POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Kierunek studiów: Informatyka **Rodzaj studiów:** Stacjonarne

Praca dyplomowa

Temat pracy: Uniwersalny panel sterowania IoT
Temat w języku angielskim: Universal IoT control panel
Promotor: dr Tadeusz Puźniakowski

Autor: Szymon Kogut
Numer albumu: s24271

Streszczenie: *Praca dyplomowa dotyczy projektowania i implementacji aplikacji mobilnej która funkcjonuje jako uniwersalny interfejs sterowania dla urządzeń IoT. Celem aplikacji jest umożliwienie użytkownikom zdalnie zarządzać różnorodnymi inteligentnymi urządzeniami za pośrednictwem jednej platformy z wykorzystaniem protokołu MQTT. W pracy omówiono wybrane technologie programistyczne które zostały użyte do stworzenia aplikacji oraz wyzwania związane z zapewnieniem bezpieczeństwa i prywatności w ramach rozproszonego środowiska IoT. Aplikacja jest dostępna na urządzenia z systemem Android. Do jej stworzenia wykorzystano język Kotlin oraz bibliotekę Jetpack Compose.*

Słowa kluczowe: iot, mqtt, android, aplikacja, kotlin

Gdańsk 9 stycznia 2025

POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Karta projektu

Akronim	Data ustalenia tematu	Data ukończenia projektu
Atom Dashboard	*28.03.2021*	*16.05.2024*

Promotor	Recenzent	Konsultanci
dr Tadeusz Puźniakowski	*-*	*1. dr hab. Marek Bednarczyk* *2. -* *3. -*

Temat projektu
Uniwersalny panel sterowania IoT
Cele projektu
Stworzenie konkurencyjnego rozwiązania umożliwiającego sterowanie urządzeniami IoT z wykorzystaniem protokołu MQTT; Nauka programowania aplikacji mobilnych
Rezultaty projektu
Aplikacja mobilna z przeznaczniem na system operacyjny Android wykorzystująca protokół MQTT umożliwiająca zbudowanie własnego panelu sterowania inteligentnymi urządzeniami.
Miary sukcesu
Stabilna aplikacja spełniająca postawione wymagania.
Ograniczenia
Dostępność bibliotek obsługujących protokół MQTT; Trudność długotrwałego utrzymania aplikacji pracującej w tle w systemie Android;

Autor	Numer albumu	Specjalizacja	Tryb studiów
Szymon Kogut	*s24271*	*Cyberbezpieczeństwo*	*Stacjonarny*

Spis treści

01. Wstęp

Współczesny świat charakteryzuje się szybkim rozwojem technologii Internetu Rzeczy. Prowadzi to do coraz większej roli automatyzacji w naszym codziennym życiu. Urządzenia te, od inteligentnych żarówek, przez systemy ogrzewania, po zaawansowane systemy bezpieczeństwa, mają na celu ułatwienie codziennych czynności i poprawę jakości życia użytkowników. Jednakże, rosnąca liczba i różnorodność tych urządzeń rodzi nowe wyzwania, zwłaszcza w kontekście ich efektywnego zarządzania i integracji.

Jednym z głównych problemów jest fakt, że producenci urządzeń IoT często stosują zamknięte protokoły komunikacyjne, co utrudnia lub nawet uniemożliwia ich integrację w ramach jednego, spójnego systemu. Taka sytuacja zmusza użytkowników do korzystania z wielu różnych aplikacji do zarządzania swoimi urządzeniami, co jest niewygodne i ogranicza potencjał automatyzacji.

Ważnym punktem również wymagającym poruszenia jest wątpliwe bezpieczeństwo jak i brak przejrzystości przy wykorzystaniu zamknietych rozwiązań jakie oferują producenci urządzeń IoT przeznaczonych na rynek konsumencki. Konsument rzadko ma możliwość dowiedzieć się co się dzieje z jego danymi lub kto ma dostęp do jego urządzeń. Infrastruktury producentów często wykorzystują wiele serwerów rozsianych po całym świecie nad którymi użytkownik nie ma kontroli.

Celem niniejszej pracy inżynierskiej jest zaprojektowanie i implementacja aplikacji mobilnej, która działałaby jako uniwersalny panel do sterowania różnorodnymi urządzeniami IoT. Aplikacja ta ma na celu zaoferowanie użytkownikom centralnego punktu dostępu do wszystkich ich inteligentnych urządzeń.

Słowa obce i pojęcia specyficzne dla kontekstu oznaczone są `szarym tłem`.
Główna część pracy została podzielona na poszczególne fazy:

1. Faza analizy i planowania gdzie analizowana jest cała koncepcja wraz z rynkiem i profilem użytkownika docelowego.

2. Faza projektowania gdzie stawiane są wymagania jakie spełnić ma aplikacja i omawiane są wykorzystane technologie oraz metodyka pracy.

3. Faza implementacji gdzie opisane są poszczególne etapy tworzenia aplikacji.

4. Faza testów końcowych i publikacji gdzie pokazane są ostateczne testy gotowego produktu jaki i publikacja aplikacji na platformie Google Play.

5. Faza utrzymania gdzie opisane są napotkane trudności oraz plany na przyszły rozwój aplikacji.

02. Słownik pojęć

1. **Android** - *system operacyjny stworzony przez Google, oparty na jądrze Linux, przeznaczony głównie dla urządzeń mobilnych, takich jak smartfony i tablety.*
2. **API** - *(ang. Application Programming Interface) interfejs programistyczny aplikacji, który umożliwia komunikację między różnymi programami lub komponentami oprogramowania.*
3. **App Store** - *platforma dystrybucji aplikacji dla urządzeń z systemem iOS, zarządzana przez firmę Apple.*
4. **Asynchroniczność** - *sposób wykonywania operacji w programowaniu, w którym zadania są realizowane niezależnie od siebie, bez blokowania głównego wątku aplikacji.*
5. **Broker** - *patrz 03.1.3*
6. **Dashboard** - *interfejs użytkownika zawierający podsumowanie kluczowych informacji i wskaźników, często w formie wizualnej, takich jak wykresy czy tabele.*
7. **Firebase** - *platforma stworzona przez Google, oferująca szeroki zestaw narzędzi i usług wspierających tworzenie, rozwój oraz utrzymanie aplikacji mobilnych i webowych. Obejmuje m.in. bazę danych w czasie rzeczywistym, uwierzytelnianie użytkowników, powiadomienia push, hosting, analitykę aplikacji, a także narzędzia takie jak Crashlytics do monitorowania błędów i awarii aplikacji w celu poprawy jej stabilności i jakości.*
8. **Fragment** - *moduł interfejsu użytkownika w Androidzie, który może być wielokrotnie używany w różnych aktywnościach lub aplikacjach.*
9. **Git** - *system kontroli wersji umożliwiający śledzenie zmian w kodzie źródłowym i współpracę wielu programistów nad jednym projektem.*
10. **Google Play** - *oficjalny sklep z aplikacjami dla urządzeń z systemem Android, zarządzany przez Google.*
11. **Google Play Console** - *platforma stworzona przez Google dla deweloperów aplikacji na Androida, umożliwiająca zarządzanie publikacją aplikacji w sklepie Google Play. Oferuje narzędzia do monitorowania statystyk pobrań, zarządzania opiniami użytkowników, testowania aplikacji, analizy wydajności, monitorowania błędów (Crash Reports) oraz konfiguracji funkcji takich jak subskrypcje, zakupy w aplikacji czy kampanie promocyjne.*
12. **IoT** - *(ang. Internet of Things) koncepcja sieci urządzeń połączonych z Internetem, które mogą wymieniać dane i działać autonomicznie, np. inteligentne domy czy urządzenia przemysłowe.*
13. **Jetpack Compose** - *nowoczesny framework UI dla Androida, oparty na deklaratywnym podejściu do tworzenia interfejsów użytkownika.*
14. **JSON** - *(ang. JavaScript Object Notation) lekki format wymiany danych, oparty na strukturze klucz-wartość, łatwy do odczytu zarówno dla ludzi, jak i maszyn.*
15. **Kotlin** - *nowoczesny język programowania stworzony przez JetBrains, oficjalnie wspierany przez Google jako główny język do tworzenia aplikacji na Androida.*

16. **Kotlin Coroutines** - *mechanizm w języku Kotlin umożliwiający łatwe tworzenie i zarządzanie operacjami asynchronicznymi w sposób sekwencyjny.*
17. **Material Design** - *język projektowania stworzony przez Google, który definiuje zasady tworzenia estetycznych, spójnych i intuicyjnych interfejsów użytkownika.*
18. **Message payload** - *patrz 03.1.3*
19. **Programowanie zdarzeniowe** - *paradygmat programowania, w którym przepływ programu jest sterowany przez zdarzenia, takie jak kliknięcia użytkownika, dane z czujników czy wiadomości sieciowe.*
20. **Retained message** - *patrz 03.1.3*
21. **Rolnictwo precyzyjne** - *nowoczesna metoda zarządzania gospodarstwami rolnymi, wykorzystująca technologie, takie jak IoT, GPS czy drony, w celu optymalizacji produkcji i minimalizacji strat.*
22. **Sentry** - *narzędzie do monitorowania błędów i zarządzania nimi w aplikacjach, które pozwala na szybkie diagnozowanie problemów w kodzie.*
23. **Serializacja** - *proces przekształcania obiektu w formę, która może być przechowywana lub przesyłana, np. do formatu JSON lub XML.*
24. **Single-Activity** - *podejście w programowaniu aplikacji na Androida, w którym cała aplikacja składa się z jednej aktywności, a jej zawartość jest zarządzana za pomocą fragmentów.*
25. **Topic** - *patrz 03.1.3*
26. **XML** - *(ang. eXtensible Markup Language) język znaczników używany do przechowywania i przesyłania danych w strukturze hierarchicznej, często wykorzystywany w konfiguracji aplikacji i interfejsach użytkownika.*

03. Przedstawienie problemu

03.1. Kontekst problemu

03.1.1. Internet rzeczy

W ostatnich latach obserwujemy gwałtowny wzrost automatyzacji i cyfryzacji. (statystyka - [22]) Inteligentne urządzenia możemy znaleźć już w wielu miejscach: w strefach przemysłowych, gdzie ich zadaniem jest ulepszenie i ułatwienie procesu produkcji, na naszych drogach, gdzie pomagają w odkorkowywaniu miast. Obecnie coraz większą popularnością cieszą się one również w naszych domach (statystyka - [23]), najczęściej pod postacią drobnych urządzeń znanych producentów. Takie urządzenia pomagają nam w automatyzacji codziennych czynności: zdalne zarządzanie światłem, kontrola poboru prądu, klimatyzacja i ogrzewanie, ale również mogą monitorować nasze bezpieczeństwo, przykładowo z wykorzystaniem czujnika zalania w łazience. Z użyciem grupy czujników i sensorów oraz elementów wykonawczych takich jak wentylatory, grzejniki, żarówki, przełączniki, głośniki i wiele innych możemy zautomatyzować wiele codziennych czynności, poprawić komfort naszego życia jak i ograniczyć zużycie zasobów.

Dla przykładu, inteligentne termostaty uczą się naszych zwyczajów i dostosowują ogrzewanie do naszego trybu życia, obniżając temperaturę, kiedy jesteśmy poza domem, co przekłada się na znaczne oszczędności energetyczne. Znając lokalizację domowników system może wyłączyć ogrzewanie jeżeli dom stoi pusty i zawczasu je włączyć gdy się zbliżamy. Możemy poprawić zużycie prądu, ogrzewając wodę w bojlerze jedynie w nocy gdy taryfa jest bardziej opłacalna czy ustawić harmonogram podlewania roślin w ogrodzie. Opcji jest multum.

Dodatkowo, inteligentne systemy mogą poprawić jakość życia osób starszych lub niepełnosprawnych, umożliwiając im większą samodzielność. Za pomocą prostych komend głosowych mogą one kontrolować różne urządzenia w domu, od telewizora po żaluzje, co sprawia, że ich codzienne życie staje się łatwiejsze i bardziej komfortowe.

Na rynku pojawiają się również inteligentne urządzenia kuchenne, takie jak lodówki, które monitorują stan zapasów i mogą automatycznie zamawiać produkty spożywcze, kiedy zaczynają się kończyć, czy piekarniki, które dostosowują czas i sposób pieczenia do rodzaju potrawy. To wszystko sprawia, że nasze życie staje się nie tylko wygodniejsze, ale także bardziej ekologiczne, ponieważ dzięki precyzyjnej kontroli zużycia możemy znacznie zmniejszyć nasz ślad węglowy.

