wspomnienia w Cloud Firestore. Zależnie od prędkości łącza, proces ten może zająć nawet kilkadziesiąt sekund. Jeżeli nie uda się wysłać zdjęcia do Cloud Storage w ciągu trzydziestu sekund, to dodawanie wspomnienia jest przerywane. Zapobiega to długiemu procesowi oczekiwania, a tym samym frustracji użytkownika. O wystąpieniu błędu na którymś z etapów, użytkownik jest informowany za pomocą stosownego okna dialogowego. Po pomyślnym zapisaniu wspomnienia użytkownik jest przenoszony na ekran galerii wspomnień.

6.4.3 Ekran mapy wspomnień

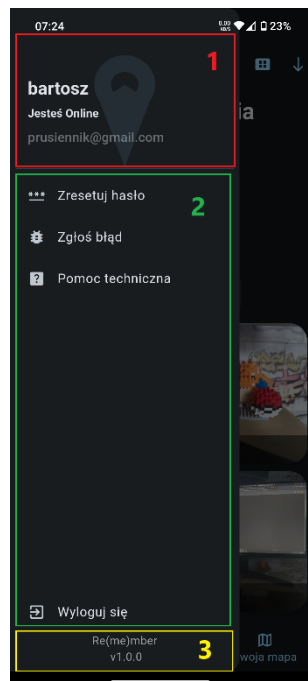
Ekran mapy wspomnień zawiera znaczniki reprezentujące zapisane przez użytkownika wspomnienia. Jego zawartość w przypadku braku zapisanych wspomnień lub wystąpienia błędu w czasie uzyskiwania znaczników w oparciu o dane z Cloud Firestore opisano w **Rozdziale 6.7.4**. Początkowym położeniem mapy jest położenie znacznika wspomnienia, które jest najnowsze pod kątem czasu jego zapisania w aplikacji (pole **uploadTimeStamp**). Po wybraniu znacznika pojawia się nad nim etykieta zawierająca datę i nazwę wspomnienia. Kliknięcie w nią przenosi użytkownika na ekran szczegółów danego wspomnienia. **Rysunek 26** przedstawia przykładowy ekran mapy wspomnień.



Rysunek 26
Przykładowy ekran mapy wspomnień

6.4.4 Szuflada użytkownika

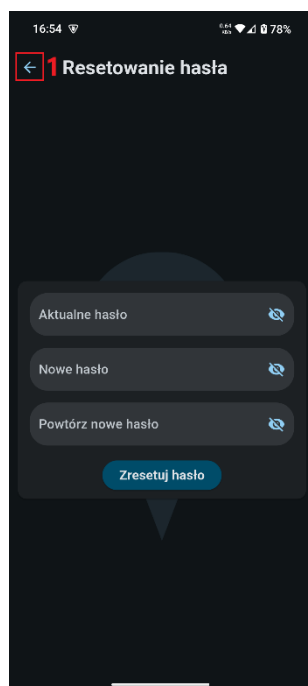
Szuflada użytkownika jest elementem dostępnym z poziomu każdej konfiguracji ekranu zawartości. Jest otwierana przy użyciu przycisku obecnego w lewym górnym rogu ekranu.



Rysunek 27
Struktura szuflady użytkownika

Zgodnie z informacjami przedstawionymi na **Rysunku 27**, szufladę użytkownika można podzielić na trzy główne części:

1. **Nagłówek** (czerwone zaznaczenie) – zawiera informacje o użytkowniku w postaci jego nazwy i adresu e-mail oraz informację o tym, czy aplikacja posiada obecnie połączenie z Internetem.
2. **Opcje interakcji** (zielone zaznaczenie) – zawarte w tej części przyciski umożliwiają:
 - a. Zresetowanie hasła z wykorzystaniem formularza przedstawionego na **Rysunku 28**.



Rysunek 28
Formularz resetowania hasła z poziomu szuflady użytkownika

Do resetowania hasła niezbędne jest połączenie z Internetem. Sam proces odbywa się z wykorzystaniem Firebase Authentication. Poprawność wprowadzonego przez użytkownika hasła jest sprawdzana poprzez podjęcie próby ponownej autentykacji. O niepowodzeniu użytkownik jest informowany. Sukces oznacza wylogowanie z aplikacji i konieczność ponownego zalogowania się z wykorzystaniem nowego hasła. Informuje go o tym stosowny panel informacyjny. Wykorzystując przycisk oznaczony na **Rysunku 28** cyfrą **1** (kolorem czerwonym), możliwy jest powrót do szuflady użytkownika.

- b. Zgłoszenie napotkanego błędu z użyciem otwieranego przez aplikację formularza, opartego na Google Forms.
- c. Skontaktowanie się z pomocą techniczną poprzez wiadomość e-mail.
- d. Wylogowanie się z aplikacji.

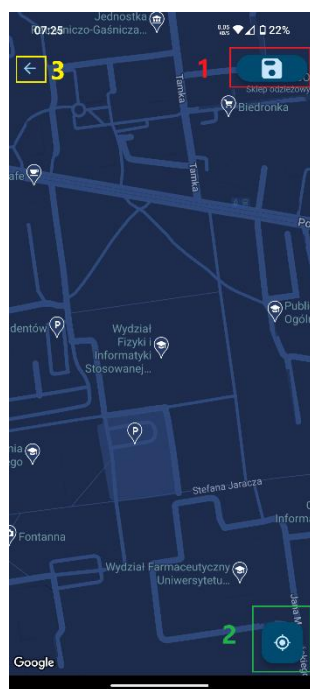
3. **Informacje o wersji aplikacji** (żółte zaznaczenie).

6.5 Ekran mapy

Ekran mapy występuje w aplikacji kilka razy, pracując przy tym w różnych trybach.

6.5.1 Ekran mapy w trybie wyboru lokalizacji

Ten tryb jest wykorzystywany podczas wybierania lokalizacji, przy dodawaniu nowego wspomnienia. Decydując się na ręczne wybranie lokalizacji, w wyznaczonym przez użytkownika miejscu tworzony jest odpowiedni znacznik. Jeżeli użytkownik chce, aby pobrana została jego aktualna lokalizacja, musi włączyć w urządzeniu usługi lokalizacyjne oraz zezwolić na dostęp do lokalizacji. Pomagają mu w tym odpowiednie okna dialogowe. Niespełnienie któregoś ze wspomnianych wymagań skutkuje komunikatem o błędzie. Po zakończeniu procesu mapa przenosi się do uzyskanej lokalizacji i wyświetla w tym miejscu odpowiedni znacznik. Pozwala to użytkownikowi na weryfikację poprawności uzyskanej lokalizacji i ręczne wprowadzenie ewentualnej korekty. Zapisanie wybranej lokalizacji odbywa się tylko po dokonaniu tego przez użytkownika.



Rysunek 29
Struktura ekranu mapy w trybie wyboru lokalizacji

Zgodnie ze strukturą ekranu mapy w trybie wyboru lokalizacji (**Rysunek 29**):

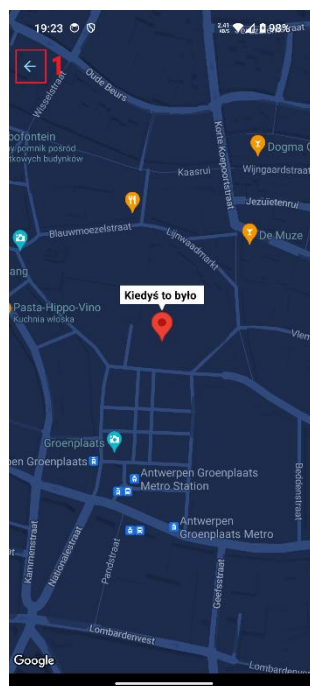
1. **Zapisanie wybranej lokalizacji i opuszczenie ekranu** (czerwone zaznaczenie).
2. **Pobranie aktualnej lokalizacji użytkownika** (zielone zaznaczenie) – zależnie od warunków (np. zasięgu sieci komórkowej) czas pobrania lokalizacji może być różny.
3. **Opuszczenie ekranu bez zapisu lokalizacji** (żółte zaznaczenie).

6.5.2 Ekran mapy w trybie mapy wspomnień

Ekran mapy w tym trybie może być częścią ekranu zawartości, gdy ten pełni rolę ekranu mapy wspomnień. Stanowi przy tym swego rodzaju historię odwiedzonych miejsc. Dokładniejszy opis ekranu mapy wspomnień, a tym samym omawianego trybu, przedstawiono w **Rozdziale 6.4.3**.

6.5.3 Ekran mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia

Ostatnim trybem, w którym może pracować mapa jest tryb powiązany z wyświetlaniem szczegółów wspomnienia. Z poziomu ekranu szczegółów wspomnienia użytkownik może wyświetlić pełnoekranową mapę, która zawiera znacznik powiązany z danym wspomnieniem. Wybranie znacznika na wyświetlanej mapie skutkuje ukazaniem się informacji o nazwie sprawdzanego wspomnienia. Przykład omawianego ekranu przedstawia **Rysunek 30**. Wykorzystując przycisk oznaczony na tym samym rysunku cyfrą **1** (kolorem czerwonym), użytkownik może wrócić do ekranu szczegółów wspomnienia.



Rysunek 30

Przykładowy ekran mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia

6.6 Ekran szczegółów wspomnienia

Ekran szczegółów wspomnienia jest otwierany poprzez wybranie danego wspomnienia z galerii.



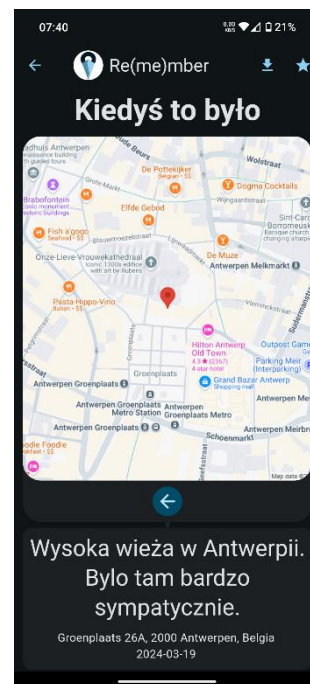
Rysunek 31
Struktura ekranu szczegółów wspomnienia

Zgodnie z **Rysunkiem 31**, elementy ekranu szczegółów wspomnienia to:

1. **Tytuł wspomnienia** (czerwone zaznaczenie).
2. **Podgląd powiązanego ze wspomnieniem zdjęcia i statycznej miniatury lokalizacji wspomnienia** (zielone zaznaczenie) – zdjęcie i miniatura wyświetlane są przy użyciu widgetu **PageView** [40], który tworzy przewijaną horyzontalnie listę. Jego zawartość można zmieniać poprzez przesuwanie jej w lewo lub w prawo (wersja na system Android), a także wykorzystując umieszczony pod nim przycisk ze strzałką. Przytrzymanie zdjęcia powiązanego ze wspomnieniem powoduje jego otwarcie w trybie pełnoekranowym (**Rysunek 32**). Do zamknięcia tego podglądu wystarczy kliknięcie w dowolne miejsce ekranu. Miniatura lokalizacji, widoczna na **Rysunku 33**, jest uzyskiwana dzięki Maps Static API. Analogicznie, jej przytrzymanie powoduje otwarcie ekranu mapy związanego z wyświetlanym wspomnieniem.

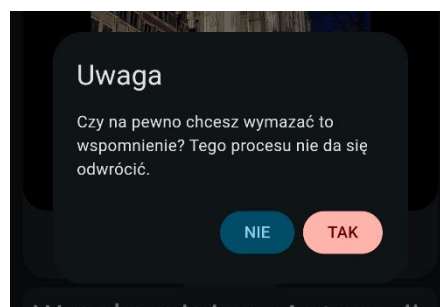


Rysunek 32
Powiązane ze wspomnieniem
zdjęcie w trybie
pełnoekranowym



Rysunek 33
Miniatura lokalizacji na
ekranie szczegółów
wspomnienia

3. **Informacje o wspomnieniu** (żółte zaznaczenie) – są to informacje zgromadzone przy zapisywaniu wspomnienia. Zawierają opis, adres miejsca (uzyskany używając Geocoding API), w którym wspomnienie zapisano i podaną przez użytkownika datę.
4. **Przycisk „Wymaż wspomnienie”** (niebieskie zaznaczenie) – użytkownik może zdecydować się wymazać zapisane przez siebie wspomnienie. W tym celu musi posiadać połączenie z Internetem oraz potwierdzić chęć usunięcia wspomnienia (**Rysunek 34**). Proces ten jest nieodwracalny. Usunięcie wspomnienia opiera się na usunięciu odpowiadającego temu wspomnieniu dokumentu z Cloud Firestore oraz powiązanego z nim zdjęcia przechowywanego w Cloud Storage.