Automatyzacja i inteligentne urządzenia zmieniają sposób, w jaki żyjemy, pracujemy i odpoczywamy, oferując nam nowe możliwości i sprawiając, że nasz codzienny komfort jest na wyższym poziomie.

03.1.2. Zamknięte oprogramowanie

Całość brzmi świetnie: lepiej, szybciej, więcej, wygodniej. Niestety wiele urządzeń popularnych producentów wykorzystuje zamknięte oprogramowanie. Często musimy używać wiele aplikacji różnych producentów aby sterować wszystkimi urządzeniami w naszym domu. To jednak problem nie tylko wpływający na wygodę użytkownika. Zamknięte oprogramowanie rodzi jeszcze jeden o wiele poważniejszy. Nie posiadamy kontroli ani wiedzy w jaki sposób te urządzenia komunikują się z infrastrukturą producenta. Jakie dane wysyłają i czy nie posiadają luk bezpieczeństwa. Co w przypadku gdy te urządzenia są źle zaprojektowane i posiadają podatności za pośrednictwem których osoby trzecie mogą przejąć kontrolę nad całą naszą siecią domową. Prosta żarówka którą możemy zdalnie włączyć, nie zależnie od tego czy jesteśmy w salonie naszego domu czy na drugim końcu świata, musi komunikować się z serwerami producenta o których wiemy niewiele.

Znane są przypadki gdy inteligentne odkurzacze czy domowe dzwonki odmawiały posłuszeństwa z powodu awarii serwerów. (artykuł BBC - [4]) Dodatkowo, te awarie mogą prowadzić nie tylko do tymczasowego braku dostępu do funkcji urządzenia, ale również do trwałej utraty danych lub nieautoryzowanego do nich dostępu przez osoby trzecie. Bez odpowiedniej transparentności i kontroli nad tym, jak nasze urządzenia komunikują się z zewnętrznymi serwerami, stajemy się podatni na różnego rodzaju ataki cybernetyczne. To, co wydaje się być wygodą, może przekształcić się w poważne zagrożenie dla naszej prywatności i bezpieczeństwa.

Otwarte oprogramowanie nie tylko zwiększa transparentność działania urządzeń, umożliwiając użytkownikom lepsze zrozumienie procesów zachodzących w ich sprzęcie, ale również pozwala na szybsze wykrywanie i łatanie ewentualnych luk bezpieczeństwa przez społeczność. (GNU Manifesto - [25])

Stosowanie lokalnych protokołów komunikacyjnych, które nie wymagają stałego połączenia z serwerami producenta zmniejsza ryzyko awarii i nieautoryzowanego dostępu do naszych danych. Przykładem mogą być systemy, które działają w pełni wewnątrz sieci domowej, bez potrzeby wysyłania danych na zewnątrz.

Ważne jest, aby konsumenci byli świadomi tych zagrożeń i domagali się od producentów większej przejrzystości oraz bezpieczeństwa. Wymaganie od producentów wdrażania standardów otwartego oprogramowania oraz lokalnych protokołów komunikacyjnych może przyczynić się do zwiększenia bezpieczeństwa i prywatności w naszych domach. Tylko w ten sposób możemy mieć pewność, że korzyści płynące z inteligentnego domu nie będą miały negatywnych konsekwencji dla naszego bezpieczeństwa.

03.1.3. Protokół MQTT

Stworzony w 1999 roku prosty i lekki protokół komunikacyjny MQTT (Message Queuing Telemetry Transport) jest przeznaczony dla urządzeń niewymagających wysokiej przepustowości. Protokół ten przeszedł kilka istotnych aktualizacji, w tym wersje 3.1 (dokumentacja - [19]) i 5.0 (dokumentacja - [20]), które wprowadziły nowe funkcje i usprawnienia, takie jak lepsze zarządzanie jakością usług, rozszerzone możliwości zabezpieczeń oraz wsparcie dla bardziej złożonych scenariuszy komunikacyjnych. (artykuły HiveMQ - [12][21]) MQTT jest szeroko wykorzystywany w IoT oraz wszędzie tam, gdzie wymagana jest oszczędność przepustowości i energii.

Broker
Architektura oparta o MQTT wymaga serwera na którym pracuje broker wiadomości implementujący protokół MQTT. Broker stanowi centralny punkt do którego łączą się wszyscy klienci. Przykładem brokera MQTT jest Eclipse Mosquitto. (dokumentacja - [18])

Topic
Abstrakcyjne identyfikatory pomagające przesyłać wiadomości do konkretnych odbiorców. Nie istnieją fizycznie a publikacja i subskrypcja do nich nie wymaga uprzedniego ich utworzenia. Dany `topic` można porównać do tablicy ogłoszeń którą każdy zainteresowany może zasubskrybować aby otrzymać powiadomienie za każdym razem gdy ktoś publikuje na niej nową wiadomość.

Aby otrzymać to powiadomienie klient musi, nie tylko wcześniej zasubskrybować dany `topic`, ale również być na czas publikacji połączonym z brokerem. Jedynym wyjątkiem od tej zasady jest przypadek
gdy `topic` posiada `retained message`.

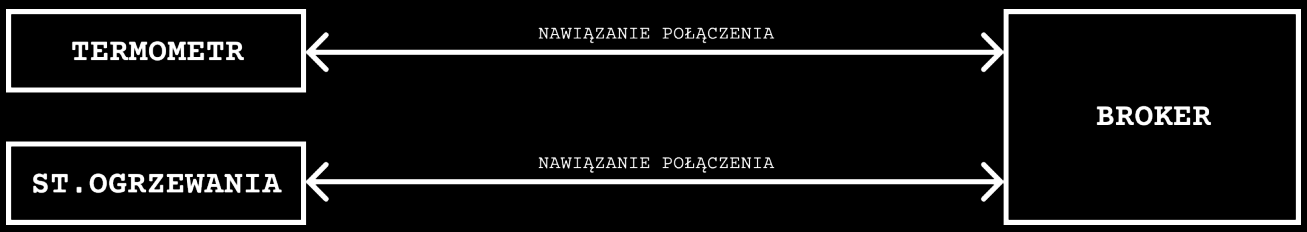
Retained message
Jest to specjalny rodzaj wiadomości wysyłanej na dany `topic` różniący się ustawioną flagą `retained`. Broker zachowuje taką wiadomość i przesyła ją do klienta na moment subskrypcji tego `topic`. Każdy `topic` może mieć tylko jedną `retained message`. Aby ją usunąć wymagana jest publikacja nowej wiadomości z flagą `retained` oraz pustym `payload`.

Message payload
Jest to ciąg znaków który wysyłany jest z każdą wysłaną wiadomością. Dla przykładowego `topic` o nazwie temperatura może być to aktualna temperatura w pokoju wyrażona w stopniach Celsjusza.

03.1.4. Przykład wykorzystania

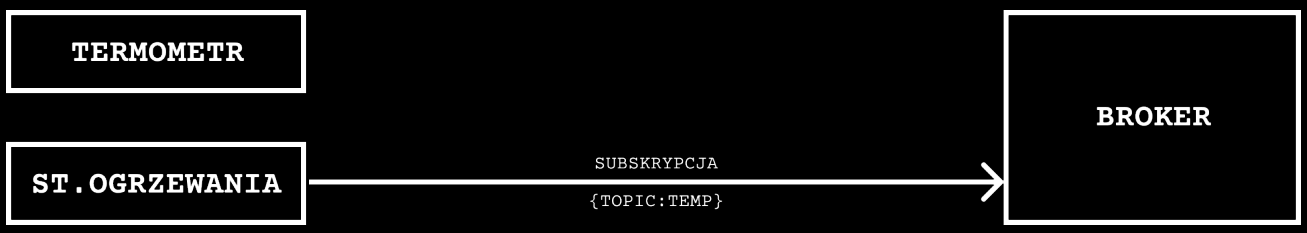
Poniższe grafiki prezentują przykład użycia protokołu MQTT w praktyce. W przykładzie bierze udział troje urządzeń: termometr mierzący temperaturę w pokoju, sterownik ogrzewania mający za zadanie na podstawie zmierzonej temperatury włączać bądź wyłączać ogrzewanie oraz panel sterowania który dla uproszczenia został celowo wykluczony ze schematów.

1. Jak przedstawiono na rysunku 1 wpierw każde z urządzeń nawiązuje połączenie do brokera które pozostaje stale utrzymane na czas pracy urządzenia. W przypadku zerwania połączenia, urządzenie stara się je ponownie nawiązać ponieważ w innym razie jest wykluczone z komunikacji.



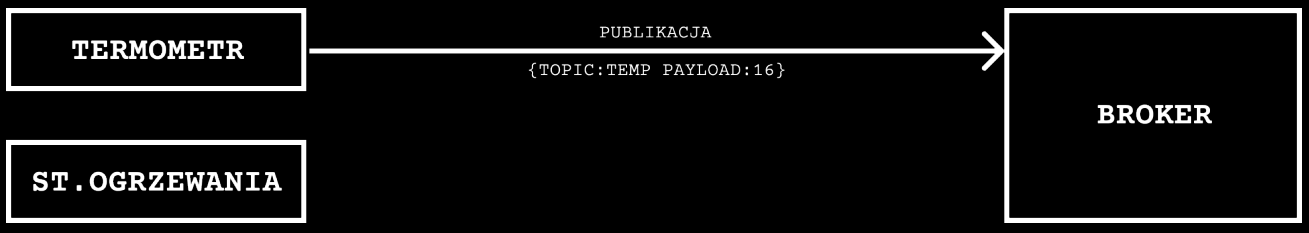
Rys. 1. Nawiązanie połączenia

2. Zgodnie z rysunkiem 2 po połączeniu każde urządzenie które tego potrzebuje subskrybuje interesujące je `topic`. W tym przykładzie sterownik ogrzewania subskrybuje `topic` temp. W tym momencie, jeżeli `topic` temp posiadałby retained message, sterownik otrzymałby tą wiadomość dokładnie tak samo jakby była ona świeżo nadana. W innym przypadku sterownik nie znałby obecnej temperatury do momentu nadania jej przez termometr w kolejnym kroku.



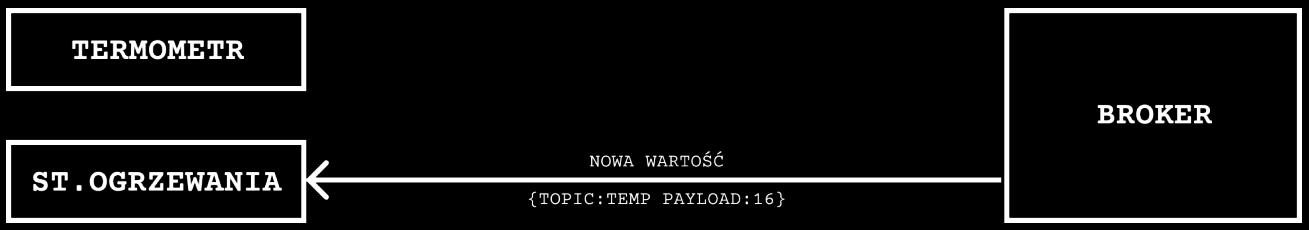
Rys. 2. Subskrypcja

3. Na rysunku 3 przedstawiono jak termometr wysyła nową wiadomość na `topic` temp z payload wynoszącym wartość zmierzonej temperatury. Ten krok powtarza się z określoną częstotliwością. W ten sposób `topic` temp reprezentuje aktualną temperaturę w pokoju.



Rys. 3. Publikacja nowej wiadomości

4. Rysunek 4 prezentuje jak w odpowiedzi na publikację nowej wiadomości broker powiadamia każde urządzenie subskrybujące `topic` temp. Po otrzymaniu aktualnej wartości sterownik może podjąć decyzję czy włączyć lub wyłączyć ogrzewanie w pokoju. Sam stan ogrzewania może być oddzielnym `topic`, tak aby panel sterowania mógł wyświetlić informację czy jest ono włączone bądź nie: gdy sterownik włącza ogrzewanie publikuje 1 na `topic` `ogrz`, a gdy je wyłącza publikuje 0.



Rys. 4. Rozgłoszenie nowej wiadomości do subskrybentów

03.2. Problem

W obliczu dynamicznego rozwoju technologii IoT oraz rosnącej liczby inteligentnych urządzeń w naszych domach, pojawiają się nowe wyzwania związane z bezpieczeństwem, prywatnością oraz wygodą użytkowania. Zamknięte oprogramowanie, które jest powszechnie stosowane przez wielu producentów, ogranicza kontrolę użytkowników nad ich własnymi urządzeniami oraz naraża ich na różnego rodzaju zagrożenia.

Jednym z głównych problemów jest brak transparentności w działaniu zamkniętych systemów. Użytkownicy nie mają wglądu w to, jakie dane są przesyłane do serwerów producentów oraz w jaki sposób są one przetwarzane. To rodzi obawy o prywatność oraz bezpieczeństwo danych osobowych. Ponadto, awarie serwerów producentów mogą prowadzić do utraty funkcjonalności urządzeń, co jest szczególnie problematyczne w przypadku urządzeń kluczowych dla codziennego funkcjonowania domu.

W odpowiedzi na te wyzwania, proponowane rozwiązanie zakłada stworzenie otwartoźródłowej aplikacji mobilnej, która będzie pełnić funkcję uniwersalnego dashboardu do zarządzania urządzeniami IoT opartymi na protokole MQTT. Aplikacja ta ma na celu zapewnienie użytkownikom pełnej kontroli nad ich inteligentnymi urządzeniami, a także zwiększenie bezpieczeństwa i prywatności.

04. Faza analizy i planowania

04.1. Koncepcja

Aplikacja mobilna która działałaby jako uniwersalny interfejs sterowania różnymi urządzeniami IoT. Projekt zakłada użycie MQTT jako protokołu komunikacyjnego z uwagi na szerokie zastosowanie w domenie IoT ale również przewiduje możliwość przyszłego rozszerzenia o inne takie pozwalające na komunikację bezpośrednio z urządzeniem lub siecią urządzeń. (artykuły HiveMQ - [15][1]) Użytkownik korzystając z szerokiej gamy predefinowanych kafelek może stworzyć interfejs pozwalający sterować dowolnym urządzeniem o ile implementuje ono jeden z obsługiwanych protokołów. Przedstawiana aplikacja może stać się centrum sterowania inteligentnego domu gdzie nadzoruje pracę całej automatyki lub też zostać wykorzystana jako interfejs użytkownika w przypadku projektów samodzielnych urządzeń lub prototypów gdzie tworzenie dedykowanego jest zbędnym wydatkiem. Z racji na nacisk na uniwersalne protokoły cała komunikacja może pozostać pod kontrolą użytkownika wykorzystując całkowicie jego infrastrukturę.

04.2. Konkurencyjne rozwiazania

MQTT Dash stworzone przez **Routix software** oraz **IoT MQTT Panel** autorstwa **Rahul Kundu**: obie te aplikacje posiadają przestarzały interfejs użytkownika co dyskwalifikuje je na start. Dodatkowym minusem jest fakt że nie umożliwiają użycia własnych certyfikatów SSL czy certyfikatów klienta. Pierwsza z wymienionych nie otrzymała aktualizacji od 2017 roku co sugeruje że jest porzuconym projektem.

Mqtt Dashboard stworzone przez **Vetru** jest jedynym bezpośrednim konkurentem. Obsługuje zarówno własne certyfikaty SSL oraz certyfikaty klientów. Posiada aktualny i przejrzysty interfejs który może przypaść wielu do gustu. Ostatnia aktualizacja nastąpiła w 2021 roku.

04.3. Profil użytkownika docelowego

Użytkownik zaznajomiony z technologią posiadający wiedzę oraz umiejętności techniczne związane ze światem IoT. Wymagający szerokich możliwości konfiguracji i dostosowywania pod własne potrzeby. Zwracający uwagę na prywatność oraz ceniący sobie używanie własnych rozwiązań ponad te z zamkniętego źródła.

04.3.1. Model biznesowy

Projekt nie kładzie silnego nacisku na komercjalizację. W związku z powyższym wprowadzono model Freemium:

- Brak reklam w wersji darmowej
- Możliwość wsparcia dewelopera poprzez dotacje
- Aplikacja dostępna w darmowej wersji posiadającej wszystkie funkcjonalności
- Zakupy i wpłaty obsługiwane bezpośrednio w aplikacji z wykorzystaniem systemu płatności Google Play Billing
- Możliwość zakupu wersji rozszerzonej zdejmującej ograniczenie co do ilości stworzonych dashboardów

04.4. Przykłady zastosowania

W tej sekcji znajdują się trzy przykładowe scenariusze wykorzystania prezentowanej aplikacji. W opisach celowo pominięto serwer MQTT.

04.4.1. Automatyka domowa

Przykład zastosowania aplikacji do sterowania automatyką domową:

Sterownik oświetlenia RGB - jego rolą jest kontrolowanie oświetlenia. Nie wysyła żadnych nowych informacji więc jedynie subskrybuje cztery `topic` związane z ustawionym kolorem, stanem, trybem oraz jasnością oświetlenia.

Sterownik z przekaźnikiem - jest odpowiedzialny za sterowanie zasilaniem podłączonego urządzenia, w tym przypadku wentylatora. Subskrybuje jeden topic. W zależności od payload włącza lub wyłącza zasilanie. W alternatywnej implementacji mógłby jedynie przełączać stan urządzenia nie zważając na payload.

Sterownik ogrzewania podłogowego - ma za zadanie kontrolować ogrzewaniem w pokoju. Z tego powodu subskrybuje dwa `topic` zawierające zadaną oraz aktualną temperaturę.

Czujnik klimatu - przesyła na dedykowane `topic` temperaturę i wilgotność powietrza oraz ciśnienie atmosferyczne.

Aplikacja - wyświetla interfejs użytkownika pokazujący informacje publikowane przez czujniki jak i dający możliwość kontroli pracy pozostałych urządzeń. Z tego powodu subskrybuje i publikuje do wszystkich wyżej opisanych `topic`.

04.4.2. Rolnictwo precyzyjne

Przykład zastosowania aplikacji do rolnictwa precyzyjnego [26]:

Czujniki wilgotności gleby - mierzą poziom wilgotności w glebie na różnych głębokościach.

Czujniki upraw - mierzące temperaturę i wilgotność powietrza w polu.

Czujnik nasłonecznienia - mierzy ilość światła słonecznego docierającego do roślin.

System nawadniania - sterowane zdalnie na podstawie danych z sensorów.

Stacja meteorologiczna - mierzy różne parametry meteorologiczne, takie jak prędkość i kierunek wiatru, opady, ciśnienie atmosferyczne.

Serwer - na podstawie zebranych pomiarów steruje automatycznym nawadnianiem. Powiadamia o wszelkich anomaliach. Agreguje oraz archiwizuje zebrane dane do przyszłej długoterminowej analizy.

Aplikacja - wyświetla interfejs użytkownika pokazujący informacje publikowane przez czujniki jak i dający możliwość kontroli pracy pozostałych urządzeń. Z tego powodu subskrybuje i publikuje do wszystkich wyżej wymienionych `topic`.

04.4.3. Inteligentne budynki

Przykład zastosowania aplikacji w kontekście inteligentnych budynków:

Czujniki obecności - wykrywają obecność osób w pomieszczeniach, co pozwala na optymalizację oświetlenia i klimatyzacji.

Inteligentne termostaty - kontrolują temperaturę w pomieszczeniach na podstawie danych z czujników obecności oraz preferencji użytkowników. Subskrybują `topic` związany z ustawioną temperaturą oraz publikują aktualną temperaturę.

Sterowniki oświetlenia - zarządzają oświetleniem w budynku, dostosowując jego intensywność i barwę w zależności od pory dnia, obecności osób i naturalnego światła. Subskrybują `topic` związany z ustawieniem oświetlenia.

Czujniki jakości powietrza - monitorują poziom CO2, wilgotność oraz inne parametry jakości powietrza, co pozwala na optymalizację wentylacji i klimatyzacji. Publikują dane na dedykowany topic.

System zarządzania energią - monitoruje zużycie energii w budynku, optymalizując jej wykorzystanie poprzez zarządzanie oświetleniem, ogrzewaniem, wentylacją i klimatyzacją. Publikuje i subskrybuje dane dotyczące zużycia energii.

Serwer budynkowy - agreguje dane ze wszystkich czujników i urządzeń, analizuje je oraz podejmuje decyzje dotyczące zarządzania budynkiem. Powiadamia odpowiednie służby o wszelkich anomaliach i awariach. Archiwizuje dane do przyszłej analizy i planowania.

Aplikacja - wyświetla interfejs użytkownika pokazujący informacje publikowane przez czujniki oraz daje możliwość kontroli pracy pozostałych urządzeń. Z tego powodu subskrybuje i publikuje do wszystkich wyżej wymienionych `topic`.

05. Faza projektowania

Ten dział pracy poświęcony został opisowi wymagań jakie miał spełniać gotowy produkt. Wymagania te zostały podzielone na trzy kategorie: jakościowe, niefunkcjonalne oraz funkcjonalne. Wymagania jakościowe opisują oczekiwania co do wyglądu oraz interakcji z aplikacją. Wymagania niefunkcjonalne opisują oczekiwania co do jakości i struktury kodu oraz rodzaju stosowanych rozwiązań w projekcie. Wymagania funkcjonalne opisują funkcje jakie aplikacja ma posiadać.

Poniżej znajduje się przykład karty wymagania z opisem poszczególnych pól.

KARTA WYMAGANIA	
Identyfikator	Unikalny identyfikator wymagania
Priorytet	Priorytet wymagania według metodyki MoSCoW
Nazwa	Nazwa wymagania
Powiązane	Opcjonalna lista identyfikatorów wymagań powiązanych
Opis	Krótki opis przedstawiający wymaganie
Uwagi	Opcjonalne dodatkowe uwagi dotyczące wymagania. Mają za zadanie uściślić opis wymagania, jeżeli zachodzi taka potrzeba. W przypadku braku wymogu dodatkowych informacji, pole to pozostaje puste. Nie obejmują informacji, które są powszechnie znane, domyślne lub standardowe.

05.1. Wymagania jakościowe

05.1.1. Estetyka

- atrakcyjny i nowoczesny design który przyciąga uwagę
- możliwość personalizacji interfejsu graficznego
- minimalistyczny i przejrzysty styl graficzny

05.1.2. Przejrzystość

- łatwy w nawigowaniu interfejs użytkownika
- krótkie oraz klarowne komunikaty dla użytkownika
- spójny układ i styl zapewniające jednolite doświadczenia

05.1.3. Użyteczność

- płynne oraz proste animacje
- łatwo dostępne i intuicyjne funkcjonalności
- powtarzalne rezultaty tych samych czynności

05.2. Wymagania niefunkcjonalne

KARTA WYMAGANIA	
Identyfikator	NF3322
Priorytet	MUST
Nazwa	Kompatybilność z systemem Andrioid
Opis	Aplikacja jest możliwa do zainstalowania na urządzeniach z systemem Android
Uwagi	Aplikacja wspiera system Android w wersji 8.0 wzwyż

KARTA WYMAGANIA	
Identyfikator	NF8047
Priorytet	MUST
Nazwa	Kompatybilność z platformą Google Play [9]
Opis	Aplikacja kwalifikuje się do publikacji na platformie Google Play
Uwagi	Aplikacja spełnia wymagania stawiane przez platformę Google Play

KARTA WYMAGANIA	
Identyfikator	NF2026
Priorytet	MUST
Nazwa	Wsparcie dla różnych ekranów
Opis	Aplikacja wspiera ekrany większości popularnych konsumenckich urządzeń mobilnych w orientacji pionowej oraz poziomej
Uwagi	Aplikacja wspiera ekrany: - w orientacji pionowej i poziomej - o niskiej i wysokiej rozdzielczości - o różnej proporcji

KARTA WYMAGANIA	
Identyfikator	NF7945
Priorytet	MUST
Nazwa	Zarządzanie błędami
Opis	Aplikacja skutecznie zarządza błędami i wyjątkami
Uwagi	• Aplikacja zawiera mechanizmy logowania błędów • Aplikacja informuje użytkowników o błędach w sposób zrozumiały

KARTA WYMAGANIA	
Identyfikator	NF2946
Priorytet	MUST
Nazwa	System kontroli wersji
Opis	Historia zmian w kodzie projektu jest śledzona z wykorzystaniem systemu Git
Uwagi	Projekt wykorzystuje system GitHub