Rysunek 34
Potwierdzenie chęci wymazania wspomnienia

5. **Przycisk służący do pobrania zdjęcia powiązanego ze wspomnieniem do pamięci urządzenia** (białe zaznaczenie) – pobranie zdjęcia wymaga połączenia

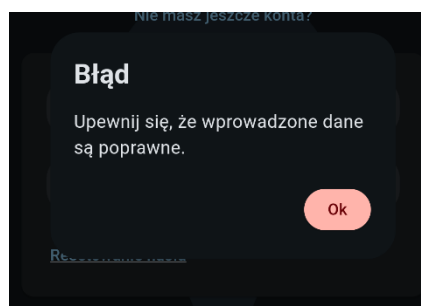
z Internetem. O pomyślnym pobraniu zdjęcia użytkownik jest odpowiednio informowany.

6. **Przycisk dodający/usuwający wspomnienie z ulubionych** (fioletowe zaznaczenie) – wypełniony lub pusty, zależnie od tego, czy wspomnienie jest tym ulubionym. Zmiana tego statusu nie wymaga połączenia z Internetem.
7. **Przycisk umożliwiający powrót do galerii wspomnień** (szare zaznaczenie).

6.7 Pozostałe elementy

6.7.1 Informacyjne okna dialogowe

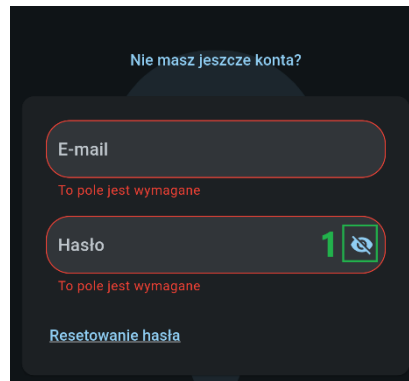
Ważnym elementem interakcji z użytkownikiem są okna dialogowe. Na potrzeby aplikacji powstało kilka ich rodzajów. Oprócz omawianych wcześniej okien powiązanych z resetowaniem hasła oraz potwierdzeniem chęci wymazania wspomnienia, opisywana aplikacja wykorzystuje okno dialogowe nastawione typowo na przekazanie użytkownikowi informacji (przede wszystkim o błędach). Wykorzystywane jest w wielu miejscach aplikacji, a jego zawartość jest zależna od tekstów przekazanych jako parametry. Przykładowe informacyjne okno dialogowe przedstawia **Rysunek 35**.



Rysunek 35
Przykładowe informacyjne okno dialogowe

6.7.2 Pola tekstowe

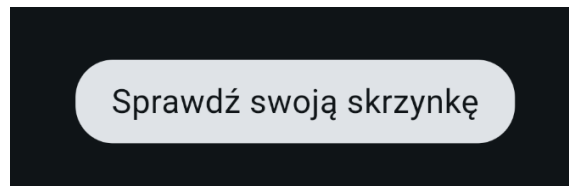
Każde zawarte w aplikacji pole tekstowe, z którym użytkownik może wejść w interakcję, korzysta z odpowiednich mechanizmów walidacyjnych. Dodatkowo, pola te zawierają odpowiednie etykiety określające rodzaj informacji, które powinny być w nich wprowadzane. Mechanizmy te mają na celu zadbanie o poprawność wprowadzanych danych, a tym samym stabilność działania aplikacji. Sprawdzane są np. prawidłowość struktury adresu e-mail czy występowanie spacji w tworzonych hasłach. Przykład nieudanej walidacji przedstawia **Rysunek 36**. Oznaczona cyfrą **1** na tym samym rysunku (zielone zaznaczenie) ikona oka pozwala na zmianę widoczności hasła (domyślnie ukryte). Opcja ta jest dostępna we wszystkich polach tekstowych związanych z wprowadzaniem przez użytkownika hasła.



Rysunek 36
Przykład nieudanej walidacji

6.7.3 Panel informacyjny

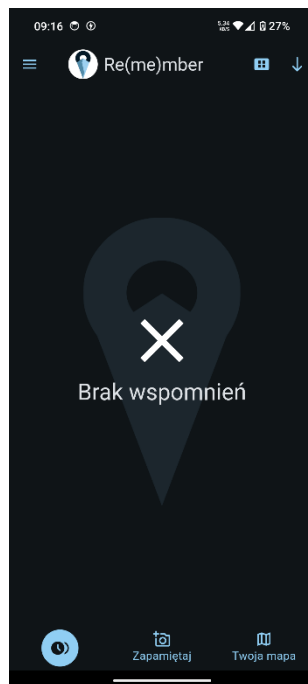
Resetując hasło lub zapisując powiązane ze wspomnieniem zdjęcie w pamięci urządzenia (wersja na system Android), użytkownik jest informowany o powodzeniu tych operacji poprzez wyświetlenie na dole ekranu specjalnego panelu informacyjnego. Przykładowy panel informacyjny przedstawia **Rysunek 37**. Podobnie jak w przypadku informacyjnego okna dialogowego, jego zawartość jest zależna od przekazanych parametrów.



Rysunek 37
Przykładowy panel informacyjny

6.7.4 Widget Infotext

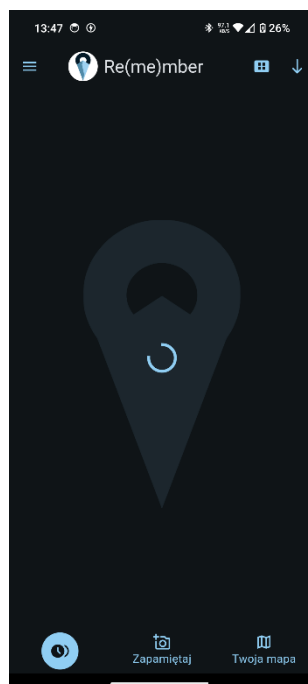
Dla sytuacji, w której użytkownik nie zgromadził jeszcze żadnych wspomnień lub w czasie ich ładowania nastąpił nieoczekiwany błąd, przygotowano specjalny widget informacyjny. Zależnie od potrzeb, zastępuje wtedy listę wspomnień w galerii lub mapę zgromadzonych wspomnień. Wyświetlana zawartość tekstowa jest zależna od sytuacji, której dotyczy, a tym samym przekazanych parametrów. Przykładowy widget **Infotext** przedstawia **Rysunek 38**.



Rysunek 38
Przykładowy widget Infotext

6.7.5 Indykator ładowania

Podczas procesów, które mogą zająć więcej czasu (np. ładowania danych z Cloud Firestore czy pobierania aktualnej lokalizacji), wyświetlane są odpowiednie indykatory ładowania. Informują użytkownika o trwających procesach, zapewniając że aplikacja działa i nie uległa zawieszeniu. Przykład indykatora ładowania przedstawia **Rysunek 39**.



Rysunek 39
Przykładowy indykator ładowania

Rozdział 7

Wybrane elementy implementacji

7.1 Nasłuchiwanie stanu uwierzytelnienia użytkownika

Chcąc na bieżąco sprawdzać stan zalogowania użytkownika, zastosowano nasłuchiwanie odpowiedniego strumienia, emitującego zdarzenia związane ze zmianą stanu uwierzytelnienia oraz zmianami **ID token**. Mechanizm za to odpowiedzialny przedstawia **Rysunek 40. StreamBuilder** [41] stanowi w tym wypadku swego rodzaju widget startowy aplikacji, dzięki czemu cała jej zawartość jest definiowana dynamicznie, w oparciu o aktualny stan uwierzytelnienia. Pozwala to na automatyczne zarządzanie dostępem użytkownika do aplikacji.

```
home: StreamBuilder(  
  stream: FirebaseAuth.instance.idTokenChanges(),  
  builder: (context, snapshot) {  
    if (snapshot.connectionState == ConnectionState.waiting) {  
      return const Scaffold(  
        body: Center(  
          child: CircularProgressIndicator(),  
        ), // Center  
      ); // Scaffold  
    }  
    if (!snapshot.hasData) {  
      return const AuthScreen();  
    }  
    return const ContentScreen();  
  },  
), // StreamBuilder
```

Rysunek 40
Mechanizm nasłuchiwania stanu uwierzytelnienia użytkownika

Zależnie od stanu strumienia zwrócone mogą zostać trzy rodzaje zawartości aplikacji:

1. **Indykator ładowania** – zwracany, gdy strumień oczekuje na dane.
2. **Ekran logowania/rejestracji** – zwracany, gdy dane ze strumienia są puste, a tym samym użytkownik nie jest zalogowany.
3. **Ekran zawartości** – zwracany, gdy strumień zawiera dane, a tym samym użytkownik jest poprawnie zalogowany.

7.2 Obsługa wyjątków związanych z Firebase

Klauzula **try – catch** stanowi kluczowy element obsługi napotkanych w aplikacji wyjątków. Dodatkowo, jak wspomniano w poprzednich rozdziałach, podstawę uwierzytelniania użytkowników i przechowywania danych stanowi platforma Firebase.

Chcąc uprościć obsługę potencjalnych wyjątków związanych z Firebase, zaimplementowano funkcję przedstawioną na **Rysunku 41**.

```
Future<void> handleFirebaseError(String code, BuildContext context) async {
  String message;
  switch (code) {
    case 'invalid-credential':
      message = invalidCredentials;
    case 'network-request-failed':
      message = noConnection;
    case 'email-already-in-use':
      message = emailTaken;
    case 'invalid-email':
      message = invalidEmail;
    case 'unavailable':
      message = serviceUnavailable;
    default:
      message = unknownError;
  }
  await showInfoPopup(
    context,
    message,
  );
}
```

Rysunek 41
Funkcja obsługująca napotkane wyjątki Firebase

Po wyłapaniu wyjątku, w oparciu o dostarczony kod błędu, wyświetlane jest powiązane z tym błędem informacyjne okno dialogowe. Przykład wyłapania i obsługi wyjątku Firebase przy pomocy omawianej funkcji przedstawia **Rysunek 42**. Przedstawione na nim sprawdzenie **if(!mounted)** to bardzo ważny zabieg, weryfikujący, czy widget, który wywołał daną funkcję, nadal jest częścią drzewa widgetów. Jeżeli tak nie jest, aby uniknąć potencjalnych błędów, działanie funkcji jest przerywane.

```
try {
  await FirebaseAuth.instance.sendPasswordResetEmail(email: email);
} on FirebaseAuthException catch (e) {
  if (!mounted) return;
  setState(() {
    _isLoading = false;
  });
  await handleFirebaseError(e.code, context);
  return;
}
```

Rysunek 42
Przykład wyłapania i obsługi wyjątku Firebase

Nieco inaczej obsługiwane jest, wspomniane w **Rozdziale 6.4.2**, przekroczenie limitu czasu wysyłania zdjęcia do Cloud Storage. Zastosowano mechanizm, dzięki któremu przekroczenie założonego limitu skutkuje przerwaniem procesu wysyłania oraz wyjątkiem **TimeoutException**. Po jego napotkaniu wyświetlane jest odpowiednie okno dialogowe, a funkcja przerywa swoje działanie, zwracając **null** w miejsce adresu **URL** wysłanego zdjęcia. Rozwiązanie to przedstawia **Rysunek 43**.

```

try {
  final uploadTask = storageRef.putFile(_chosenImage!);
  await uploadTask.timeout(
    const Duration(seconds: 30),
    onTimeout: () async {
      await uploadTask.cancel();
      throw TimeoutException(
        uploadError,
      );
    },
  );
  final url = await storageRef.getDownloadURL();
  return url;
} on TimeoutException catch (e) {
  if (!mounted) return null;
  showInfoPopup(context, e.message!);
  return null;
}

```

Rysunek 43

Fragment funkcji odpowiedzialnej za wysyłanie zdjęcia do Cloud Storage, prezentujący wykorzystanie `TimeoutException`

7.3 Mapa nazw dla danych z Firebase

Nazwy kolekcji, dokumentów i pól z Cloud Firestore oraz folderów z Cloud Storage wydzielono do specjalnej mapy. Dotyczy to nazw, które nie ulegają zmianie. Choć aktualnie klucz i wartość są takie same, to takie rozwiązanie umożliwia przechowywanie wszystkich nazw w jednym miejscu, a tym samym łatwiejsze wprowadzanie modyfikacji. Zmiana wartości przypisanej do danego klucza automatycznie aktualizuje tę wartość w całej aplikacji. Utworzoną mapę przedstawia **Rysunek 44**.

```

const Map<String, String> firebaseDataKeys = {
  "geopoint": "geopoint",
  "address": "address",
  "title": "title",
  "description": "description",
  "memoryDate": "memoryDate",
  "uploadTimeStamp": "uploadTimeStamp",
  "username": "username",
  "email": "email",
  "userId": "userId",
  "imageUrl": "imageUrl",
  "isFavourite": "isFavourite",
  "memories_by_user": "memories_by_user",
  "memories": "memories",
  "user_memories": "user_memories",
};

```

Rysunek 44

Struktura mapy nazw dla danych z Firebase

7.4 Cached Network Image

Źródłem większości obecnych w aplikacji zdjęć (wyjątek stanowi jedynie zdjęcie wybrane przez użytkownika na ekranie dodawania wspomnienia) jest Internet. Dotyczy to zarówno obrazów powiązanych ze wspomnieniem, jak i miniatur lokalizacji, otrzymanych przy użyciu Maps Static API. Mając na celu zwiększenie wydajności aplikacji, zastosowano pakiet **cached_network_image** [42]. Po pierwszym pobraniu zdjęcia są przechowywane lokalnie. Pozwala to na ich szybsze ładowanie, przy jednoczesnym zmniejszeniu pobieranych danych. Wpływa to więc pozytywnie nie tylko na wydajność samej aplikacji, ale także na komfort użytkownika. Wykorzystując omawiany pakiet, stworzony został widget **CustomCachedImage**, którego kod źródłowy przedstawiono na **Rysunku 45**.