KARTA WYMAGANIA	
Identyfikator	NF9356
Priorytet	MUST
Nazwa	Jetpack Compose
Opis	Projekt wykorzystuje framework Jetpack Compose do tworzenia interfejsu graficznego
Uwagi	Ograniczone do minimum użycie XML w projekcie

KARTA WYMAGANIA	
Identyfikator	NF5543
Priorytet	MUST
Nazwa	Kotlin
Opis	Aplikacja jest napisana w języku kotlin
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	NF6042
Priorytet	MUST
Nazwa	Architektura Single-Activity
Opis	Aplikacja jest oparta o architekturę Single-Activty
Uwagi	• Minimalna ilość Activity w projekcie • Interfejs użytkownika jest podzielony na fragmenty

KARTA WYMAGANIA	
Identyfikator	NF5154
Priorytet	MUST
Nazwa	Obiektowość
Opis	Aplikacja jest napisana zgodnie z paradygmatem programowania obiektowego
Uwagi	• Wszystkie główne komponenty aplikacji są reprezentowane jako klasy • Klasy są odpowiednio zorganizowane w pakiety zgodnie z ich funkcjonalnością • Wykorzystanie dziedziczenia oraz polimorfizmu tam gdzie jest to odpowiednie • Kod jest modularny i łatwy do rozszerzenia oraz utrzymania

KARTA WYMAGANIA	
Identyfikator	NF9657
Priorytet	MUST
Nazwa	Asynchroniczność
Opis	Aplikacja wykorzystuje Kotlin Coroutines do zarządzania operacjami asynchronicznymi aby zapewnić efektywne wykonywanie zadań bez blokowania głównego wątku
Uwagi	Długotrwałe operacje wykonywane są z użyciem Kotlin Coroutines

KARTA WYMAGANIA	
Identyfikator	NF7396
Priorytet	MUST
Nazwa	Model oparty na zdarzeniach
Opis	Aplikacja jest zaprojektowana w sposób oparty na zdarzeniach aby reagować na różne akcje użytkownika oraz zmiany stanu aplikacji
Uwagi	• Wykorzystanie rozwiązań wspierających programowanie zdarzeniowe • Użycie wzorca obserwatora do reagowania na zmiany w aplikacji

KARTA WYMAGANIA	
Identyfikator	NF4361
Priorytet	MUST
Nazwa	Ekrany ładowania
Opis	Aplikacja używa ekranów ładowania gdzie jest ryzyko długotrwałego oczekiwania przez użytkownika
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	NF5400
Priorytet	MUST
Nazwa	Monitorowanie
Opis	Projekt wykorzystuje zewnętrzne platformy do śledzenia stanu aplikacji po opublikowaniu
Uwagi	Projekt używa minimum jednej z poniższych platform: - Sentry - Firebase

KARTA WYMAGANIA	
Identyfikator	NF6368
Priorytet	MUST
Nazwa	Material Design [16]
Opis	Projekt implementuje system projektowania Material Design
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	NF7440
Priorytet	SHOULD
Nazwa	Krótki czas uruchamienia
Opis	Aplikacja uruchamia się w krótkim czasie
Uwagi	Aplikacja uruchamia się poniżej 1 sekundy na flagowych urządzeniach

KARTA WYMAGANIA	
Identyfikator	NF3608
Priorytet	COULD
Nazwa	Material Design 3 [17]
Opis	Projekt implementuje system projektowania Material Design 3
Uwagi	Ograniczone do minimum użycie Material Design 2 w projekcie

KARTA WYMAGANIA	
Identyfikator	NF7108
Priorytet	COULD
Nazwa	Wielojęzyczność
Opis	Aplikacja wspiera wiele języków
Uwagi	• Aplikacja jest dostępna w co najmniej trzech językach • Projekt umożliwia łatwe dodawanie kolejnych języków

KARTA WYMAGANIA	
Identyfikator	NF2166
Priorytet	COULD
Nazwa	Kompatybilność z systemem iOS
Opis	Aplikacja jest możliwa do zainstalowania na urządzeniach z systemem iOS
Uwagi	Aplikacja wspiera system iOS w wersji 15 wzwyż

KARTA WYMAGANIA	
Identyfikator	NF2343
Priorytet	COULD
Nazwa	Kompatybilność z platformą App Store
Opis	Aplikacja spełnia wymagania stawiane przez platformę App Store
Uwagi	Aplikacja kwalifikuje się do publikacji na platformie App Store

05.3. Wymagania funkcjonalne

KARTA WYMAGANIA	
Identyfikator	F1733
Priorytet	MUST
Nazwa	Ekran główny
Powiązane	F2678 | F6487 | F3297 | F8612
Opis	Aplikacja posiada ekran główny
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F5502
Priorytet	MUST
Nazwa	Ekran ustawień
Powiązane	F4998 | F4785 | F3357 | F9885 | F5157 | F6742 | F1655 | F9898 | F3461 | F9203
Opis	Aplikacja posiada ekran ustawień
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F2496
Priorytet	MUST
Nazwa	Ekran dashboardu
Powiązane	F3412 | F9152 | F8766 | F8323 | F1891 | F9703 | F1463
Opis	Aplikacja posiada ekran wyświetlający dany dashboard
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F5060
Priorytet	MUST
Nazwa	Ekran konfiguracji kafelek
Powiązane	F3412 | F7166 | F9215 | F4666 | F3321 | F9132 | F4100 | F1124 | F1612
Opis	Aplikacja posiada ekran konfiguracji danej kafelki
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F1685
Priorytet	MUST
Nazwa	Ekran konfiguracji dashboardów
Powiązane	F3412 | F8156 | F5780 | F6612 | F9272 | F3936
Opis	Aplikacja posiada ekran konfiguracji danego dashboardu
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F3044
Priorytet	MUST
Nazwa	Ekran wsparcia
Powiązane	F3109 | F7429 | F8159
Opis	Aplikacja posiada ekran wsparcia
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F4884
Priorytet	MUST
Nazwa	Protokół MQTT
Powiązane	F5780 | F6612 | F9272 | F3936
Opis	Aplikacja komunikuje się z wykorzystaniem protokołu MQTT
Uwagi	Każdy dashboard jest oddzielnym klientem protokołu MQTT

KARTA WYMAGANIA	
Identyfikator	F4734
Priorytet	MUST
Nazwa	Kafelki
Powiązane	F7166 | F9215 | F4666 | F9132 | F4100 | F1124 | F1612 | F5163 | F8766
Opis	Aplikacja umożliwia dodawanie kafelek różnego typu do dashboardów
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F3750
Priorytet	MUST
Nazwa	Dashboardy
Powiązane	F8612 | F8156 | F1146 | F4884 | F9152
Opis	Aplikacja umożliwia dodanie wielu dashboardów
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F3412
Priorytet	MUST
Nazwa	Szybka nawigacja
Powiązane	F6395
Opis	Aplikacja przewiduje możliwość szybkiej nawigacji wykorzystując gesty lub przyciski nawigacyjne
Uwagi	• Szybka nawigacja umożliwia przełączanie się pomiędzy ekranami: - dashboardów - konfiguracji dashboardów - konfiguracji kafelek tego samego dashboaru • Strzałki nawigacyjne są schowane w przypadku pojedynczego dashboardu lub pojedynczej kafelki

KARTA WYMAGANIA	
Identyfikator	F6395
Priorytet	MUST
Nazwa	Konfiguracja strzałek nawigacyjnych
Powiązane	F3412
Opis	Istnieje możliwość ukrycia strzałek nawigacyjnych
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F7166
Priorytet	MUST
Nazwa	Powiadomienia kafelek
Powiązane	F9203
Opis	Kafelki posiadają możliwość włączenia funkcji wysłania powiadomienia w przypadku otrzymania nowej wartości
Uwagi	Konfiguracja powiadomień kafelek obejmuje - wyłączenie dźwięku powiadomienia - ustawienie payloadu powiadomienia - ustawienie tytułu powiadomienia

KARTA WYMAGANIA	
Identyfikator	F1612
Priorytet	MUST
Nazwa	Animacje kafelek
Powiązane	F9898
Opis	Kafelki posiadają animację aktualizacji
Uwagi	• Animacja jest uruchamiana przy każdej nowej wiadomości • Animowany jest rozmiar kafelek poprzez zmniejszenie ich na krótką chwilę

KARTA WYMAGANIA	
Identyfikator	F3357
Priorytet	MUST
Nazwa	Personalizacja
Powiązane	F9885
Opis	Aplikacja daje możliwość personalizacji poprzez ustawienie koloru wiodącego
Uwagi	• Użytkownik wybiera kolor używając interfejsu w formacie HSV • Aplikacja generuje paletę kolorów na podstawie wybranego koloru • Wygenerowana paleta wpływa na cały interfejs aplikacji • Interfejs zachowuje wysoki kontrast niezależnie od wybranego koloru • Dozwolona jest implementacja monochromatycznej palety • Paleta kolorów jest dynamicznie generowana zależnie od tego czy włączony jest tryb ciemny

KARTA WYMAGANIA	
Identyfikator	F9885
Priorytet	MUST
Nazwa	Tryb ciemny
Powiązane	F9885
Opis	Aplikacja daje możliwość włączenia trybu ciemnego
Uwagi	• Istnieje możliwość zmiany trybu • Zachowany jest odpowiedni kontrast niezależnie od trybu • Tryb ciemny jest domyślnie włączony

KARTA WYMAGANIA	
Identyfikator	F9898
Priorytet	MUST
Nazwa	Konfiguracja animacji kafelek
Powiązane	F1612
Opis	Istnieje możliwość wyłączenia animacji aktualizacji kafelek
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F9203
Priorytet	MUST
Nazwa	Nadpisywanie powiadomień
Powiązane	F7166
Opis	Istnieje możliwość włączenia funkcji nadpisywania poprzednich powiadomień przez nowe
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F9152
Priorytet	MUST
Nazwa	Stan dashboardu
Powiązane	F4884
Opis	Aplikacja wyświetla aktualny stan dashboardu na ekranie wyświetlającym dany dashboard oraz na ekranie konfiguracji dashboardu
Uwagi	Możliwe stany dashboardu: - DISCONNECTED - FAILED TO CONNECT - ATTEMPTING - CONNECTED

KARTA WYMAGANIA	
Identyfikator	F1463
Priorytet	MUST
Nazwa	Dziennik dashboardu
Powiązane	F9215 | F3461
Opis	Każdy dashboard posiada swój własny dziennik logów
Uwagi	• Dziennik jest dostępny do wyświetlenia poprzez przeciągnięcie górnej części ekranu dashboardu • Przeciągnięcie do końca ekranu powoduje wyczyszczenie wpisów dziennika

KARTA WYMAGANIA	
Identyfikator	F1891
Priorytet	MUST
Nazwa	Tryb edycji ekranu dashboardu
Powiązane	F4821
Opis	Ekran dashboardu posiada tryb edycji
Uwagi	W tym trybie wyświetlany jest pasek narzędzi dających możliwość: - dodania nowych kafelek - usunięcia kafelek - zmiany układu wyświetlania kafelek - przejścia do konfiguracji danej kafelki

KARTA WYMAGANIA	
Identyfikator	F2678
Priorytet	MUST
Nazwa	Tryb edycji ekranu głównego
Powiązane	F5151
Opis	Ekran główny aplikacji posiada tryb edycji
Uwagi	W tym trybie wyświetlany jest pasek narzędzi dających możliwość: - dodania nowych dashboardów - usunięcia dashboardów - zmiany kolejności wyświetlania kart dashboardów - przejścia do konfiguracji danego dashboardu

KARTA WYMAGANIA	
Identyfikator	F5151
Priorytet	MUST
Nazwa	Usuwanie dashboardów
Opis	Tryb edycji ekranu głównego daje dostęp do narzędzia umożliwiającego usuwanie dashboardów
Uwagi	• Narzędzie daje możliwość zaznaczenia kilku kart dashboardów • Ikona narzędzia pulsuje gdy zaznaczona jest przynajmniej jedna karta • Ponowne naciśnięcie ikony narzędzia wyświetla monit z prośbą o potwierdzenie usunięcia zaznaczonych dashboardów