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/material.dart';
import 'package:remember/helpers/strings.dart';

You, 2 minuty temu | 1 author (You)
class CustomCachedImage extends StatelessWidget {
  const CustomCachedImage({
    required this.imageUrl,
    this.fit = BoxFit.cover,
    super.key,
  });

  final String imageUrl;
  final BoxFit fit;
  @override
  Widget build(BuildContext context) {
    return CachedNetworkImage(
      placeholder: (context, url) => const Center(
        child: CircularProgressIndicator(),
      ), // Center
      errorWidget: (context, url, error) {
        return Center(
          child: Container(
            width: double.infinity,
            color: Colors.black,
            child: const Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(
                  size: 35,
                  Icons.error,
                  color: Colors.white,
                ), // Icon
                Text(
                  textAlign: TextAlign.center,
                  failedDownload,
                  style: TextStyle(
                    color: Colors.white,
                  ), // TextStyle
                ), // Text
              ],
            ), // Column
          ), // Container
        ); // Center
      },
      imageUrl: imageUrl,
      fit: fit,
    ); // CachedNetworkImage
  }
}
```

Rysunek 45
Kod źródłowy widgetu CustomCachedImage

Najważniejsze elementy **CustomCachedImage** to:

1. **placeholder** – wyświetla indykator ładowania zdjęcia.
2. **errorWidget** – wyświetla widget informujący o błędzie napotkanym w czasie ładowania zdjęcia. Przykład wystąpienia błędu przedstawia **Rysunek 46**.



Rysunek 46
Przykładowy błąd ładowania CustomCachedImage

3. **imageUrl** – adres **URL** obrazu, który ma zostać załadowany.

7.5 Zarządzanie stanem

Zastosowanie pakietu **flutter_riverpod** [43] umożliwiło wprowadzenie mechanizmu zarządzania stanem. Z jego pomocą można automatycznie przebudować jeden widget w reakcji na modyfikacje stanu dokonane w innym widżecie. Na zmiany te może reagować zarówno **StatefulWidget**, jak i **StatelessWidget**. Ten drugi, jak wspomniano w **Rozdziale 4.2**, nie posiada własnego stanu. W omawianym przypadku jest zwyczajnie odtwarzany w oparciu o nowe informacje dostarczone przez odpowiednie komponenty. Może to sprawiać wrażenie, że **StatelessWidget** zachowuje się jak **StatefulWidget**.

Plik **providers.dart** (**Rysunek 47**) zawiera wykorzystywane wariacje **StateProvider** [44], czyli komponentu umożliwiającego przechowywanie i modyfikowanie stanu opartego na prostych typach danych. Przy definiowaniu **StateProvider** określa się również domyślną wartość stanu, która jest przywracana przy każdym ponownym uruchomieniu aplikacji.

```
import 'package:flutter_riverpod/flutter_riverpod.dart';

final indexProvider = StateProvider<int>(  
  (ref) => 0,  
);

final memoryOrderProvider = StateProvider<bool>(  
  (ref) => true,  
);

final memoryOverlayProvider = StateProvider<bool>(  
  (ref) => false,  
);
```

Rysunek 47
Zawartość pliku providers.dart

Zgodnie z zawartością pliku **providers.dart**:

1. **indexProvider** – odpowiada za aktualnie wybraną pozycję paska nawigacyjnego. Domyślna wartość stanu to **0**, co wpływa na wyświetlenie ekranu galerii wspomnień.
2. **memoryOrderProvider** – odpowiada za aktualnie wybraną kolejność sortowania wspomnień w galerii. Domyślna wartość stanu to **true**, co wpływa na sortowanie wspomnień od najnowszego.
3. **memoryOverlayProvider** – odpowiada za aktualnie wybrany rozmiar kafelków wspomnień w galerii. Domyślna wartość stanu to **false**, co wpływa na wyświetlanie małych kafelków.

Widżety, które wymagają implementacji mechanizmu zarządzania stanem z wykorzystaniem omawianego pakietu, muszą zostać odpowiednio do tego przystosowane. Proces ten przebiega przy użyciu komponentów dostarczonych przez **flutter_riverpod**. **StatelessWidget** powinien zostać przekształcony w **ConsumerWidget**, a **StatefulWidget** w **ConsumerStatefulWidget**. Ponadto całą aplikację (korzeń drzewa widżetów) należy otoczyć dostarczonym przez omawiany pakiet widżetem, jakim jest **ProviderScope**.

Takie przygotowanie widżetów umożliwia przede wszystkim:

1. **Następowanie zmian stanu z wykorzystaniem `ref.watch`** – każda taka zmiana wywołuje metodę **build** (to w niej powinno się następować zmian) widżetu, który nastękuje zmian stanu. Przykład wykorzystania tego mechanizmu przedstawia **Rysunek 48**.

```
_currentIndex = ref.watch(indexProvider);
```

Rysunek 48

Przykład następowania zmian stanu z wykorzystaniem `ref.watch`

2. **Jednorazowe odczytanie stanu przy użyciu `ref.read`** – dzięki wykorzystaniu **StateProvider** możliwa jest również modyfikacja stanu. Przykład użycia **`ref.read`** do modyfikacji stanu przedstawia **Rysunek 49**.

```
onTap: (index) {  
  ref.read(indexProvider.notifier).state = index;  
},
```

Rysunek 49

Przykład użycia `ref.read` do modyfikacji stanu

Zastosowanie omawianych mechanizmów jest kluczowym elementem zarządzania aktualną kolejnością sortowania i rozmiarem kafelków wspomnień. Choć wartości te są modyfikowane przez przyciski obecne na pasku aplikacji, każda zmiana powoduje automatyczną przebudowę widżetu **MemoriesList** (i jego odpowiednich potomków), położonego w innej części drzewa widżetów. Zarządzanie stanem stanowi także podstawę działania paska nawigacyjnego, a tym samym pomaga zdefiniować aktualną

konfigurację ekranu zawartości. Warto tutaj zauważyć, że zaimplementowane mechanizmy to jedynie niewielka część tego, co oferuje omawiany pakiet.

7.6 Rozróżnianie platform

Flutter oferuje wbudowaną możliwość sprawdzenia aktualnie wykorzystywanej platformy. Poprzez użycie odpowiednich stałych i właściwości (np. **kIsWeb** dla wersji webowej lub **Platform.isAndroid** dla wersji na system Android) można bezproblemowo zweryfikować aktualną platformę, a tym samym dopasować interfejs, a także oferowane przez aplikację funkcje. **Rysunek 50** przedstawia proste wykorzystanie tych mechanizmów na przykładzie funkcji, która dopasowuje ekran zawartości do pozycji wybranej z paska nawigacyjnego, sprawdzając przy tym, czy aplikacja działa w wersji webowej.

```
Widget get _currentContent {
  switch (_currentIndex) {
    case 0:
      return const MemoriesGallery();
    case 1:
      return kIsWeb ? const MemoriesMap() : const NewMemory();
    case 2:
      return const MemoriesMap();
    default:
      return const MemoriesGallery();
  }
}
```

Rysunek 50
Przykład sprawdzenia tego, czy aplikacja działa w wersji webowej

7.7 Implementacja ekranu galerii wspomnień

Ekran galerii wspomnień jest kluczowym elementem aplikacji. Został przygotowany w sposób modularny, aby ułatwić jego potencjalny rozwój, zwiększyć czytelność kodu i uniknąć zbędnego wywołania metody **build** głównego widgetu galerii wspomnień. Podstawową część implementacji tego ekranu stanowi nastuchiwanie odpowiedniego strumienia, który dostarcza dane o wspomnieniach z Cloud Firestore.

Zgodnie z **Rysunkiem 51**, w oparciu o te dane ekran może zawierać:

1. **Indykator ładowania** – występuje w czasie ładowania wspomnień.
2. **Widget Infotext z informacją o błędzie** – występuje, gdy w czasie ładowania wspomnień nastąpił błąd.
3. **Widget Infotext z informacją o braku zapisanych wspomnień** – występuje, gdy użytkownik nie posiada żadnych zapisanych wspomnień.
4. **Listę wspomnień (widget MemoriesList)** – występuje w pozostałych przypadkach.

```

child: StreamBuilder(
  stream: stream,
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(
        child: CircularProgressIndicator(),
      ); // Center
    }
    if (snapshot.hasError) {
      return const Infotext(text: unknownError);
    }
    if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
      return const Infotext(text: noMemories);
    }
    return MemoriesList(memories: snapshot.data!.docs);
  },
), // StreamBuilder

```

Rysunek 51
Warunki definiujące zawartość ekranu galerii wspomnień

Z punktu widzenia implementacji najważniejszy jest ostatni punkt – widget **MemoriesList**. To właśnie on odpowiada za właściwe wyświetlenie listy wspomnień. Przeprowadza sortowanie wspomnień oraz ich grupowanie, w zależności od tego, czy są tymi ulubionymi. Metodę **build** widgetu **MemoriesList** przedstawia **Rysunek 52**.

```

@override
Widget build(BuildContext context, WidgetRef ref) {
  final overlay = ref.watch(memoryOverlayProvider);
  final descending = ref.watch(memoryOrderProvider);
  final sortedMemories = _getSortedMemories(descending);
  final groupedMemories = _getGroupedMemories(sortedMemories);
  final favouriteMemories = groupedMemories['favouriteMemories']!;
  final basicMemories = groupedMemories['basicMemories']!;

  return CustomScrollView(
    slivers: [
      if (favouriteMemories.isNotEmpty)
        const MemoriesSliverHeader(text: favMemories),
      MemoriesSliver(
        memories: favouriteMemories,
        overlay: overlay,
      ), // MemoriesSliver
      if (basicMemories.isNotEmpty)
        const MemoriesSliverHeader(text: yourMemories),
      MemoriesSliver(
        memories: basicMemories,
        overlay: overlay,
      ), // MemoriesSliver
    ],
  ); // CustomScrollView
}

```

Rysunek 52
Metoda build widgetu MemoriesList

Wartość **overlay** jest związana z aktualnym rozmiarem kafelków wspomnień (**true** – duże, **false** – małe), a **descending** z kolejnością sortowania wspomnień (**true** – od najnowszych, **false** – od najstarszych). Informacje na temat sposobu uzyskiwania tych wartości zawarto w **Rozdziale 7.5**.

Rysunek 53 przedstawia funkcję odpowiedzialną za sortowanie wspomnień.

```
List<QueryDocumentSnapshot<Map<String, dynamic>>> _getSortedMemories(
    bool descending) {
    return List.of(memories)
        ..sort((a, b) {
            final memoryDateA = a.data()[firebaseDataKeys['memoryDate']]! as String;
            final memoryDateB = b.data()[firebaseDataKeys['memoryDate']]! as String;
            final comparison = descending
                ? memoryDateB.compareTo(memoryDateA)
                : memoryDateA.compareTo(memoryDateB);
            if (comparison != 0) return comparison;
            final uploadTimeStampA =
                a.data()[firebaseDataKeys['uploadTimeStamp']]! as Timestamp;
            final uploadTimeStampB =
                b.data()[firebaseDataKeys['uploadTimeStamp']]! as Timestamp;
            return descending
                ? uploadTimeStampB.compareTo(uploadTimeStampA)
                : uploadTimeStampA.compareTo(uploadTimeStampB);
        });
}
```

Rysunek 53
Funkcja odpowiedzialna za sortowanie wspomnień

Początkowo porównywana jest data wspomnienia (pole **memoryDate**). Jeżeli wartość ta jest równa dla obu wspomnień, to porównywany jest czas zapisania wspomnienia w aplikacji (pole **uploadTimeStamp**).

Zależnie od wybranej kolejności sortowania, porównuje się element **B** z elementem **A** (porządek malejący) lub element **A** z elementem **B** (porządek rosnący). Porównując dwa elementy, mogą zostać zwrócone wartości:

1. **0** – elementy **równe**.
2. **1** – element pierwszy jest **większy** od drugiego.
3. **-1** – element pierwszy jest **mniejszy** od drugiego.

Posortowane wspomnienia są następnie grupowane przy użyciu funkcji przedstawionej na **Rysunku 54**. Grupowanie polega na sprawdzeniu wartości pola **isFavourite** każdego wspomnienia i na tej podstawie, przydzieleniu wspomnienia do odpowiedniej listy.