KARTA WYMAGANIA	
Identyfikator	F4821
Priorytet	MUST
Nazwa	Usuwanie kafelek
Opis	Tryb edycji ekranu dashboardu daje dostęp do narzędzia umożliwiającego usuwanie kafelek
Uwagi	• Narzędzie daje możliwość zaznaczenia kilku kafelek • Ikona narzędzia pulsuje gdy zaznaczona jest przynajmniej jedna kafelka • Ponowne naciśnięcie ikony narzędzia wyświetla monit z prośbą o potwierdzenie usunięcia zaznaczonych kafelek

KARTA WYMAGANIA	
Identyfikator	F3109
Priorytet	MUST
Nazwa	Dotacje
Powiązane	F7429
Opis	Aplikacja umożliwia dotację jako formę wsparcia
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F8159
Priorytet	MUST
Nazwa	Wersja profesjonalna
Powiązane	F7429
Opis	Aplikacja umożliwia zakup wersji profesjonalnej pozbawionej ograniczeń
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F8612
Priorytet	MUST
Nazwa	Wersja darmowa
Opis	Aplikacja w wersji darmowej jest ograniczona
Uwagi	Ilość dashboardów jest ograniczona do dwóch

KARTA WYMAGANIA	
Identyfikator	F7429
Priorytet	MUST
Nazwa	Płatności opóźnione
Opis	Aplikacja obsługuje płatności opóźnione
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F9215
Priorytet	MUST
Nazwa	Logowanie wartości kafelek
Opis	Kafelki posiadają możliwość włączenia funkcji logowania nowych wartości do dziennika dashboardu
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F8156
Priorytet	MUST
Nazwa	Personalizacja dashboardów
Opis	Istnieje możliwość ustawienia nazwy, koloru oraz ikony dla dashboardów
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F5780
Priorytet	MUST
Nazwa	Protokół transportowy dla MQTT
Opis	Istnieje możliwość wyboru wykorzystywanego protokołu transportowego dla MQTT
Uwagi	Dostępne protokoły transportowe to: TCP, SSL, WS, WSS

KARTA WYMAGANIA	
Identyfikator	F6612
Priorytet	MUST
Nazwa	Podstawowa konfiguracja klienta MQTT
Opis	Istnieje możliwość podstawowej konfiguracji klienta protokołu MQTT
Uwagi	• Login i hasło są zakrywane po wpisaniu • Konfiguracja połączenia z serwerem MQTT zawiera: - adres serwera - protokół transportowy - port serwera - unikalny identyfikator klienta - keep alive interval - opcjonalne login oraz hasło • W przypadku pozostawienia pustego pola: - id klienta jest losowane - keep alive interval jest ustawiane na 60

KARTA WYMAGANIA	
Identyfikator	F9272
Priorytet	MUST
Nazwa	Konfigruacja protokołu transportowego WebSocket dla MQTT
Opis	Istnieje możliwość dodatkowej konfiguracji przy wykorzystaniu protokołu WebSocket
Uwagi	Zawiera możliwość zdefiniowania query string oraz server path

KARTA WYMAGANIA	
Identyfikator	F3936
Priorytet	MUST
Nazwa	Konfigruacja połączenia szyfrowanego dla MQTT
Opis	Istnieje możliwość dodatkowej konfiguracji przy wykorzystaniu protokołów szyfrowanych takich jak SSL lub WSS
Uwagi	Zawiera możliwość: - zaufania serwerom z certyfikatem selfsigned - ustawienia niestandardowego certyfikatu CA - ustawienia certyfikatu klienta z opcjonalnie szyfrowanym kluczem

KARTA WYMAGANIA	
Identyfikator	F8766
Priorytet	MUST
Nazwa	Karty kafelek
Opis	Aplikacja wyświetla kafelki na ekranie dashboardu
Uwagi	• Przytrzymanie powoduje przejście do konfiguracji danej kafelki • Wyświetlane są na siatce złożonej z: - dwóch kolumn dla orientacji pionowej - czterech kolumn dla orientacji poziomej

KARTA WYMAGANIA	
Identyfikator	F3297
Priorytet	MUST
Nazwa	Karty dashboardów
Opis	Aplikacja wyświetla karty dashboardów na ekranie głównym
Uwagi	• Zawierają ikonę dashboardu oraz jego nazwę • Są w kolorze przypisanym do danego dashboardu • Przytrzymanie powoduje przejście do konfiguracji danego dashboardu • Naciśnięcie powoduje przejscie do danego dashboardu

KARTA WYMAGANIA	
Identyfikator	F4666
Priorytet	MUST
Nazwa	Payload jako JSON
Opis	Kafelki posiadają możliwość włączenia funkcji parsowania odbieranych wiadomości jako JSON
Uwagi	Funkcja jest konfigurowalna oddzielnie dla każdego topic jakiego używa kafelka

KARTA WYMAGANIA	
Identyfikator	F9132
Priorytet	MUST
Nazwa	Flaga retain kafelek
Opis	Kafelki posiadają możliwość ustawienia flagi retain dla wysyłanych wiadomości
Uwagi	Flaga jest konfigurowalna oddzielnie dla każdego topic jakiego używa kafelka

KARTA WYMAGANIA	
Identyfikator	F4100
Priorytet	MUST
Nazwa	QoS kafelek
Opis	Kafelki posiadają możliwość ustawienia QoS dla wysyłanych wiadomości i subskrypcji
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F5163
Priorytet	MUST
Nazwa	Wibracja kafelek
Opis	Naciskanie kafelek powoduje uruchomienie wibracji symulującej fizyczny przycisk
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F1124
Priorytet	MUST
Nazwa	Personalizacja kafelek
Opis	Kafelki mają możliwość personalizacji przez ustawienie ikony, koloru i tagu
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F3461
Priorytet	MUST
Nazwa	Format czasu logów
Opis	Istnieje możliwość wyboru 12h lub 24h formatu czasu wpisów logów dashboardów
Uwagi	Zmiana formatu wpływa jedynie na nowe wpisy

KARTA WYMAGANIA	
Identyfikator	F5157
Priorytet	MUST
Nazwa	Praca w tle
Opis	Aplikacja umożliwia konfigurację pracy ciągłej w tle
Uwagi	• Tryb pracy w tle umożliwia ciągłą pracę aplikacji po jej zamknięciu • Aby włączyć tryb pracy w tle wymagane jest wyłaczenie optymalizacji baterii • Aplikacja uruchamia się bez trybu pracy w tle w przypadku gdy włączona jest optymalizacja baterii

KARTA WYMAGANIA	
Identyfikator	F6742
Priorytet	MUST
Nazwa	Wersja aplikacji
Opis	Użytkownika ma możliwość sprawdzenia wersji aplikacji
Uwagi	• Wyświetlana wersja aplikacji zawiera: - informację czy jest to wersja profesjonalna - numer wersji - informację czy jest to wersja stabilna • Przykłady: - pro | 3.4.0 | stable - free | 3.0.0 | beta

KARTA WYMAGANIA	
Identyfikator	F1146
Priorytet	MUST
Nazwa	Klonowanie konfiguracji dashboardu
Opis	Aplikacja umożliwia odtworzenie już istniejącej konfiguracji innego dashbordu poprzez kopiowanie
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F3321
Priorytet	MUST
Nazwa	Potwierdzenie wysłania
Opis	Kafelki posiadają możliwość włączenia funkcji potwierdzenia przed wysłaniem nowej wartości
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F9703
Priorytet	MUST
Nazwa	Poradnik ekranu dashboardu
Opis	Ekran dashboardu wyświetla poradnik tekstowy w przypadku braku kafelek
Uwagi	• Jest formie pytań i odpowiedzi • Uczy użytkownika jak: - dodawać i usuwać kafelki - zmieniać układ kafelek - przechodzić do konfiguracji kafelek - gdzie znajduje się dziennik dashboardu - jak używać szybkiej nawigacji

KARTA WYMAGANIA	
Identyfikator	F6487
Priorytet	MUST
Nazwa	Poradnik ekranu głównego
Opis	Ekran główny wyświetla poradnik tekstowy w przypadku braku dashboardów
Uwagi	• Jest formie pytań i odpowiedzi • Uczy użytkownika jak: - dodawać i usuwać dashboardy - zmieniać kolejność wyświetlania dashboardów - przechodzić do konfiguracji dashboardów - włączyć funkcję pracy w tle oraz zmienić kolor wiodący - jak zmienić kolor wiodący aplikacji

KARTA WYMAGANIA	
Identyfikator	F8323
Priorytet	MUST
Nazwa	Wyświetlanie nazwy dashboardu
Opis	Aplikacja wyświetla nazwę dashboardu na ekranie wyświetlającym dany dashboard
Uwagi	Nazwa dashboardu jest zawsze wyświetlana dużymi literami

KARTA WYMAGANIA	
Identyfikator	F1655
Priorytet	MUST
Nazwa	Ostatni dashboard
Opis	Istnieje możliwość włączenia funkcji automatycznego przejścia do ostatniego dashboardu zamiast ekranu głównego przy starcie aplikacji
Uwagi	-

KARTA WYMAGANIA	
Identyfikator	F4998
Priorytet	MUST
Nazwa	Import konfiguracji
Opis	Aplikacja przewiduje możliwość importu konfiguracji z pliku
Uwagi	• Istnieje możliwość wgrania pliku z pamięci urządzenia • Po przeprowadzeniu procesu importowania konfiguracji aplikacja sprawdza czy potrzebne są nowe permisje

KARTA WYMAGANIA	
Identyfikator	F4785
Priorytet	MUST
Nazwa	Export konfiguracji
Powiązane	F8612
Opis	Aplikacja przewiduje możliwość eksportu konfiguracji do pliku
Uwagi	• Istnieje możliwość zapisu pliku do pamięci urządzenia • Export zawiera cały trwały stan aplikacji

KARTA WYMAGANIA	
Identyfikator	F7856
Priorytet	MUST
Nazwa	Kafelka button
Powiązane	F4734
Opis	Jedna z dostępnych kafelek to button
Uwagi	• Po naciśnięciu wysyła wiadomość • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki jeżeli nie jest on pusty • Posiada możliwość konfiguracji: - payloadu - publish topic

KARTA WYMAGANIA	
Identyfikator	F3557
Priorytet	MUST
Nazwa	Kafelka slider
Powiązane	F4734 | F8814
Opis	Jedna z dostępnych kafelek to slider
Uwagi	• Umożliwia wysłanie wiadomości z konkretną liczbą z zakresu • Zmienia obecną wartość w przypadku wiadomości będącej liczbą • Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika (patrz powiązane) • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki lub trzy znaki zapytania jeżeli jest on pusty - czas upłynięty od otrzymania ostatniej wiadomości - obecną wartość kafelki • Posiada możliwość konfiguracji: - zakresu liczbowego slidera - kroku slidera - publish/subscribe topic - payloadu z dostępem do flagi **@value** zawierającej obecną wartość kafelki - funkcjonalności umożliwiającej szybkie wysłanie wiadomości poprzez przeciągnięcie kafelki zamiast używania interfejsu pełnoekranowego

KARTA WYMAGANIA	
Identyfikator	F8814
Priorytet	MUST
Nazwa	Kafelka slider - interfejs pełnoekranowy
Opis	Kafelka slider posiada interfejs pełnoekranowy
Uwagi	• Wyświetla suwak oraz obecnie ustawioną liczbę • Umożliwia zmianę obecnego nastawu poprzez przeciągnięcie • Po puszczeniu wysyła wybraną liczbę

KARTA WYMAGANIA	
Identyfikator	F7786
Priorytet	MUST
Nazwa	Kafelka switch
Powiązane	F4734
Opis	Jedna z dostępnych kafelek to switch
Uwagi	• Umożliwia wysyłanie wiadomości w zależności od jednego z dwóch stanów (on/off) • Pełni rolę przełącznika • Zmienia stan w zależności od odebranej wiadomości • Karta kafelki wyświetla: - ikonę kafelki zależną od jej obecnego stanu - tag kafelki jeżeli nie jest on pusty • Posiada możliwość konfiguracji: - off/on payload - publish/subscribe topic - ikony oraz koloru karty kafelki zależenie od stanu

KARTA WYMAGANIA	
Identyfikator	F4450
Priorytet	MUST
Nazwa	Kafelka text
Powiązane	F4734 | F3254
Opis	Jedna z dostępnych kafelek to text
Uwagi	• Umożliwia wysyłanie wiadomości tekstowych • Zmienia obecną wartość w przypadku wiadomości • Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika (patrz powiązane) • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki lub trzy znaki zapytania jeżeli jest on pusty - czas upłynięty od otrzymania ostatniej wiadomości - obecną wartość kafelki • Posiada możliwość konfiguracji: - publish/subscribe topic - trybu pracy (stały payload lub ustalany przy wysłaniu) - payloadu w przypadku trybu pracy ze stałym payloadem - opcji aby kafelka zajmowała wszystkie kolumny dashboardu

KARTA WYMAGANIA	
Identyfikator	F3254
Priorytet	MUST
Nazwa	Kafelka text - interfejs pełnoekranowy
Opis	Kafelka text posiada interfejs pełnoekranowy
Uwagi	• Zawiera pole tekstowe zawierające ustawiony payload wiadomości • Wyświetla przycisk potwierdzenia oraz anulowania • Zawiera informację o docelowym topic

KARTA WYMAGANIA	
Identyfikator	F4400
Priorytet	MUST
Nazwa	Kafelka select
Powiązane	F4734
Opis	Jedna z dostępnych kafelek to select
Uwagi	• Umożliwia wysłanie z listy jednej z predefiniowanych wiadomości tekstowych • Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki jeżeli nie jest on pusty • Posiada możliwość konfiguracji: - publish topic - listy par wartości alias-payload - opcji wyświetlania również payloadu na liście opcji w interfejsie użytkownika

KARTA WYMAGANIA	
Identyfikator	F7673
Priorytet	MUST
Nazwa	Kafelka terminal
Powiązane	F4734 | F9894
Opis	Jedna z dostępnych kafelek to terminal
Uwagi	• Umożliwia wysyłanie wiadomości tekstowych • Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika (patrz powiązane) • Karta kafelki zajmuje dwa rzędy oraz wszystkie kolumny dashboardu • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki lub trzy znaki zapytania jeżeli jest on pusty - czas upłynięty od otrzymania ostatniej wiadomości - historię odebranych wiadomości • Posiada możliwość konfiguracji: - publish/subscribe topic - trybu pracy (stały payload lub ustalany przy wysłaniu) - payloadu w przypadku trybu pracy ze stałym payloadem - opcji aby kafelka zajmowała wszystkie kolumny dashboardu

KARTA WYMAGANIA	
Identyfikator	F9894
Priorytet	MUST
Nazwa	Kafelka terminal - interfejs pełnoekranowy
Opis	Kafelka terminal posiada interfejs pełnoekranowy
Uwagi	• Zawiera pole tekstowe zawierające ustawiony payload wiadomości • Wyświetla przycisk potwierdzenia oraz anulowania • Zawiera informację o docelowym topic

KARTA WYMAGANIA	
Identyfikator	F6252
Priorytet	MUST
Nazwa	Kafelka color picker
Powiązane	F4734
Opis	Jedna z dostępnych kafelek to color picker
Uwagi	• Umożliwia wysłanie koloru korzystając z próbnika • Zmienia obecną wartość jeżeli zawiera ona poprawnie sformatowany payload • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki jeżeli nie jest on pusty • Posiada możliwość konfiguracji: - publish/subscribe topic - payloadu z dostępem do flag zależnych od formatu koloru - formatu koloru (HSV, HEX, RGB) - opcji kolorowania karty kafelki zależnie od obecnego stanu - opcji kolorowania karty kafelki z ignorowaniem kontrastu

KARTA WYMAGANIA	
Identyfikator	F7285
Priorytet	MUST
Nazwa	Kafelka thermostat
Powiązane	F4734 | F7772
Opis	Jedna z dostępnych kafelek to thermostat

Uwagi	• Umożliwia kontrolowanie termostatu:
	- zadanie temperatury
	- opcjonalnie zadanie wilgotności
	- zadanie trybu pracy termostatu
	
	• Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika (patrz powiązane)
	
	• Karta kafelki wyświetla:
	- ikonę kafelki
	- tag kafelki lub trzy znaki zapytania jeżeli jest on pusty
	- czas upłynięty od otrzymania ostatniej wiadomości
	- obecne wartości kafelki:
	  - w formie tekstowej temperaturę i wilgotność
	  - w formie wykresu kołowego temperaturę i wilgotność
	  - w formie wykresu kołowego wartości zadane temperatury i wilgotności
	
	• Posiada możliwość konfiguracji:
	- subscribe topic temperatury
	- subscribe topic wilgotności
	- publish/subscribe topic nastawu temperatury
	- opcjonalnie publish/subscribe topic nastawu wilgotności
	- publish/subscribe topic trybu pracy
	- opcji kontroli nastawu wilgotności
	- kroku slidera nastawu temperatury
	- zakresu liczbowego slidera nastawu temperatury
	- opcjonalnie kroku slidera nastawu wilgotności
	- listy par wartości alias-payload dla trybu pracy termostatu

KARTA WYMAGANIA	
Identyfikator	F7772
Priorytet	MUST
Nazwa	Kafelka thermostat - interfejs pełnoekranowy
Opis	Kafelka thermostat posiada interfejs pełnoekranowy
Uwagi	• Wyświetla przycisk potwierdzenia oraz anulowania • Zawiera przycisk wyboru trybu pracy • Wyświetla obecną wilgotność oraz temperaturę • Wyświetla temperaturę docelową • Wyświetla kołowy suwak nastawu temperatury docelowej • W zależności od ustawień wyświetla wilgotność docelową • W zależności od ustawień wyświetla kołowy suwak nastawu wilgotności docelowej

KARTA WYMAGANIA	
Identyfikator	F8673
Priorytet	MUST
Nazwa	Kafelka time
Powiązane	F4734 | F3635
Opis	Jedna z dostępnych kafelek to time
Uwagi	• Umożliwia wysyłanie wiadomości z godziną lub datą • Zmienia obecną wartość jeżeli zawiera ona poprawnie sformatowany payload • Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika (patrz powiązane) • Karta kafelki wyświetla: - ikonę kafelki - tag kafelki lub trzy znaki zapytania jeżeli jest on pusty - czas upłynięty od otrzymania ostatniej wiadomości - obecną wartość kafelki • Posiada możliwość konfiguracji: - publish/subscribe topic - typu payloadu (czas/data) - payloadu z dostępem do flag **@hour** i **@minute** lub **@day** oraz **@month** i **@year**

KARTA WYMAGANIA	
Identyfikator	F3635
Priorytet	MUST
Nazwa	Kafelka time - interfejs pełnoekranowy
Opis	Kafelka time posiada interfejs pełnoekranowy
Uwagi	• Wyświetla przycisk potwierdzenia oraz anulowania • Wyświetla obecnie ustawioną godzinę lub datę • Interfejs wyboru godziny lub daty

KARTA WYMAGANIA	
Identyfikator	F6385
Priorytet	MUST
Nazwa	Kafelka lights
Powiązane	F4734 | F4287
Opis	Jedna z dostępnych kafelek to lights
Uwagi	• Umożliwia kontrolowanie oświetlenia - włączanie/wyłączanie oświetlenia - ustawienie jasności oświetlenia - opcjonalne zadanie koloru oświetlenia - zadanie trybu pracy oświetlenia • Po naciśnięciu otwiera pełnoekranowy interfejs użytkownika (patrz powiązane) • Karta kafelki wyświetla: - ikonę kafelki zależną od jej obecnego stanu - tag kafelki jeżeli nie jest on pusty • Posiada możliwość konfiguracji: - off/on payload - ikony oraz koloru karty kafelki zależenie od stanu - state publish/subscribe topic - brightness publish/subscribe topic - color publish/subscribe topic - mode publish/subscribe topic - listy par wartości alias-payload dla trybu pracy oświetlenia - opcji kontrolowania koloru oświetlenia - opcjonalnie payloadu koloru z dostępem do flag zależnych od formatu koloru - opcjonalnie formatu koloru (HSV, HEX, RGB) - opcjonalnie opcji kolorowania karty kafelki zależnie od obecnego stanu - opcjonalnie opcji kolorowania karty kafelki z ignorowaniem kontrastu

KARTA WYMAGANIA	
Identyfikator	F4287
Priorytet	MUST
Nazwa	Kafelka lights - interfejs pełnoekranowy
Opis	Kafelka lights posiada interfejs pełnoekranowy
Uwagi	• Wyświetla przycisk potwierdzenia oraz anulowania • Zawiera przycisk przełączający stanu włączenia • Zawiera przycisk wyboru trybu pracy • Wyświetla kołowy suwak nastawu jasności oraz jego wartość • W zależności od ustawień wyświetla interfejs wyboru koloru

05.4. Wykorzystane technologie

Technologia	**Zastosowanie**
Kotlin [14]	Język programowania
Jetpack Compose [13]	Interfejs graficzny
Gradle [10]	Narzędzie automatyzacji kompilacji projektu
Git [8]	Śledzenie zmian w projekcie
Sentry [24]	Monitorowanie aplikacji
Firebase crashlytics [7]	Monitorowanie aplikacji
HiveMQ MQTT Client [11]	Klient MQTT
Android Studio [3]	Środowisko programistyczne
FasterXML Jackson [5]	Biblioteka serializacji
Eclipse Mosquitto [18]	MQTT broker

05.5. Metodyka pracy

Prezentowana aplikacja powstawała na przestrzeni kilku lat.

Głównym celem jaki mi przyświecał była nauka programowania aplikacji mobilnych ale również stworzenie konkurencyjnego rozwiązania umożliwiającego sterowanie urządzeniami IoT z wykorzystaniem protokołu MQTT.

Z założenia precyzyjne wymagania dotyczące projektu miały być ustalone wraz z postępem pracy. W związku z tym zdecydowałem się na stopniowe wdrażanie kolejnych funkcjonalności oraz testowanie ich integracji z całą aplikacją na koniec każdego etapu.

W profesjonalnym środowisku jasno postawione wymagania jakie ma spełniać końcowy produkt oraz twardo określony schemat pracy są kluczem do powtarzalnych sukcesów. Wymaga to nie tylko doświadczenia ze strony zespołu programistów ale również trzymania się szeregu ścisłych norm. Z jednej strony pomaga ustrukturyzować to pracę czy podejmowanie kluczowych decyzji, co jest szczególnie potrzebne przy dużych projektach lub gdzie w projekt zaangażowane jest wiele osób. Z drugiej strony każda kolejna norma czy wytyczna ogranicza nam możliwość adaptacji czy wprowadzania zmian.

Ten kompromis pomiędzy dowolnością a strukturą dającą twarde wytyczne ma miejsce niezależnie od wybranej metodyki. Jedyną różnicą jest stosunek pomiędzy tymi dwoma aspektami. Dla przykładu metodyka waterfall (artykuł Adobe - [27]) daje nam niską elastyczność w zamian za prostolinijność na etapie implementacji. Gdzie metodyka agile (Agile Manifesto - [2]) stawia na luźniejsze podejście do norm ułatwiając wprowadzanie zmian.

Dodatkowo samo wprowadzanie i trzymanie się danej metodyki pracy jest kosztem samy w sobie, często wymagającym zatrudnienia dodatkowego pracownika którego czas poświęcony jest jedynie temu.

Biorąc pod uwagę założone cele oraz fakt, że była to praca jednoosobowa, musiałbym zmierzyć się z minimalnymi benefitami i wysokimi kosztami w innym przypadku. Brak narzuconej metodyki pracy dawał mi pełną swobodę w wprowadzaniu zmian, co umożliwiało eksperymentowanie oraz bezproblemowe wycofywanie się z błędnych decyzji.

06. Faza implementacji

W tym rozdziale opisano część implementacji aplikacji. Rozwiązania i koncepty dotyczą systemu operacyjnego Android. Fragmenty kodu zostały uproszczone na potrzeby tego rozdziału. Między innymi, dodano komentarze w języku polskim oraz pominięto część implementacji.

06.1. Architektura aplikacji

Architektura aplikacji została zaprojektowana w oparciu o model z jedną główną aktywnością która pełni rolę centralnego punktu zarządzania nawigacją oraz cyklem życia aplikacji. W aplikacji istnieje także aktywność dedykowana do obsługi płatności, co pozwala na oddzielenie logiki związanej z transakcjami od głównych funkcji aplikacji. Za poszczególne ekrany interfejsu użytkownika odpowiadają fragmenty.