```

Map<String, List<QueryDocumentSnapshot<Map<String, dynamic>>>>
  _getGroupedMemories(
    List<QueryDocumentSnapshot<Map<String, dynamic>>> memories) {
  final Map<String, List<QueryDocumentSnapshot<Map<String, dynamic>>>>
    grouped = {
      'favouriteMemories': [],
      'basicMemories': [],
    };
  for (final memory in memories) {
    final isFavourite =
      memory.data()[firebaseDataKeys['isFavourite']]! as bool;
    isFavourite
      ? grouped["favouriteMemories"]!.add(memory)
      : grouped["basicMemories"]!.add(memory);
  }
  return grouped;
}

```

Rysunek 54

Funkcja odpowiedzialna za grupowanie wspomnień

W oparciu o posortowane i pogrupowane wspomnienia wyświetlane są właściwe nagłówki oraz siatki kafelków reprezentujących wspomnienia. Kliknięcie kafelka przenosi użytkownika na ekran szczegółów danego wspomnienia.

Rysunek 55 przedstawia metodę **build** widgetu **MemoriesSliver**, reprezentującego siatkę kafelków wspomnień.

```

@override
Widget build(BuildContext context) {
  return SliverGrid(
    delegate: SliverChildBuilderDelegate(
      childCount: memories.length,
      (context, index) => MemoryCard(
        data: memories[index].data(),
        id: memories[index].id,
      ), // MemoryCard
    ), // SliverChildBuilderDelegate
    gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(
      maxCrossAxisExtent: overlay ? 550 : 250,
      crossAxisSpacing: 2.0,
      mainAxisSpacing: 2.0,
    ), // SliverGridDelegateWithMaxCrossAxisExtent
  ); // SliverGrid
}

```

Rysunek 55

Metoda build widgetu MemoriesSliver

Każdy reprezentujący wspomnienie kafelek to odpowiednia instancja widgetu **MemoryCard**. Użycie **SliverChildBuilderDelegate** [45] sprawia, że te są budowane tylko wtedy, gdy mają być wyświetlane na ekranie. Takie podejście optymalizuje zużycie zasobów i poprawia wydajność aplikacji. Dodatkowo, wartości parametru **maxCrossAxisExtent** (odpowiedzialnego za rozmiar omawianych kafelków) zostały dobrane w oparciu o liczne testy i w sposób mający zapewnić spójność doświadczenia na różnych urządzeniach oraz czytelność interfejsu. Wykorzystując ten parametr, Flutter

automatycznie dobiera dokładny rozmiar oraz liczbę kafelków wspomnień zawartych w jednym rzędzie kolumny.

7.8 Dodawanie i usuwanie wspomnienia z ulubionych

Ekran szczegółów wspomnienia otrzymuje jako parametry komplet danych dotyczących danego wspomnienia. Podczas inicjalizacji tego ekranu, zmiennej `_isFavourite` przypisywana jest wartość pola `isFavourite` dokumentu reprezentującego wspomnienie w Cloud Firestore. Jest to zmienna typu `bool`, powiązana z przyciskiem gwiazdki odpowiedzialnym za dodawanie i usuwanie wspomnienia z ulubionych. Zależnie od jej wartości, przycisk może być wypełniony (wspomnienie jest ulubionym – `true`) lub pusty (wspomnienie nie jest ulubionym – `false`). Za zmianę tego, czy wspomnienie jest ulubionym, odpowiada funkcja przedstawiona na **Rysunku 56**.

```
void _toggleFavourite() async {
  setState(() {
    _isFavourite = !_isFavourite;
  });
  try {
    await _firestore
      .collection(firebaseDataKeys['memories_by_user']!)
      .doc(_user)
      .collection(firebaseDataKeys['memories']!)
      .doc(widget.id)
      .update({firebaseDataKeys["isFavourite"]!: _isFavourite});
  } on FirebaseException catch (e) {
    if (!mounted) return;
    setState(() {
      _isFavourite = !_isFavourite;
    });
    handleFirebaseError(e.code, context);
    return;
  } catch (_) {
    if (!mounted) return;
    setState(() {
      _isFavourite = !_isFavourite;
    });
    showInfoPopup(context, unknownError);
    return;
  }
}
```

Rysunek 56

Funkcja odpowiedzialna za dodawanie i usuwanie wspomnienia z ulubionych

Funkcja ta wywoływana jest przez wspomniany wcześniej przycisk. Proces rozpoczyna się od aktualizacji stanu, przypisującej zmiennej `_isFavourite` jej odwrotność. Następnie podejmowana jest próba przypisania tej samej wartości polu `isFavourite` dokumentu, który reprezentuje wspomnienie w Cloud Firestore. Niepowodzenie oznacza przywrócenie zmiennej `_isFavourite` jej pierwotnej wartości (poprzez aktualizację stanu) oraz wyświetlenie informacji o błędzie. Dzięki obsłudze trybu offline przez Cloud Firestore cały proces jest możliwy bez dostępu do Internetu. Zmodyfikowane informacje są synchronizowane po nawiązaniu połączenia.

7.9 Dodawanie szczegółów wspomnienia

7.9.1 Dodawanie zdjęcia

Przy wybieraniu zdjęcia na ekranie dodawania wspomnienia zastosowano mechanizm polegający na przekazywaniu funkcji w dół drzewa widgetów. **StatefulWidget** **NewPhotoWidget** (Rysunek 57) reprezentuje wybrane przez użytkownika zdjęcie i przyjmuje jako parametr funkcję **onChooseImage**. Ta przypisuje wybrane zdjęcie do odpowiedniej zmiennej (związanej z ekranem dodawania wspomnienia), a następnie podejmuje próbę odczytania z danych **EXIF** daty wykonania zdjęcia.

```
NewPhotoWidget(  
  onChooseImage: (image) async {  
    _chosenImage = image;  
    await _checkDateTime();  
  },  
) // NewPhotoWidget
```

Rysunek 57

Przykład użycia widgetu NewPhotoWidget

NewPhotoWidget reprezentuje wybrane zdjęcie. Po jego kliknięciu wywoływana jest funkcja przedstawiona na **Rysunku 58**. Otwiera ona widget **PhotoModal**, przekazując do niego funkcję otrzymaną wcześniej jako parametr. Przekazanie to zostaje rozbudowane o zaktualizowanie stanu **NewPhotoWidget**, tak aby ten wyświetlał wybrane zdjęcie jako swoją zawartość.

```
Future<void> _showPhotoModal() async {  
  await showModalBottomSheet(  
    constraints: const BoxConstraints.expand(),  
    context: context,  
    builder: (context) {  
      return PhotoModal(  
        onChooseImage: (image) {  
          setState(() {  
            _chosenPhoto = image;  
          });  
          widget.onChooseImage(image);  
        },  
      );  
    },  
  );  
}
```

Rysunek 58

Funkcja otwierająca widget PhotoModal

PhotoModal to dolna szuflada umożliwiająca wybór źródła zdjęcia (wybór z galerii lub wykonanie nowego). Otrzymana jako parametr funkcja stanowi część funkcji odpowiedzialnej za uzyskanie zdjęcia od użytkownika (**Rysunek 59**), której wywołanie odbywa się poprzez kliknięcie odpowiedniej wersji (zależnej od wybranego źródła zdjęcia) widgetu **PhotoModalButton** (**Rysunek 60**), czyli przycisku odpowiedzialnego za wybór zdjęcia z określonego źródła. Wykorzystując pakiet **image_picker** [46], podejmowana jest próba uzyskania zdjęcia. Jeżeli to nie zostało wybrane (**image == null**), działanie funkcji jest przerywane. W przeciwnym wypadku następuje wywołanie, przekazanej jako parametr do **PhotoModalButton**, funkcji **onChooseImage**. Ta aktualizuje zawartość widgetu **NewPhotoWidget** i wywołuje pierwszą przekazywaną funkcję – modyfikującą zawartość zmiennej przechowującej zdjęcie (powiązanej z ekranem dodawania wspomnienia) i podejmującą próbę odczytania informacji o dacie wykonania zdjęcia. Następnie szuflada jest zamykana.

```
Future<void> _choosePhoto(ImageSource source, BuildContext context) async {
  final ImagePicker picker = ImagePicker();
  final XFile? image;
  try {
    image = await picker.pickImage(source: source);
  } catch (e) {
    if (!context.mounted) return;
    showInfoPopup(context, cameraPermission);
    return;
  }

  if (image == null) return;

  onChooseImage(File(image.path));
  if (!context.mounted) return;
  Navigator.pop(context);
}
```

Rysunek 59

Funkcja odpowiedzialna za uzyskanie zdjęcia od użytkownika

```
PhotoModalButton(
  onTap: () async {
    await _choosePhoto(ImageSource.gallery, context);
  },
  text: pickGallery,
  icon: Icons.image_rounded,
), // PhotoModalButton
```

Rysunek 60

Przykład użycia widgetu PhotoModalButton

7.9.2 Dodawanie daty

Bezpośrednio po wyborze zdjęcia podejmowana jest próba odczytania z danych **EXIF** daty jego wykonania. Funkcję odpowiedzialną za tę próbę przedstawia **Rysunek 61**.

```

Future<void> _checkDateTime() async {
  final exif = await readExifFromFile(_chosenImage!);
  final exifDateTime = exif["EXIF DateTimeOriginal"]?.toString();

  if (exifDateTime == null) {
    if (!mounted) return;
    await showInfoPopup(context, noExif, title: warning);
    return;
  }
  setState(() {
    _chosenDate = exifDateTime.split(' ')[0].replaceAll(':', '-');
  });
}

```

Rysunek 61

Funkcja odpowiedzialna za próbę odczytania daty wykonania zdjęcia

Do odczytywania danych **EXIF** wykorzystywany jest odpowiedni pakiet [47]. Jeżeli informacje zawarte pod tagiem **EXIF DateTimeOriginal** są puste, to użytkownik jest informowany o konieczności ręcznego wyboru daty. Jeżeli jednak uda się je odczytać, to otrzymane dane są przekształcane do formatu odpowiedniego dla opisywanej aplikacji i przypisywane do pomocniczej zmiennej. Towarzyszy temu aktualizacja stanu.

Rysunek 62 przedstawia funkcję umożliwiającą ręczny wybór daty.

```

Future<void> _pickDateTime() async {
  final tempDateTime = await showDatePicker(
    locale: const Locale("pl"),
    initialEntryMode: DatePickerEntryMode.calendarOnly,
    initialDate: _chosenDate == null
      ? DateTime.now()
      : DateTime.tryParse(_chosenDate!),
    context: context,
    firstDate: DateTime(DateTime.now().year - 100),
    lastDate: DateTime.now(),
  );
  if (tempDateTime == null) return;
  setState(() {
    _chosenDate = tempDateTime.toString().split(' ')[0];
  });
}

```

Rysunek 62

Funkcja umożliwiająca ręczny wybór daty

Ręczny wybór daty odbywa się z wykorzystaniem odpowiedniego przycisku i otwieranego z jego pomocą okna dialogowego. Użytkownik może wybrać datę z roku, który przypada maksymalnie **100** lat przed rokiem bieżącym. Początkowo wybrana data jest datą aktualną lub wcześniej wybraną (jeżeli takowa istnieje). Wybór daty powoduje (podobnie jak dla odczytu z danych **EXIF**) przekształcenie jej do formatu odpowiedniego dla opisywanej aplikacji i przypisanie do zmiennej pomocniczej, czemu towarzyszy aktualizacja stanu. Niewybranie daty sprawia, że działanie funkcji jest przerywane.

Jak wspomniano, po odczytaniu lub ręcznym wybraniu daty następuje aktualizacja stanu. Dzięki temu uzyskana data zostaje wyświetlona wewnątrz przycisku odpowiedzialnego za jej wybór.

7.9.3 Dodawanie lokalizacji

Dodawanie lokalizacji do wspomnienia, podobnie jak dodawanie zdjęcia, wykorzystuje przekazywanie funkcji w dół drzewa widgetów. **NewLocationWidget** (**Rysunek 63**) to **StatefulWidget** reprezentujący wybraną przez użytkownika lokalizację, który przyjmuje jako parametr funkcję **onPickedLocation**. Ta przypisuje odpowiedniej zmiennej (powiązanej z ekranem dodawania wspomnienia) informacje na temat wybranej lokalizacji.

```
NewLocationWidget(  
  onPickedLocation: (locationInfo) {  
    _chosenLocation = locationInfo;  
  },  
), // NewLocationWidget
```

Rysunek 63

Przykład użycia widgetu NewLocationWidget

Kliknięcie **NewLocationWidget** powoduje wywołanie funkcji przedstawionej na **Rysunku 64**.

```
Future<void> _getLocation() async {  
  final GeoPoint? coordinates =  
    await Navigator.of(context).push(MaterialPageRoute(  
      builder: (context) => const BaseMapScreen(isSelecting: true),  
    )); // MaterialPageRoute  
  if (coordinates == null) return;  
  final address = await _getAddress(coordinates);  
  if (address == null) return;  
  if (!mounted) return;  
  setState(() {  
    _imageUrl = getStaticMap(coordinates);  
  });  
  widget  
    .onPickedLocation(MapData(coordinates: coordinates, address: address));  
}
```

Rysunek 64

Funkcja odpowiedzialna za uzyskanie lokalizacji

Początkowo funkcja przenosi użytkownika na ekran mapy w trybie wyboru lokalizacji. Zwracana przy zamykaniu tego ekranu informacja o lokalizacji jest przechwytywana. Następnie, gdy omawiana wartość nie jest **null** (lokalizacja została wybrana i odpowiednio zapisana), podejmowana jest próba uzyskania adresu wspomnienia z wykorzystaniem Geocoding API. Odpowiedzialną za to funkcję przedstawia **Rysunek 65**. Jej działanie opiera się na prostym żądaniu **HTTP**. Jeżeli w jego czasie wystąpi błąd, to użytkownik jest o tym informowany odpowiednim oknem dialogowym. Nieudana próba uzyskania adresu (zwrócenie wartości **null**) powoduje także przerwanie działania funkcji odpowiedzialnej za uzyskanie lokalizacji. Zabezpiecza to aplikację przed nieprawidłowym działaniem, które mogłoby wynikać z braku oczekiwanej wartości.

```

Future<String?> _getAddress(GeoPoint coordinates) async {
  final status = await checkConnection();
  if (!status) {
    if (!mounted) return null;
    await showInfoPopup(context, noConnection);
    return null;
  }
  final lat = coordinates.latitude;
  final lng = coordinates.longitude;

  try {
    final url = Uri.parse(
      'https://maps.googleapis.com/maps/api/geocode/json?latlng=$lat,$lng&language=pl&key=$apiKey');
    final response = await http.get(url);
    if (response.statusCode == 200) {
      final resData = json.decode(response.body);
      final String address = resData['results'][0]['formatted_address'];
      return address;
    }
  } on SocketException {
    if (!mounted) return null;
    await showInfoPopup(context, noConnection);
    return null;
  } catch (_) {
    if (!mounted) return null;
    await showInfoPopup(context, unknownError);
    return null;
  }
  return null;
}

```

Rysunek 65

Funkcja odpowiedzialna za próbę uzyskania adresu wspomnienia z wykorzystaniem Geocoding API

Po uzyskaniu adresu aktualizowany jest stan **NewLocationWidget**. Dzięki temu zawartością tego widgetu staje się statyczny obraz mapy, który reprezentuje wybraną lokalizację (wykorzystanie Maps Static API). **Rysunek 66** przedstawia funkcję, która zwraca adres **URL** statycznego obrazu mapy przy wykorzystaniu dostarczonych współrzędnych geograficznych.

```

String getStaticMap(GeoPoint location) {
  final lat = location.latitude;
  final lng = location.longitude;

  return '''https://maps.googleapis.com/maps/api/staticmap?
  center=$lat,$lng&zoom=17&size=600x600&maptype=roadmap
  &markers=color:red%7Clabel:""%7C$lat,$lng&key=$apiKey''';
}

```

Rysunek 66

Funkcja zwracająca adres URL statycznego obrazu mapy

Na koniec wywoływana jest przekazana jako parametr funkcja **onPickedLocation**, która aktualizuje (powiązaną z ekranem dodawania wspomnienia) zmienną odpowiedzialną za przechowywanie danych o wybranej lokalizacji. Jest to zmienna typu **MapData**, zdefiniowanego specjalnie na potrzeby aplikacji. Reprezentuje model danych przedstawiony na **Rysunku 67**, związany z przechowywaniem informacji o lokalizacji, obejmujących współrzędne geograficzne, a także odpowiadający im adres.

```
class MapData {
    const MapData({
        required this.coordinates,
        required this.address,
    });
    final GeoPoint coordinates;
    final String address;
}
```

Rysunek 67
Definicja modelu danych MapData

7.10 Wielokrotne wykorzystanie ekranu mapy

Ekran mapy w różnej formie jest kilkakrotnie wykorzystywany w aplikacji. W związku z tym, mając na celu unikanie powtórzeń kodu, został wydzielony do osobnego **StatefulWidget**, jakim jest **BaseMapScreen**. Zabieg ten umożliwił wykorzystanie parametrów konstruktora, co pozwoliło na dostosowywanie interaktywnej mapy do konkretnych potrzeb oraz zagnieżdżanie jej wewnątrz innych widgetów.

```
const BaseMapScreen({
    this.isSelecting = false,
    this.initialPosition = const LatLng(51.77689791254236, 19.489274125911784),
    this.markers = const {},
    this.isMemoriesMap = false,
    super.key,
});
```

Rysunek 68
Konstruktor widgetu BaseMapScreen

Zgodnie z, przedstawionym na **Rysunku 68**, konstruktorem widgetu **BaseMapScreen**:

1. **isSelecting** – użycie mapy w trybie wyboru lokalizacji. Domyślnie fałsz.
2. **initialPosition** – początkowo wyświetlana lokalizacja. Domyślnie odpowiada lokalizacji **Wydziału Fizyki i Informatyki Stosowanej Uniwersytetu Łódzkiego**.
3. **markers** – zbiór wyświetlanych na mapie znaczników. Domyślnie pusty.
4. **isMemoriesMap** – użycie mapy w trybie mapy wspomnień. Domyślnie fałsz.

To, jakie znaczniki będą aktualnie wyświetlane na mapie, definiuje funkcja (**getter**) **_markersList**, przedstawiona na **Rysunku 69**.

```

Set<Marker> get _markersList {
  if (widget.isSelecting && _pickedPosition != null) {
    return {
      Marker(
        markerId: const MarkerId('picked_location'),
        position: _pickedPosition!,
        infoWindow: const InfoWindow(title: yourLocation)),
    );
  }
  if (!widget.isSelecting) return widget.markers;
  return {};
}

```

Rysunek 69
Funkcja definiująca wyświetlane na mapie znaczniki

Zgodnie ze strukturą **_markersList**:

1. Jeżeli ekran mapy jest w trybie wyboru lokalizacji i lokalizacja została wybrana, to znacznik będzie znajdował się w tej lokalizacji.
2. Jeżeli ekran mapy nie jest w trybie wyboru lokalizacji, to znaczniki będą odpowiadały tym, które przekazano jako parametr konstruktora.
3. Jeżeli żaden z wcześniejszych warunków nie został spełniony, to nie będą wyświetlane żadne znaczniki.

7.10.1 Implementacja ekranu mapy w trybie wyboru lokalizacji

Użytkownik jest przenoszony na ten ekran chcąc dodać lokalizację do zapisywanego wspomnienia. Parametr **isSelecting** przyjmuje wartość **true**, a pozostałe parametry przyjmują swoje wartości domyślne.

Lokalizacja może zostać wybrana na dwa sposoby:

1. **Ręczny wybór lokalizacji przez użytkownika** – obsługiwany przez funkcję przedstawioną na **Rysunku 70**. Aktualizuje stan widgetu, przypisując wybraną lokalizację do odpowiedniej zmiennej. Powoduje to przebudowanie mapy, skutkujące wyświetleniem znacznika reprezentującego wybraną lokalizację.

```

(markerPosition) {
  setState(() {
    _pickedPosition = markerPosition;
  });
}

```

Rysunek 70
Funkcja obsługująca ręczny wybór lokalizacji

2. **Pobranie aktualnej lokalizacji użytkownika** – realizowane przez funkcję przedstawioną na **Rysunku 71**. Poprzez użycie odpowiedniego przycisku podejmowana jest próba uzyskania aktualnej lokalizacji użytkownika. Najpierw sprawdzane jest to, czy aplikacja posiada właściwe uprawnienia oraz ewentualnie

podejmowana jest próba ich uzyskania. W przypadku prawidłowych uprawnień rozpoczyna się proces (symbolizowany przez odpowiedni indykator) uzyskiwania aktualnej lokalizacji użytkownika. Po zakończeniu procesu (poprzez aktualizację stanu) do odpowiedniej zmiennej pomocniczej przypisywane są uzyskane współrzędne geograficzne. Indykator ładowania znika, a mapa przenosi się do uzyskanej lokalizacji i wyświetla w tym miejscu odpowiedni znacznik.

```
Future<void> _getCurrentLocation() async {
  final Position position;
  final check = await _checkPermissions();
  if (!check) return;
  setState(() {
    _isGettingCurrentLocation = true;
  });
  try {
    position = await Geolocator.getCurrentPosition();
  } catch (_) {
    if (!mounted) return;
    setState(() {
      _isGettingCurrentLocation = false;
    });
    await showInfoPopup(context, locationError);
    return;
  }
  final lat = position.latitude;
  final lng = position.longitude;
  if (!mounted) return;
  setState(() {
    _isGettingCurrentLocation = false;
    _pickedPosition = LatLng(lat, lng);
  });
  _moveCamera(_pickedPosition!, zoom);
}
```

Rysunek 71

Funkcja pobierająca aktualną lokalizację użytkownika

Jeżeli użytkownik chce zapisać wybraną lokalizację, to musi tego dokonać ręcznie. W przeciwnym wypadku opuszczenie ekranu automatycznie zwraca wartość **null**. Ta sama wartość (**null**) zostanie zwrócona również w sytuacji, w której użytkownik zapisze lokalizację bez jej wcześniejszego wybrania. Naturalnie, zapisanie prawidłowo wybranej lokalizacji zwraca właściwe informacje na jej temat. Zwracaną wartość obsługuje funkcja przedstawiona na **Rysunku 64**. **Rysunek 72** przedstawia funkcję odpowiedzialną za zapisanie wybranej lokalizacji. Zwraca ona odpowiednią wartość, zamykając przy tym ekran mapy.

```

void _savePosition() {
    _pickedPosition == null
        ? Navigator.pop(context)
        : Navigator.pop(
            context,
            GeoPoint(
                _pickedPosition!.latitude,
                _pickedPosition!.longitude,
            ),
        );
}

```

Rysunek 72

Funkcja odpowiedzialna za zapisanie wybranej lokalizacji

7.10.2 Implementacja ekranu mapy wspomnień

Nadanie parametrowi **isMemoriesMap** wartości **true** powoduje wyświetlenie ekranu mapy w trybie mapy wspomnień. Przekazywane są także **markers** i **initialPosition**, a **isSelecting** przyjmuje wartość domyślną. Ekran mapy w wspomnianym trybie może być zagnieżdżony w widżecie **MemoriesMap**.

Znaczniki przekazywane jako parametr **markers** tworzone są w oparciu o odczytane z Cloud Firestore informacje o wspomnieniach. **Rysunek 73** przedstawia funkcję (**getter**) odpowiedzialną za uzyskanie znaczników na podstawie danych z Cloud Firestore. **Rysunek 74** przedstawia funkcję zwracającą konkretny znacznik.

```

Future<Set<Marker>> get _markersList async {
    final Set<Marker> markers = {};
    final user = FirebaseAuth.instance.currentUser!;
    final docs = await FirebaseFirestore.instance
        .collection(firebaseDataKeys['memories_by_user']!)
        .doc(user.uid)
        .collection(firebaseDataKeys['memories']!)
        .orderBy(firebaseDataKeys['uploadTimeStamp']!, descending: true)
        .get();
    for (final doc in docs.docs) {
        final data = doc.data();
        final marker = _createMarker(
            doc.id,
            data,
        );
        markers.add(marker);
    }
    return markers;
}

```

Rysunek 73

Funkcja odpowiedzialna za uzyskanie znaczników wspomnień na podstawie danych z Cloud Firestore


```

Marker _createMarker(
  String id,
  Map<String, dynamic> data,
) {
  return Marker(
    markerId: MarkerId(id),
    infowindow: Infowindow(
      onTap: () {
        Navigator.of(context).push(
          MaterialPageRoute(
            builder: (context) => MemoryDetails(data: data, id: id),
          ), // MaterialPageRoute
        );
      },
      title: data[firebaseDataKeys["memoryDate"]!],
      snippet: data[firebaseDataKeys["title"]!],
    ), // Infowindow
    position: LatLng(
      data[firebaseDataKeys["geopoint"]!].latitude,
      data[firebaseDataKeys["geopoint"]!].longitude,
    ), // LatLng
  ); // Marker
}

```

Rysunek 74
Funkcja tworząca konkretny znacznik

Wspomnienia są odczytywane z Cloud Firestore w kolejności od najnowszego pod kątem czasu zapisania w aplikacji (pole **uploadTimeStamp**). Następnie na podstawie tych danych tworzone są poszczególne znaczniki. Kliknięcie w znacznik ukazuje odpowiednią etykietę, która zawiera informacje o tytule i dacie (pole **memoryDate**) wspomnienia reprezentowanego przez konkretny znacznik. Wybranie tej etykiety przenosi użytkownika na ekran szczegółów danego wspomnienia.