06.2. Zarządzanie stanem aplikacji

Aplikacja implementuje klasę `Application` z przeznaczeniem na przechowywanie stanu aplikacji oraz zarządzaniem procesem inicjalizacji po uruchomieniu. Jest to specjalna klasa której instancję system tworzy przed jakąkolwiek inną klasą w aplikacji. Klasa ta ma cykl życia niezależny od cyklu życia aktywności, trwający przez cały czas pracy aplikacji. Dzięki temu stan aplikacji jest niewrażliwy na wszelkie zmiany w konfiguracji które powodują ponowne uruchomienie aktywności.

06.3. Foreground service

`Service` to komponent aplikacji, który potrafi wykonywać długotrwałe operacje w tle nawet po tym gdy użytkownik przejdzie do innej aplikacji. `Foreground service` jest to specjalny rodzaj usługi działającej na pierwszym planie wykonującej operacje zauważalne dla użytkownika. Usługa tego typu ma obowiązek wyświetlać stałe powiadomienie aby informować użytkownika o procesie działającym w tle. W nowszych wersjach Androida powiadomienie to może być odrzucone przez użytkownika. Przykładem takiej usługi jest odtwarzacz multimedialny z interfejsem w formie powiadomienia wyświetlanego gdy jakaś aplikacja odtwarza w tle muzykę.

Połączenie odpowiedniej konfiguracji z użyciem `Foreground service` zmniejsza szanse na zabicie aplikacji przez system. Tym sposobem aplikacja może działać w tle po tym gdy użytkownik ją zamknie. Jest to kluczowa funkcjonalność z racji tego że MQTT jest protokołem komunikacyjnym w czasie rzeczywistym.

Niestety praca w tle aplikacji jest obłożona szeregiem obostrzeń w imię wydłużenia czasu pracy na baterii. Dla przykładu jeżeli aplikacja obsługuje Android w wersji 14 wzwyż, deweloper jest zobowiązany do zadeklarowania typu `Foreground service`. (dokumentacja - [6]) Pula tych typów jest ograniczona, a każdych z nich ma swoje ściśle unormowane zastosowanie. Istnieje również typ zarezerwowany dla specjalnych przypadków których pozostałe typy nie pokrywają. Wybranie tego typu wiąże się z obowiązkiem podania szczegółowego wytłumaczenia, jak i również przygotowania nagrania tego w jaki sposób nasza aplikacja wykorzystuje usługę oznaczoną tym typem. Mimo spełnienia tych wymogów w dalszym ciągu możemy spotkać się z odmową przy próbie publikacji aplikacji na platformie Google Play.

06.4. Daemons

Aplikacja w początkowych założeniach miała obsługiwać więcej niż jeden protokół komunikacyjny. Rozwiązania wprowadzone na podstawie tego wymagania stanowią fundamentalną część implementacji. W związku z powyższym, mimo tego że wymaganie to zostało jednak na ten moment porzucone, dalej ma ono wpływ na obecną architekturę aplikacji.

06.4.1. Daemon

Demon (ang. daemon) to termin używany w informatyce, który odnosi się do programu działającego w tle, zazwyczaj bez interakcji z użytkownikiem. Może on wykonywać różne zadania, takie jak monitorowanie systemu, zarządzanie zasobami czy obsługa żądań sieciowych.

W kontekście przedstawianej aplikacji jest to abstrakcyjna klasa nadrzędna po której dziedziczą wszystkie inne demony. Pełnią one rolę źródła informacji i zdarzeń. Każdy dashboard ma przypisany dedykowany demon danego rodzaju, zależnie od tego z wykorzystaniem jakiego protokołu się on komunikuje. Rolą tego konceptu jest odseparowanie części architektury odpowiedzialnej za komunikację.

Poniżej opisano niektóre części klasy `Daemon`.

```
//Czy demon został zwolniony z użytku
//Demon zostaje zwolniony wtedy gdy nie jest już dłużej używany
//Na przykład gdy dashboard do którego jest przypisany zostaje usunięty
var isDischarged = false

//Czy demon jest włączony/czy pracuje
protected abstract val isEnabled: Boolean

//Specjalne obserwowalne pole
//Jej wartość zmienia się w momencie gdy zmienia się stan demona
//Poszczególne części aplikacji mogą nasłuchiwać nowych wartości
//na tym polu w związku z czym być powiadamiane o zmianie stanu demona
abstract val statePing: MutableLiveData<String?>

//Stan demona
//Konkretna implementacja tego pola jest zależna od rodzaju demona
abstract val state: Any
```

```
//Metoda uruchamiana w momencie gdy demon jest przydzielany do dashboardu
open fun notifyAssigned() {
    isDischarged = false
}

//Metoda uruchamiana w momencie gdy demon jest zwalniany z użytku
open fun notifyDischarged() {
    isDischarged = true
}

//Metoda uruchamiana w momencie zmiany konfiguracji demona
//np. zmiana adresu serwera, portu, itp.
open fun notifyConfigChanged() {}
```

Dodatkowo klasa `Daemon` posiada `companion object` definiujący `invoke operator`.

W Kotlinie `companion object` jest to miejsce gdzie zdefiniowane są metody i właściwości które są dostępne bez tworzenia instancji klasy.
Natomiast `invoke operator` pozwala na wywołanie obiektu jako funkcji.

```
companion object {
    operator fun invoke(context: Context, dashboard: Dashboard, type: Type) =
    when (type) {
        Type.MQTTD -> Mqttd::class
        Type.BLUETOOTHD -> Bluetoothd::class
    }.constructors.first().call(context, dashboard)
}
```

Użycie w kodzie `Daemon()` zamiast ”tworzyć instancję” abstrakcyjnej klasy `Daemon` uruchamia funkcję `invoke` która zwraca instancję klasy danego typu demona.

06.4.2. Daemonized

Interfejs który implementuje jedynie klasa `Tile`. Powstał on w celu odseparowania tej części kafelki dla przejrzystości.

Zawiera on już zaimplementowane metody `send` oraz `receive`. Metoda `send` służy do wysyłania nowych wiadomości. Natomiast metoda `receive` używana jest do przekazania przychodzącej wiadomości do kafelki.

Dodatkowo istnieją metody `onSend` oraz `onReceive` wykorzystywane przez te wyżej wspomniane. Obie z przeznaczeniem bycia zaimplementowanym przez docelowy element.

06.4.3. Mqttd

Klasa pełniąca rolę demona dla dashboardów komunikujących się z wykorzystaniem protokołu MQTT. Zawiera następujące pola:

```
//Asynchroniczny klient MQTT
private var client: Mqtt5AsyncClient? = null

//Obecna konfiguracja
private var currentConfig = MqttConfig()

//Lista topic która obecnie subskrybuje client
private var topics: MutableList<Pair<String, Int>> = mutableListOf()

//Pola dziedziczone po klasie Deamon

public override val isEnabled
    get() = d.mqtt.isEnabled && !isDischarged

private val isConnected
    get() = client?.config?.state?.isConnected ?: false

override val statePing: MutableLiveData<String?> = MutableLiveData(null)

override val state: State
    get() = if (manager.isWorking) State.ATTEMPTING
    else try {
        if (!isConnected) State.DISCONNECTED
        else if (currentConfig.sslRequired && !currentConfig.sslTrustAll) {
            State.CONNECTED_SSL
        } else State.CONNECTED
    } catch (e: Exception) {
        Debug.recordException(e)
        State.FAILED
    }
```

`Mqttd` korzysta z `StatusManager`. Jego zasada działania opisana jest w dalszym rozdziale.

```
inner class Manager : StatusManager() {
    override fun isStable(): Boolean {
        return isConnected == isEnabled && (currentConfig == d.mqtt || !isEnabled)
    }

    override fun makeStable() {
        if (isEnabled) {
            if (isConnected) {
                Debug.log("MQTT_DISCONNECT")
                client?.disconnect()
            } else {
                if (currentConfig != d.mqtt || client == null) {
                    Debug.log("MQTT_BUILD_NEW_CLIENT")
                    buildClient(d.mqtt.copy())
                }
                Debug.log("MQTT_CONNECT")
                client?.connect()
            }
        } else {
            Debug.log("MQTT_CLOSE")
            client?.disconnect()
        }
    }

    override fun onJobDone() = statePing.postValue("")
    override fun onJobStart() = statePing.postValue(null)
    override fun onException(e: Exception) {
        super.onException(e)
        when (e) {
            is IllegalArgumentException -> statePing.postValue(e.message)
        }
    }
}
```

Kluczową częścią klasy `Mqttd` jest funkcja `buildClient`. Bierze ona `MqttConfig` jako parametr i na jego podstawie buduje odpowiedniego klienta.

```
fun buildClient(config: MqttConfig) {
    var client = Mqtt5Client.builder()
        .identifier(config.clientId)
        .serverHost(config.address)
        .serverPort(config.port)
        .addConnectedListener {
            Debug.log("MQTT_ON_CONNECTED")
            topicCheck()
            statePing.postValue("")
        }
        .addDisconnectedListener {
            Debug.log("MQTT_ON_DISCONNECTED")
            topics = mutableListOf()
            manager.dispatch(reason = "connection")

            if (it.cause !is ConnectionFailedException) statePing.postValue("")
            else statePing.postValue(it.cause.cause?.message)
        }

    //W razie potrzeby konfiguruje autoryzację
    if (config.includeCred) client = client.simpleAuth()
        .username(config.username)
        .password(config.pass.toByteArray())
        .applySimpleAuth()

        //W razie potrzeby konfiguruje użycie protokołu WebSocket
        if (config.protocol == Protocol.WS || config.protocol == Protocol.WSS) {
            client = client
                .webSocketConfig()
                .queryString(config.queryString)
                .serverPath(config.serverPath)
                .applyWebSocketConfig()
        }

    //W razie potrzeby konfiguruje szyfrowanie
    //Metoda setupSSL jest opisana niżej
    if (config.sslRequired) client = client.setupSSL(config)

    //Buduje klienta i aktualizuje obecny config
    this.client = client.buildAsync()
    currentConfig = config
}
```

Poniżej znajduje się metoda odpowiedzialna za konfigurację:

- szyfrowania połaczenia oraz użycia certyfikatów i kluczy klienta
- związaną z wykorzystaniem niestandardowego urzędu certyfikacji (CA)
- związaną z opcją pominięcia weryfikacji certyfikatu serwera

```
private fun Mqtt5ClientBuilder.setupSSL(config: MqttConfig): Mqtt5ClientBuilder {
    //-------------------------------------------------------------------------
    //Tworzy magazyn kluczy z certyfikatem i kluczem klienta
    val kmfStore = KeyStore.getInstance(KeyStore.getDefaultType())
    val kmfKeyPassword = UUID.randomUUID().toString().toCharArray()
    kmfStore.load(null, null)
    kmfStore.setCertificateEntry("cc", config.clientCert)
    config.clientKey?.let {
        kmfStore.setKeyEntry(
            "k",
            it.private,
            kmfKeyPassword,
            arrayOf<Certificate?>(config.clientCert)
        )
    }
    //-------------------------------------------------------------------------
    //Tworzy magazyn kluczy używany do weryfikacji serwera
    //Opcjonalne dodawany jest certyfikat niestandardowego urzędu certyfikacji
    //Ma to na celu dodanie wsparcia dla self-signed certyfikatów serwera
    //zachowując jednocześnie bezpieczeństwo przed atakami typu MITM
    var tmfStore: KeyStore? = null
    if (config.caCert != null) {
        tmfStore = KeyStore.getInstance(KeyStore.getDefaultType())
        tmfStore.load(null, null)
        tmfStore.setCertificateEntry("c", config.caCert)
    }

    //KeyManager decyduje które dane uwierzytelniające
    //powinny zostać wysłane do hosta zdalnego
    val kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm())
    kmf.init(kmfStore, kmfKeyPassword)

    val alg = TrustManagerFactory.getDefaultAlgorithm()

    //TrustManager określa czy zdalne połączenie powinno być uznawane za zaufane
    val tmf = if (!config.sslTrustAll) TrustManagerFactory.getInstance(alg)
    else TrustAllTrustManagerFactory()

    if (!config.sslTrustAll) tmf.init(tmfStore)

    return this.sslConfig()
        .keyManagerFactory(kmf)
        .trustManagerFactory(tmf)
        .applySslConfig()
}
```

Metody związane z wiadomościami:

```
//Metoda używana do publikacji wiadomości
fun publish(topic: String, msg: String, qos: Int = 0, retain: Boolean = false)

//Metody służące do zarządzania subskrypcjami
private fun subscribe(topic: String, qos: Int)
private fun unsubscribe(topic: String, qos: Int)

//Buduje listę topic które powinny być subskrybowane
//Porównuje tą listę z obecnie subskrybowanymi topic
//W przypadku różnic korzysta z metod subscribe i/lub unsubscribe
private fun topicCheck()
```

06.4.4. MqttConfig

Klasa zawierająca konfigurację dashboardu dla `Mqttd`.
Zawiera następujące pola:

```
var isEnabled: Boolean
var sslTrustAll: Boolean
@JsonIgnore
var caCert: X509Certificate?
var caFileName: String
@JsonIgnore
var clientCert: X509Certificate?
var clientFileName: String
@JsonIgnore
var clientKey: KeyPair?
var keyFileName: String
var address: String
var protocol: Mqttd.Protocol
var port: Int
var keepAlive: Int
var includeCred: Boolean
var queryString: String
var serverPath: String
var username: String
var pass: String
var clientId: String

//Pole z getterem
//Wartość jest generowana na podstawie pola protocol
val sslRequired: Boolean

//Pola z setterem
//Przypisanie wartości powoduje wygenerowanie certyfikatów
var caCertStr: String?
var clientCertStr: String?
var clientKeyStr: String?
```

Wartość części pól jest generowana na postawie wartości innego pola. Na przykład wartość pola `caCert` jest generowana na podstawie wartości pola `caCertStr`. Z tego powodu część pól jest oznaczona anotacją `@JsonIgnore`. Ma ona za zadanie wyłączyć te pola z procesu serializacji.