```

@override
Widget build(BuildContext context) {
  return FutureBuilder(
    future: _markers,
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(
          child: CircularProgressIndicator(),
        ); // Center
      }
      if (snapshot.hasError) {
        return const Infotext(text: unknownError);
      }
      if (!snapshot.hasData || snapshot.data!.isEmpty) {
        return const Infotext(text: noMemories);
      }

      return BaseMapScreen(
        isMemoriesMap: true,
        markers: snapshot.data!,
        initialPosition: snapshot.data!.first.position,
      ); // BaseMapScreen
    },
  ); // FutureBuilder
}

```

Rysunek 75
Metoda build widgetu MemoriesMap

Zgodnie z metodą **build**, przedstawioną na **Rysunku 75**, uzyskane znaczniki definiują to, jaka będzie zawartość widgetu **MemoriesMap**, a tym samym ekranu mapy wspomnień:

1. **Indykator ładowania** – występuje podczas uzyskiwania znaczników na podstawie danych z Cloud Firestore.
2. **Informacja o błędzie** – wykorzystuje widget **InfoText**. Występuje, gdy w czasie uzyskiwania znaczników nastąpił błąd.
3. **Informacja o braku wspomnień** – wykorzystuje widget **InfoText**. Występuje, gdy użytkownik nie posiada żadnych zapisanych wspomnień.
4. **Właściwa mapa wspomnień** – występuje w pozostałych przypadkach. Utworzone znaczniki są przekazywane jako wartość parametru **markers**, a **initialPosition** odpowiada pozycji pierwszego znacznika na liście.

7.10.3 Implementacja ekranu mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia

Ten ekran jest wywoływany poprzez przytrzymanie miniatury lokalizacji na ekranie szczegółów wspomnienia. Parametry **isMemoriesMap** i **isSelecting** przyjmują wartości domyślne. Podobnie jak w trybie mapy wspomnień, przekazywane są **markers** (w tym wypadku pojedynczy znacznik) i **initialPosition**. Wartości te reprezentują przeglądane wspomnienie i odpowiadają danym na jego temat. Funkcję odpowiedzialną za wyświetlenie omawianego ekranu przedstawia **Rysunek 76**. Wybranie widocznego na mapie znacznika powoduje wyświetlenie etykiety z nazwą sprawdzanego wspomnienia.

```
void _showMap() {
  final position =
    LatLng(widget.location.latitude, widget.location.longitude);
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => BaseMapScreen(
        initialPosition: position,
        markers: {
          Marker(
            markerId: MarkerId(widget.title),
            infoWindow: InfoWindow(
              title: widget.title,
            ), // InfoWindow
            position: position), // Marker
        },
      ), // BaseMapScreen
    ), // MaterialPageRoute
  );
}
```

Rysunek 76

Funkcja odpowiedzialna za wyświetlenie ekranu mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia

Rozdział 8

Problemy w Implementacji

8.1 Implementacja mapy

8.1.1 Brak ładowania mapy

Jednym z głównych problemów zaobserwowanych w trakcie implementacji mapy był ten związany z niepoprawnym jej ładowaniem. Błąd ten nie pojawiał się zawsze, dlatego stosunkowo ciężko było go wychwycić. Przejawiał się w tym, że mapa, pomimo prawidłowej inicjalizacji (co potwierdzały uzyskiwane dzięki funkcji **print** informacje), nie wyświetlała się prawidłowo. Aby wymusić prawidłowe działanie, należało dokonać jakiegokolwiek interakcji z mapą (np. wykonać gest przeciągnięcia). W związku z tym bezpośrednio podczas tworzenia mapy i inicjalizowania jej kontrolera wywoływana jest funkcja, która zwiększa przybliżenie kamery o 1 (**Rysunek 77**). Podejście to zdaje się rozwiązywać problem, który występował w wersji aplikacji na system Android.

```
void _onMapCreated(GoogleMapController controller) {  
    _controller = controller;  
    _moveCamera(widget.initialPosition, zoom + 1);  
}
```

Rysunek 77

Rozwiązanie problemu z nieprawidłowym ładowaniem się mapy

8.1.2 Niewykrywanie włączonej lokalizacji

Problem występował w sytuacji, w której podejmowana była próba uzyskania bieżącej lokalizacji użytkownika, a usługi lokalizacyjne urządzenia były wyłączone przed rozpoczęciem tego procesu. W pewnych przypadkach aplikacja nie rozpoczynała pobierania lokalizacji. Problematycznym okazał się pakiet **location** [48], wykorzystywany do pobierania lokalizacji. Zdecydowano więc o wykorzystaniu innego pakietu (**geolocator** [49]) do uzyskania aktualnej lokalizacji użytkownika. Poprzednie rozwiązanie pozostaje jednak cały czas wykorzystywane do sprawdzania uprawnień aplikacji dotyczących dostępu do lokalizacji oraz tego, czy usługi lokalizacyjne urządzenia są włączone. Naturalnie problem ten występował w wersji aplikacji na system Android.

8.1.3 Brak trybu ciemnego dla interaktywnej mapy

Zaimplementowana w aplikacji interaktywna mapa domyślnie nie była kompatybilna z ciemnym motywem aplikacji. W celu implementacji tej funkcjonalności niezbędnym okazało się wykorzystanie dostarczonego przez Google narzędzia do tworzenia stylu

mapy [50]. Wygenerowany z jego pomocą styl został odpowiednio zaimplementowany w aplikacji, co pozwoliło na dodanie trybu ciemnego do interaktywnej mapy.

8.2 Konieczność rozdzielenia logiki pobierania zdjęć między platformami

Jeden z najpopularniejszych pakietów do lokalizowania katalogów w systemie plików urządzenia (***path_provider*** [51]) okazał się niekompatybilny z platformą webową. W związku z tym musiała zostać podjęta decyzja o przygotowaniu osobnych rozwiązań, umożliwiających pobieranie powiązanych ze wspomnieniem zdjęć na różnych platformach.

W przypadku wersji na system Android uzyskiwana jest ścieżka do katalogu, w którym mogą być przechowywane pliki pobierane przez aplikację. Następnie wybrane zdjęcie jest do niego pobierane [52], a na końcu zapisywane w galerii urządzenia [53] i usuwane z pierwotnego katalogu. Funkcję odpowiedzialną za realizację tej logiki przedstawia **Rysunek 78**.

```
Future<void> _downloadAndroid() async {
  final directory = await getDownloadsDirectory();
  final path = '${directory!.path}/${widget.id}.jpg';
  try {
    await Dio().download(widget.data[firebaseDataKeys['imageUrl']], path);
  } on DioException catch (_) {
    if (!mounted) return;
    await showInfoPopup(context, photoFailed);
    setState(() {
      _isDownloading = false;
    });
    return;
  } catch (_) {
    if (!mounted) return;
    await showInfoPopup(context, unknownError);
    setState(() {
      _isDownloading = false;
    });
    return;
  }
  await Gal.putImage(path);
  final file = File(path);
  await file.delete();
  if (!mounted) return;
  showToast(imageSaved, context);
}
```

Rysunek 78

Funkcja odpowiedzialna za pobieranie zdjęcia do pamięci urządzenia w wersji aplikacji na system Android

Paradoksalnie, dla wersji webowej proces ten jest znacznie prostszy, gdyż wykorzystuje gotowe rozwiązanie (pakiet ***image_downloader_web*** [54]) dostępne na platformie pub.dev. Implementację tego rozwiązania przedstawia **Rysunek 79**.

```

Future<void> _downloadWeb() async {
  try {
    await WebImageDownloader.downloadImageFromWeb(
      widget.data[firebaseDataKeys['imageUrl']!],
      imageType: ImageType.jpeg,
      name: widget.id,
    );
  } catch (e) {
    if (!mounted) return;
    showInfoPopup(context, photoFailed);
    return;
  }
}

```

Rysunek 79

Funkcja odpowiedzialna za pobieranie zdjęcia do pamięci urządzenia w webowej wersji aplikacji

Nazwa pobieranego pliku odpowiada nazwie dokumentu w Cloud Firestore, który reprezentuje dane wspomnienie.

8.3 Ukrycie klucza API Map Google dla wersji webowej w repozytorium

Opisywany punkt nie jest do końca problemem w implementacji aplikacji, a problemem związanym z procesem przechowywania jej plików źródłowych.

W celu dodania Map Google do webowej wersji aplikacji, wymagany do ich działania klucz API musiałby zostać dodany bezpośrednio do zawartości pliku **index.html** z katalogu web [55]. To mogłoby prowadzić do potencjalnych naruszeń bezpieczeństwa przy przechowywaniu kodu źródłowego aplikacji w zdalnym repozytorium. W związku z tym zaimplementowano mechanizm dodający klucz API Map Google do pliku **index.html** dopiero na etapie działania kodu. Odpowiedzialną za to funkcję przedstawia **Rysunek 80**. Ta (wywoływana w funkcji **main**) musiała zostać wydzielona do osobnego pliku, którego importowanie musi być wstrzymywane przy pracy z platformami innymi niż web z powodu braku kompatybilności. Co więcej, wykorzystywany w aplikacji klucz API Map Google jest wczytywany z pliku **.env**. Pakiet **flutter_dotenv** [56] umożliwia odczytanie zawartości tego pliku na etapie uruchamiania aplikacji. Uzyskane dane są dostępne w każdym miejscu aplikacji.

```

void addAPIkeyWeb() {
  final script = HTMLScriptElement();
  script.src = "https://maps.googleapis.com/maps/api/js?key=$apiKey";
  document.head!.append(script);
}

```

Rysunek 80

Funkcja dodająca klucz API Map Google do pliku index.html

Rozdział 9

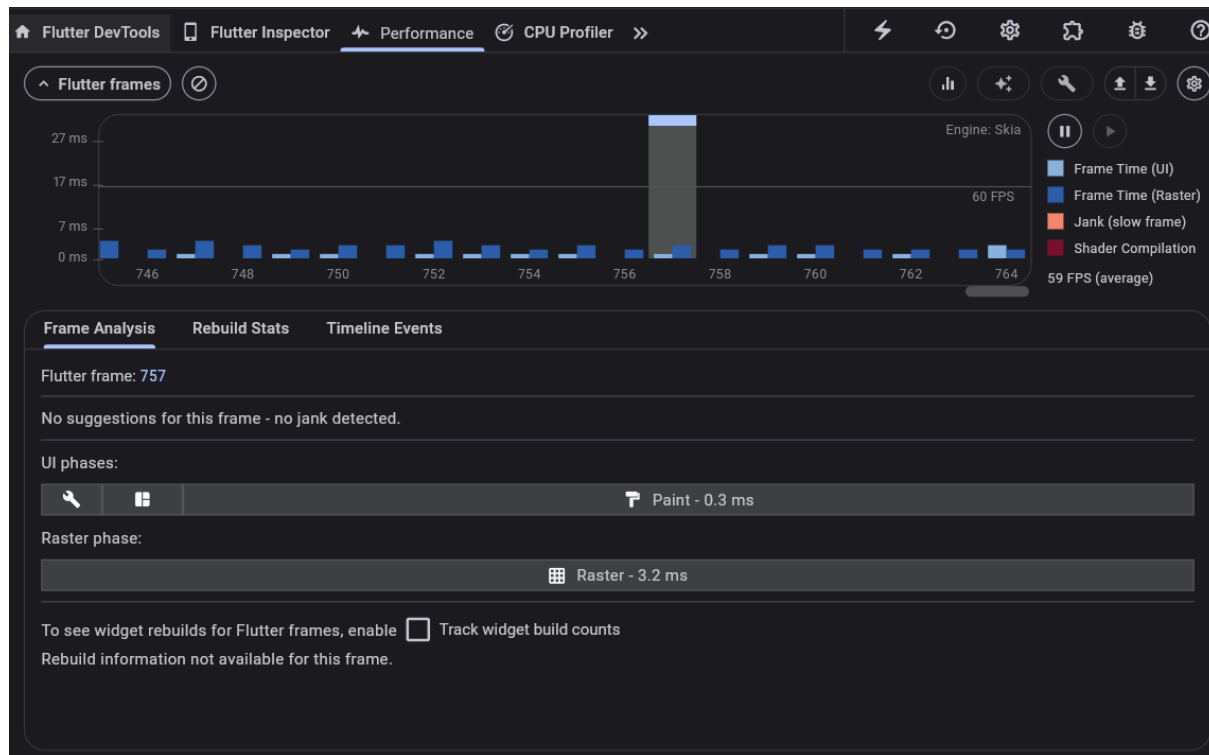
Testy i Diagnostyka

9.1 Flutter DevTools

Tworząc aplikację z użyciem Flutter, można skorzystać z niezwykle zaawansowanego zestawu narzędzi, jakim jest **Flutter DevTools** [57]. Pozwala on na dogłębną diagnostykę i debugowanie tworzonej aplikacji.

Do najważniejszych narzędzi, zawartych we Flutter DevTols należą:

1. **Flutter Inspector** – umożliwia podgląd wygenerowanego drzewa widgetów, a także dokładną analizę interfejsu pod kątem wymiarów i rozłożenia jego elementów.
2. **Debugger**
3. **Performance view** – pozwala na dokładne analizowanie generowanej liczby klatek na sekundę (**FPS**) wraz ze zdarzeniami wpływającymi na płynność działania aplikacji. Przykładowy widok tego narzędzia przedstawia **Rysunek 81**. W przypadku wersji webowej funkcja ta zawiera odwołanie do **Chrome DevTools** [58] [59].



Rysunek 81
Performance View we Flutter DevTools

Oraz niedostępne dla webowej wersji aplikacji:

1. **CPU Profiler** – na podstawie zarejestrowanych próbek umożliwia dokładną analizę obciążenia procesora w czasie działania aplikacji.
2. **Memory view** – umożliwia nadzorowanie w czasie rzeczywistym pamięci **RAM** wykorzystywanej przez aplikację.
3. **Network view** – umożliwia nadzorowanie ruchu sieciowego powiązanego z aplikacją, takiego jak żądania **HTTP**.

Łatwo zauważyć, że większość narzędzi powiązana jest z analizowaniem szeroko pojętej wydajności i stabilności aplikacji. Pozwala to na wyłapywanie potencjalnych problemów z aplikacją już na wstępnym etapie prac nad nią, dzięki czemu łatwiej dostarczyć dopracowany produkt końcowy.

9.2 Testowanie aplikacji

W czasie procesu deweloperskiego aplikacja była uruchamiana w trybie **debug** (umożliwia dostęp do zaawansowanych narzędzi deweloperskich). Właściwe testy aplikacji przeprowadzono jednak z wykorzystaniem aplikacji skompilowanej w trybie **release** (oferującym najlepszą optymalizację i wydajność). Wśród testerów znalazły się osoby w różnym wieku (**10 – 50 lat**), co pozwoliło na zebranie szerokiego spektrum opinii na temat aplikacji.

Testowane były przede wszystkim:

- **Wydajność**
- **Stabilność**
- **Poprawność działania zaimplementowanych funkcjonalności**
- **Czytelność interfejsu**

9.2.1 Wersja na system Android

Testy wersji aplikacji na system Android przeprowadzono z wykorzystaniem:

1. **Nothing Phone 2a** – Android 15
2. **Samsung Galaxy A34** – Android 14
3. **Samsung Galaxy Tab A9+** – Android 14
4. **Realme Gt Master Edition** – Android 13
5. **Samsung Galaxy M21** – Android 12
6. **Honor 8x** – Android 10

Są to urządzenia o różnej konfiguracji sprzętowej, w różnym wieku i wykorzystujące różne wersje systemu Android. Przeprowadzone na tak szerokiej gamie urządzeń testy pozwoliły uzyskać szersze spektrum wyników.

9.2.2 Wersja webowa

Testy wersji webowej były przeprowadzane z wykorzystaniem lokalnego serwera oraz przeglądarek:

1. **Microsoft Edge** – wersja 131.0.2903.70
2. **Brave** – wersja 1.73.97
3. **Opera** – wersja 115.0.5322.77
4. **Mozilla Firefox** – wersja 133.0
5. **Google Chrome** – wersja 131.0.6778.134

Podobnie jak w przypadku wersji na system Android, również tym razem wykorzystano możliwie szeroką gamę platform testowych.

9.2.3 Wyniki testów

Przeprowadzone testy okazały się być niezwykle efektywne. Potwierdzono stabilność i poprawność działania aplikacji w różnych warunkach. Chwalone były czytelność interfejsu oraz jego responsywność. Użytkownicy docenili również intuicyjność i prostotę obsługi aplikacji. Dzięki wsparciu testerów wszelkie drobne problemy i niedogodności, które napotkano w trakcie trwania testów, zostały skutecznie zdiagnozowane, a następnie rozwiązane w sposób zgodny z uzyskanymi opiniami i sugestiami.

9.3 System automatycznego wykrywania błędów

Choć wyniki testów są niezwykle optymistyczne, to należy pamiętać, że niemal niemożliwym jest stworzenie oprogramowania pozbawionego jakichkolwiek błędów, działającego tak samo w każdych warunkach. Choć w kodzie aplikacji podjęta została próba obsłużenia jak największej ilości błędów, to zaimplementowano również dodatkowe mechanizmy mające pomóc w diagnozie potencjalnych problemów aplikacji.

Podstawę tego systemu stanowi opisywany w **Rozdziale 3.1.5** Firebase Crashlytics. Tak jak wspomniano, narzędzie to nie obsługuje jednak wykrywania błędów w webowej wersji aplikacji. W związku z tym zaimplementowano dodatkowy system automatycznego zgłaszania błędów napotkanych w aplikacji. W przypadku błędu aplikacja automatycznie uzupełnia i przesyła odpowiedni formularz Google Forms (**Rysunek 82**). Przesyłany formularz zawiera informacje na temat szczegółów błędu, wersji aplikacji oraz tego, czy błąd wystąpił w webowej wersji aplikacji.


```
await http.post(  
  Uri.parse(autoReportUrl),  
  body: {  
    'entry.1191287436': error,  
    'entry.1479404169': version,  
    'entry.517717971': kIsWeb.toString(),  
  },  
);
```

Rysunek 82

Automatyczne uzupełnienie i wysłanie formularza z informacjami o błędzie napotkanym w aplikacji

Zastosowanie takiego dwustopniowego mechanizmu oraz udostępnienie możliwości ręcznego zgłaszania błędów (opcja w szufladzie użytkownika) daje nadzieję na to, że wszelkie błędy, które mogą wystąpić w przyszłości, uda się łatwo zdiagnozować i rozwiązać.

Podsumowanie

Przedmiotem pracy było stworzenie multiplatformowej aplikacji umożliwiającej gromadzenie wspomnień. Aplikacja miała powstać z myślą o systemie Android oraz w wersji webowej. Miała opierać się na przechowywaniu wspomnień w chmurze, przy wykorzystaniu Firebase, dzięki czemu użytkownik posiadałby dostęp do wspomnień z różnych urządzeń.

Ostatecznie udało się spełnić postawione na początku wymagania oraz zrealizować zadane cele. System logowania i rejestracji działa prawidłowo, a wspomnienia są na bieżąco synchronizowane i łatwe w dodawaniu. Stworzono czytelny i responsywny interfejs, który pozostaje jednolity na różnych platformach.

Wykorzystanie technologii Flutter okazało się doskonale sprawdzać w przypadku tworzenia tego typu aplikacji. Łatwo zauważyć, że jest to rozwiązanie stawiające na komfort programisty oraz dostarczenie wysokiej jakości rozwiązań. Pomaga tworzyć dopracowane interfejsy, dzięki czemu nawet osoby, dla których nie jest to tradycyjne zajęcie, będą w stanie stworzyć coś z niczego. Nieocenione okazały się także pakiety dostępne na platformie pub.dev. Jak wszystko jednak, technologia ta posiada swoje problemy. Głównym zaobserwowanym problemem jest zdecydowanie specyfika tworzenia aplikacji na platformę webową. Mniejsza ilość pakietów kompatybilnych z platformą webową sprawia, że tworzenie aplikacji trzeba lepiej przemyśleć pod tym kątem. Optymizmem nie napawają również wspomniane przez twórców problemy z pozycjonowaniem strony. Mimo wszystko opisywana technologia nieustannie się rozwija i można z nią wiązać duże nadzieje na przyszłość.

Aplikacja została stworzona w sposób możliwie modułarny, aby jej potencjalny rozwój w przyszłości przebiegał względnie bezproblemowo. Podstawowym elementem, który warto byłoby zaimplementować w przyszłości jest opcja udostępniania wspomnień innym użytkownikom np. poprzez tworzenie odpowiednich pokoi lub grup. Sensownym krokiem w dalszym rozwoju jest także stopniowe dodawanie kolejnych wersji językowych. Dodatkowo, by jeszcze lepiej wykorzystać zdolności technologii Flutter do tworzenia aplikacji multiplatformowych, kolejnym krokiem w rozwoju powinno być stworzenie wersji aplikacji przeznaczonej na system iOS.

Bibliografia

- [1] Oberlo. *How Many People Have Smartphones? (2014–2029)*
<https://www.oberlo.com/statistics/how-many-people-have-smartphones>
(dostęp: 10 lutego 2025)
- [2] Nikoniarze.pl. *EXIF bez tajemnic*
<https://nikoniarze.pl/exif-bez-tajemnic>
(dostęp: 10 lutego 2025)
- [3] Google Play. *Zdjęcia Google*
<https://play.google.com/store/apps/details?id=com.google.android.apps.photos&hl=pl>
(dostęp: 10 lutego 2025)
- [4] Google Play. *Journey: Diary, Journal, Notes*
<https://play.google.com/store/apps/details?id=com.journey.app>
(dostęp: 10 lutego 2025)
- [5] Journey Cloud help center. *In-app Purchase Comparison*
<https://help.journey.cloud/en/article/in-app-purchase-comparison-1tqlxue/>
(dostęp: 10 lutego 2025)
- [6] Visual Studio Code
<https://code.visualstudio.com/>
(dostęp: 10 lutego 2025)
- [7] Android Studio
<https://developer.android.com/studio>
(dostęp: 10 lutego 2025)
- [8] Flutter documentation. *Start building Flutter Android apps on Windows – Configure Android development*
<https://docs.flutter.dev/get-started/install/windows/mobile#configure-android-development>
(dostęp: 10 lutego 2025)
- [9] Git
<https://git-scm.com>
(dostęp: 10 lutego 2025)
- [10] Google Maps Platform
<https://mapsplatform.google.com>
(dostęp: 10 lutego 2025)

- [11] Google Maps Platform Documentation. Maps SDK for Android
<https://developers.google.com/maps/documentation/android-sdk>
(dostęp: 10 lutego 2025)
- [12] Google Maps Platform Documentation. Maps JavaScript API
<https://developers.google.com/maps/documentation/javascript>
(dostęp: 10 lutego 2025)
- [13] Google Maps Platform Documentation. Maps Static API
<https://developers.google.com/maps/documentation/maps-static>
(dostęp: 10 lutego 2025)
- [14] Google Maps Platform Documentation. Geocoding API
<https://developers.google.com/maps/documentation/geocoding>
(dostęp: 10 lutego 2025)
- [15] Firebase
<https://firebase.google.com/>
(dostęp: 10 lutego 2025)
- [16] Firebase documentation. Cloud Firestore
<https://firebase.google.com/docs/firestore>
(dostęp: 10 lutego 2025)
- [17] Custer C. Cockroach Labs. What is UUID, and what is it used for?; 29 czerwca 2023
<https://www.cockroachlabs.com/blog/what-is-a-uuid>
(dostęp: 10 lutego 2025)
- [18] Firebase documentation. Cloud Storage for Firebase
<https://firebase.google.com/docs/storage>
(dostęp: 10 lutego 2025)
- [19] Firebase documentation. Firebase Authentication
<https://firebase.google.com/docs/auth>
(dostęp: 10 lutego 2025)
- [20] Firebase documentation. Firebase Crashlytics
<https://firebase.google.com/docs/crashlytics>
(dostęp: 10 lutego 2025)
- [21] Google Forms
<https://www.google.com/forms/about/>
(dostęp: 10 lutego 2025)
- [22] Gmail
<https://workspace.google.com/intl/pl/gmail>
(dostęp: 10 lutego 2025)

- [23] *Flutter documentation. Flutter SDK overview*
<https://docs.flutter.dev/tools/sdk>
(dostęp: 10 lutego 2025)
- [24] *Dart documentation. Introduction to Dart*
<https://dart.dev/language>
(dostęp: 10 lutego 2025)
- [25] *Flutter documentation. Flutter architectural overview*
<https://docs.flutter.dev/resources/architectural-overview>
(dostęp: 10 lutego 2025)
- [26] *Flutter documentation. Flutter architectural overview - Flutter web support*
<https://docs.flutter.dev/resources/architectural-overview#flutter-web-support>
(dostęp: 10 lutego 2025)
- [27] *Flutter documentation. Building user interfaces with Flutter*
<https://docs.flutter.dev/ui>
(dostęp: 10 lutego 2025)
- [28] *Flutter documentation. Material component widgets*
<https://docs.flutter.dev/ui/widgets/material>
(dostęp: 10 lutego 2025)
- [29] *Google. Material Design 3*
<https://m3.material.io/>
(dostęp: 10 lutego 2025)
- [30] *Flutter documentation. Cupertino widgets*
<https://docs.flutter.dev/ui/widgets/cupertino>
(dostęp: 10 lutego 2025)
- [31] *Platforma pub.dev*
<https://pub.dev/>
(dostęp: 10 lutego 2025)
- [32] *Google Trends. Porównanie zainteresowania React Native, Flutter, Xamarin w okresie od 10.01.2015 do 10.01.2025*
<https://trends.google.com/trends/explore?cat=31&date=2015-01-10%202025-01-10&q=React%20Native,Flutter,Xamarin>
(dostęp: 10 lutego 2025)
- [33] *Flutter. Showcase - Flutter apps in production*
<https://flutter.dev/showcase>
(dostęp: 10 lutego 2025)

- [34] *Flutter documentation. Platform integration - Web FAQ*
<https://docs.flutter.dev/platform-integration/web/faq>
(dostęp: 10 lutego 2025)
- [35] *GitHub. Flutter Roadmap - Web Platform*
<https://github.com/flutter/flutter/blob/master/docs/roadmap/Roadmap.md#web-platform>
(dostęp: 10 lutego 2025)
- [36] *Material Theme Builder*
<https://material-foundation.github.io/material-theme-builder/>
(dostęp: 10 lutego 2025)
- [37] *Firebase Authentication documentation. Manage User Sessions*
<https://firebase.google.com/docs/auth/admin/manage-sessions>
(dostęp: 10 lutego 2025)
- [38] *Wenger J. Medium. Demystifying Firebase Auth Tokens; 5 listopada 2018*
<https://medium.com/@jwngr/demystifying-firebase-auth-tokens-e0c533ed330c>
(dostęp: 10 lutego 2025)
- [39] *Google Cloud - Identity Platform documentation. Enable or disable email enumeration protection*
<https://cloud.google.com/identity-platform/docs/admin/email-enumeration-protection>
(dostęp: 10 lutego 2025)
- [40] *Flutter API reference documentation. Widgets library - PageView class*
<https://api.flutter.dev/flutter/widgets/PageView-class.html>
(dostęp: 10 lutego 2025)
- [41] *Flutter API reference documentation. Widgets library – StreamBuilder class*
<https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html>
(dostęp: 10 lutego 2025)
- [42] *pub.dev. cached_network_image*
https://pub.dev/packages/cached_network_image
(dostęp: 10 lutego 2025)
- [43] *pub.dev. flutter_riverpod*
https://pub.dev/packages/flutter_riverpod
(dostęp: 10 lutego 2025)

- [44] *Riverpod documentation. All Providers – StateProvider*
https://riverpod.dev/docs/providers/state_provider
(dostęp: 10 lutego 2025)
- [45] *Flutter API reference documentation. Widgets library – SliverChildBuilderDelegate class*
<https://api.flutter.dev/flutter/widgets/SliverChildBuilderDelegate-class.html>
(dostęp: 10 lutego 2025)
- [46] *pub.dev. image_picker*
https://pub.dev/packages/image_picker
(dostęp: 10 lutego 2025)
- [47] *pub.dev. exif*
<https://pub.dev/packages/exif>
(dostęp: 10 lutego 2025)
- [48] *pub.dev. location*
<https://pub.dev/packages/location>
(dostęp: 10 lutego 2025)
- [49] *pub.dev. geolocator*
<https://pub.dev/packages/geolocator>
(dostęp: 10 lutego 2025)
- [50] *Styling Wizard: Google Maps APIs*
<https://mapstyle.withgoogle.com/>
(dostęp: 10 lutego 2025)
- [51] *pub.dev. path_provider*
https://pub.dev/packages/path_provider
(dostęp: 10 lutego 2025)
- [52] *pub.dev. dio*
<https://pub.dev/packages/dio>
(dostęp: 10 lutego 2025)
- [53] *pub.dev. gal*
<https://pub.dev/packages/gal>
(dostęp: 10 lutego 2025)
- [54] *pub.dev. image_downloader_web*
https://pub.dev/packages/image_downloader_web
(dostęp: 10 lutego 2025)

- [55] *pub.dev. google_maps_flutter_web*
https://pub.dev/packages/google_maps_flutter_web
(dostęp: 10 lutego 2025)
- [56] *pub.dev. flutter_dotenv*
https://pub.dev/packages/flutter_dotenv
(dostęp: 10 lutego 2025)
- [57] *Flutter documentation. Flutter and Dart DevTools*
<https://docs.flutter.dev/tools/devtools>
(dostęp: 10 lutego 2025)
- [58] *Flutter documentation. Debug performance for web apps*
<https://docs.flutter.dev/perf/web-performance>
(dostęp: 10 lutego 2025)
- [59] *Chrome for developers documentation. Chrome DevTools*
<https://developer.chrome.com/docs/devtools>
(dostęp: 10 lutego 2025)

Spis rysunków

Rysunek 1 Zdjęcia Google w wersji na system Android	8
Rysunek 2 Journey: Diary, Journal, Notes w wersji na system Android	9
Rysunek 3 Diagram przypadków użycia	10
Rysunek 4 Struktura bazy danych Cloud Firestore dla opisywanej aplikacji.....	15
Rysunek 5 Struktura dokumentu reprezentującego wspomnienie w Cloud Firestore ...	15
Rysunek 6 Struktura plików w Cloud Storage dla opisywanej aplikacji	16
Rysunek 7 Ogólna struktura plików w projekcie	17
Rysunek 8 Struktura katalogu lib.....	18
Rysunek 9 Struktura katalogu widgets	18
Rysunek 10 Porównanie zainteresowania wybranymi narzędziami do tworzenia aplikacji multiplatformowych w Google Trends.....	20
Rysunek 11 Porównanie ekranu logowania w wersji webowej (na górze) i na system Android (na dole)	23
Rysunek 12 Porównanie ekranu galerii wspomnień w wersji webowej (na górze) i na system Android (na dole)	24
Rysunek 13 Porównanie ekranu szczegółów wspomnienia w wersji webowej (na górze) i na system Android (na dole)	25
Rysunek 14 Zestawienie ciemnego (po lewej) i jasnego (po prawej) motywu aplikacji na przykładzie ekranu logowania w wersji na system Android.....	26
Rysunek 15 Ekran ładowania aplikacji.....	27
Rysunek 16 Ekran logowania	28
Rysunek 17 Ekran rejestracji	28
Rysunek 18 Okno resetowania hasła.....	29
Rysunek 19 Struktura ekranu zawartości.....	30
Rysunek 20 Struktura ekranu galerii wspomnień.....	31
Rysunek 21 Duże kafelki wspomnień	31
Rysunek 22 Struktura ekranu dodawania wspomnienia.....	32
Rysunek 23 Szuflada wyboru źródła zdjęcia.....	33
Rysunek 24 Okno wyboru daty.....	33
Rysunek 25 Reprezentacje wybranego zdjęcia i lokalizacji.....	34
Rysunek 26 Przykładowy ekran mapy wspomnień	35
Rysunek 27 Struktura szuflady użytkownika	35
Rysunek 28 Formularz resetowania hasła z poziomu szuflady użytkownika	36
Rysunek 29 Struktura ekranu mapy w trybie wyboru lokalizacji	37
Rysunek 30 Przykładowy ekran mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia	38
Rysunek 31 Struktura ekranu szczegółów wspomnienia	39
Rysunek 32 Powiązane ze wspomnieniem zdjęcie w trybie pełnoekranowym.....	40
Rysunek 33 Miniatura lokalizacji na ekranie szczegółów wspomnienia	40
Rysunek 34 Potwierdzenie chęci wymazania wspomnienia	40
Rysunek 35 Przykładowe informacyjne okno dialogowe	41
Rysunek 36 Przykład nieudanej walidacji	42
Rysunek 37 Przykładowy panel informacyjny	42
Rysunek 38 Przykładowy widget Infotext	43

Rysunek 39 Przykładowy indyktor ładowania	43
Rysunek 40 Mechanizm nasłuchiwania stanu uwierzytelnienia użytkownika	44
Rysunek 41 Funkcja obsługująca napotkane wyjątki Firebase	45
Rysunek 42 Przykład wyłapania i obsługi wyjątku Firebase	45
Rysunek 43 Fragment funkcji odpowiedzialnej za wysyłanie zdjęcia do Cloud Storage, prezentujący wykorzystanie TimeoutException	46
Rysunek 44 Struktura mapy nazw dla danych z Firebase	46
Rysunek 45 Kod źródłowy widgetu CustomCachedImage	47
Rysunek 46 Przykładowy błąd ładowania CustomCachedImage	48
Rysunek 47 Zawartość pliku providers.dart	48
Rysunek 48 Przykład nasłuchiwania zmian stanu z wykorzystaniem ref.watch	49
Rysunek 49 Przykład użycia ref.read do modyfikacji stanu	49
Rysunek 50 Przykład sprawdzenia tego, czy aplikacja działa w wersji webowej	50
Rysunek 51 Warunki definiujące zawartość ekranu galerii wspomnień	51
Rysunek 52 Metoda build widgetu MemoriesList	51
Rysunek 53 Funkcja odpowiedzialna za sortowanie wspomnień	52
Rysunek 54 Funkcja odpowiedzialna za grupowanie wspomnień	53
Rysunek 55 Metoda build widgetu MemoriesSliver	53
Rysunek 56 Funkcja odpowiedzialna za dodawanie i usuwanie wspomnienia z ulubionych	54
Rysunek 57 Przykład użycia widgetu NewPhotoWidget	55
Rysunek 58 Funkcja otwierająca widget PhotoModal	55
Rysunek 59 Funkcja odpowiedzialna za uzyskanie zdjęcia od użytkownika	56
Rysunek 60 Przykład użycia widgetu PhotoModalButton	56
Rysunek 61 Funkcja odpowiedzialna za próbę odczytania daty wykonania zdjęcia	57
Rysunek 62 Funkcja umożliwiająca ręczny wybór daty	57
Rysunek 63 Przykład użycia widgetu NewLocationWidget	58
Rysunek 64 Funkcja odpowiedzialna za uzyskanie lokalizacji	58
Rysunek 65 Funkcja odpowiedzialna za próbę uzyskania adresu wspomnienia z wykorzystaniem Geocoding API	59
Rysunek 66 Funkcja zwracająca adres URL statycznego obrazu mapy	59
Rysunek 67 Definicja modelu danych MapData	60
Rysunek 68 Konstruktor widgetu BaseMapScreen	60
Rysunek 69 Funkcja definiująca wyświetlane na mapie znaczniki	61
Rysunek 70 Funkcja obsługująca ręczny wybór lokalizacji	61
Rysunek 71 Funkcja pobierająca aktualną lokalizację użytkownika	62
Rysunek 72 Funkcja odpowiedzialna za zapisanie wybranej lokalizacji	63
Rysunek 73 Funkcja odpowiedzialna za uzyskanie znaczników wspomnień na podstawie danych z Cloud Firestore	63
Rysunek 74 Funkcja tworząca konkretny znacznik	64
Rysunek 75 Metoda build widgetu MemoriesMap	64
Rysunek 76 Funkcja odpowiedzialna za wyświetlenie ekranu mapy w trybie związanym z wyświetlaniem szczegółów wspomnienia	65
Rysunek 77 Rozwiązanie problemu z nieprawidłowym ładowaniem się mapy	66
Rysunek 78 Funkcja odpowiedzialna za pobieranie zdjęcia do pamięci urządzenia w wersji aplikacji na system Android	67

Rysunek 79 Funkcja odpowiedzialna za pobieranie zdjęcia do pamięci urządzenia w webowej wersji aplikacji	68
Rysunek 80 Funkcja dodająca klucz API Map Google do pliku index.html	68
Rysunek 81 Performance View we Flutter DevTools	69
Rysunek 82 Automatyczne uzupełnienie i wysłanie formularza z informacjami o błędzie napotkanym w aplikacji	72