06.4.5. MqttDaemonizedConfig

Klasa zawierająca konfigurację danej kafelki dla `Mqttd`.
Zawiera następujące pola:

```
var isEnabled: Boolean
var lastReceive: Date?

//Mapa topic używanych przez kafelkę
//Kluczami są funkcje w obrębie kafelki
//Większość kafelek posiada jedną funkcję (np. button) więc i jedną parę topic
//Część jest bardziej skomplikowana (np. lights) i posiada wiecej takich par
val subs: MutableMap<String, String>
val pubs: MutableMap<String, String>

//Kafelki posiadają możliwość parsowania payload jako json
//To pole zawiera mapę ścieżek json dla każdej funkcji w obrębie kafelki
val jsonPaths: MutableMap<String, String>

//Pole zawierające payload jaki ma być wysłany w zależności od funkcji kafelki
var payloads: MutableMap<String, String>
var qos: Int

//Część kafelek ma możliwość dynamicznego ustawienia payload przed wysłaniem
//To pole zawiera informację czy payload jest statyczny
var payloadIsVar: Boolean

//Czy payload ma być parsowany jako json
var payloadIsJson: Boolean

//Czy wyświetlić komunikat z prośbą o potwierdzenie przed wysłaniem
var doConfirmPub: Boolean

var doRetain: Boolean
var doLog: Boolean
var doNotify: Boolean
var silentNotify: Boolean
var notifyTitle: String
var notifyPayload: String
```

06.5. Klasy

06.5.1. Dashboard

Kluczowa klasa całej implementacji. Jak nazwa wskazuje ma reprezentować dashboard. Dziedziczy po klasie `RecyclerViewItem`. Dla uproszczenia dziedziczone pola i metody zostały wyłączone z opisu poniżej.
Więcej na ten temat znajduje się w dziale **Recycler View**.

```
var name: String = ""

//Typ dashboardu
var type: Daemon.Type = Daemon.Type.MQTTD

//Klucz ikony dashboardu
var iconKey = "il_interface_plus_circle"

//Ikona dashboardu
//Wartość tego pola jest generowana przy każdym użyciu na podstawie pola iconKey
val iconRes: Int

//Kolor ikony dashboardu
var hsv: FloatArray

//Paleta kolorów ikony dashboardu
//Wartość tego pola jest generowana przy każdym użyciu na podstawie pola hsv
//Więcej na ten temat znajduje się w dziale Theme
val pallet: Theme.ColorPallet

//Dziennik logów dashboardu
var log = Log()

//Konfiguracja dla Mqttd
var mqtt = MqttConfig()

@JsonIgnore
//Demon przypisany do tego dashboardu
//Proces przypisywania demonów opisany jest w działach DaemonsManager oraz Setup
var daemon: Daemon? = null

//Lista kafelek jakie posiada dashboard
var tiles: MutableList<Tile>
```

06.5.2. Tile

Bazowa klasa nadrzędna dla wszystkich kafelek. Po tej klasie dziedziczą pozostałe klasy reprezentujące już konkretny rodzaj kafelki (np. `ButtonTile`). Podobnie jak dashboard dziedziczy ona po klasie `RecyclerViewItem` oraz oczywiście implementuje interfejs `Daemonized`. Dla uproszczenia dziedziczone pola i metody zostały wyłączone z opisu poniżej.

Klasa `Tile` zawiera następujące metody i pola:

```
@JsonIgnore
//Pole określające ile slotów ma zajmować kafelka
//Przeznaczone jest ono do ewentualnego nadpisania
//Przykładem kafelki która to robi jest kafelka thermostat
open var height = 1f

//Opcjonalny tag/nazwa kafelki
//Wyświetla się w interfejsie kafelki na ekranie dashboardu
var tag = ""

//Typ kafelki
//Pole przeznaczone do nadpisana przez klasy dziedziczące
//override var typeTag = "button"
abstract var typeTag: String

//Pola związane z kolorem i ikoną kafelki
//Tożsame pola opisane są w klasie Dashboard
abstract var iconKey: String
val iconRes: Int
var hsv: FloatArray
val pallet: Theme.ColorPallet

//Metoda uruchamiana w momencie tworzenia kafelki
//Dokładnie wtedy gdy dodawana jest ona jako nowa do dashboardu
open fun onCreateTile() {}

//Metoda uruchamiana cyklicznie dla każdej kafelki co sekundę
//Aktualizuje ona informację kiedy ostatnio kafelka odebrała wiadomość
//Informacja ta wyświetla się w interfejsie kafelki na ekranie dashboardu
//Przykład: 10 sekund temu
fun updateTimer()
```

06.5.3. Settings

Prosta klasa z przeznaczeniem do przechowywania ustawień aplikacji.
Zawiera następujące pola:

```
//Obecna wersja aplikacji
var version = BuildConfig.VERSION_CODE
//Czy wysyłane powiadomienia mają być układanie w stos
var notifyStack = true
//Czy ukryć przyciski nawigacyjne
var hideNav = false
//ID ostatnio odwiedzonego dashboardu
var lastDashboardId = 0L
//Format czasu wpisów w logach dashboardów
var militaryTime = true
//Czy przechodzić od razu do ostatniego dashboardu przy starcie
var startFromLast = true
//W jakim stanie użytkownik zostawił sekcję komunikacja we właściwościach kafelek
//Przechowuje informację czy sekcja ma być rozwinięta
var mqttTabShow = true
//Czy animować kafelki przy przyjściu nowej wiadomości
var animateUpdate = true
//Czy aplikacja oczekuje na jakąś płatność opóźnioną
var pendingPurchase = false
//Czy dozwolono na pracę w tle po zamknięciu aplikacji
var fgEnabled = false
```

06.5.4. Log

Klasa pełniąca rolę dziennika logów. Pole jej typu posiada klasa `Dashboard`.
Zawiera następujące pola i metody:

```
//Lista wpisów w dzienniku
val list: MutableList<LogEntry>

//Metoda dodająca nowy wpis
fun newEntry(text: String)

//Metoda czyszcząca dziennik
fun flush()
```

Wpisy w dzienniku są typu `LogEntry`. Klasa ta dziedziczy po `RecyclerViewItem`.
Poza dziedziczonymi metodami i polami zawiera następujące pola:

```
//Wartość wpisu
var text: String

//Czas dodania wpisu w formie tekstowej
val time: String
```

06.5.5. App

Klasa dziedzicząca po klasie `Application`. Jest to specjalna klasa której instancję system tworzy przed jakąkolwiek inną klasą w aplikacji. Klasa ta ma cykl życia niezależny od cyklu życia aktywności, trwający przez cały czas pracy aplikacji. Dzięki temu stan aplikacji jest niewrażliwy na wszelkie zmiany w konfiguracji które powodują ponowne uruchomienie aktywności.

```
class AtomApp : Application() {
    companion object {
        //Globalny stan aplikacji
        //Więcej na ten temat w kolejnym dziale
        val aps = AtomAppState()

        //Globalne odniesienie do aplikacji
        lateinit var app: AtomApp
    }

    //Metoda uruchamiana w momencie startu aplikacji
    override fun onCreate() {
        super.onCreate()
        app = this
        //Uruchomienie procesu inicjalizacji aplikacji
        Setup.initialize()
    }
}
```

06.5.6. AppState

Klasa zawierająca stan aplikacji. Zawiera następujące pola i metody:

```
//Czy aplikacja jest licencjonowana
//Czy użytkownik wykupił wersję profesjonalną
var isLicensed = false

//Czy aplikacja przeszła już proces inicjalizacji
var isInitialized = MutableLiveData(false)

//Obiekty związane ze stanem aplikacji
var settings = Settings()
var theme = Theme()
var dashboards = mutableListOf<Dashboard>()

//Obecnie wybrane obiekty
//Dashboard i kafelka w którym obecnie jest użytkownik
//Te wartości mają między innymi związek z ekranami właściwości tych elementów
lateinit var dashboard: Dashboard
lateinit var tile: Tile
var dashboardIndex = -2

//Ścieżka do głównego folderu aplikacji
//Jest to folder dedykowany dla danej aplikacji
//Inne aplikacje nie mają do niego dostępu
//Wartość przypisywana jest w procesie inicjalizacji aplikacji
var rootFolder: String = ""

//Ścieżki do plików serializowanych obiektów
//Wartość przypisywana jest w procesie inicjalizacji aplikacji
//Więcej na temat serializacji znajduje się w dziale Storage
var path: Map<KClass<out Any>, String> = mapOf()

//Metody zmieniająca obecnie wybrany dashboard
fun setCurrentDashboard(index: Int): Boolean
fun setCurrentDashboard(id: Long): Boolean
```

06.5.7. Theme

Klasa odpowiedzialna za dynamiczny motyw aplikacji wszędzie gdzie wykorzystany jest XML do tworzenia interfejsu użytkownika. Zawiera wewnętrzną klasę `Artist` generującą monochromatyczną paletę kolorów na podstawie koloru wybranego przez użytkownika.

Wygenerowana paleta składa się z następujących kolorów:

- **color**: kolor bazowy
- **background**: kolor tła
- **a, b, c, d**: kolejne kolory akcentowe zbliżające się do koloru tła

Klasa `Artist` zawiera następujące metody i pola:

```
//Czy włączony jest tryb ciemny
//Zmiana wartości pola powoduje ponowne wygenerowanie palety
var isDark: Boolean

//Wybrany przez użytkownika kolor
//Zmiana wartości pola powoduje ponowne wygenerowanie palety
var hsv: FloatArray

@JsonIgnore
//Wygenerowana paleta
var pallet: ColorPallet

//Metoda generującą paletę
fun getColorPallet(
    hsv: FloatArray,
    isAltCon: Boolean = false,
    isRaw: Boolean = false
): ColorPallet
```

Proces generowania palety przebiega w kilku krokach:

1. Generowany jest kolor tła. Nie jest on zależny od koloru źródłowego.
   Ma on oddzielną wartość dla trybu ciemnego i dla trybu jasnego.

2. Na podstawie koloru źródłowego generowany jest kolor bazowy. Jeżeli paleta `isRaw` to kolor bazowy jest równy kolorowi źródłowemu.
   W innym przypadku jest to kolor źródłowy ograniczony w jak najmniejszym stopniu, ale na tyle aby spełniał wymóg kontrastu.

3. Generowane są kolory akcentowe poprzez mieszanie ich z kolorem tła. Stosunek mieszania jest taki sam dla każdego koloru. Ma on oddzielną wartość dla trybu ciemnego i oddzielną wartość dla trybu jasnego:

   - **a**: kolor bazowy z kolorem tła
   - **b**: kolor **a** z kolorem tła
   - **c**: kolor **b** z kolorem tła
   - **d**: kolor **c** z kolorem tła

Klasa `Theme` zawiera następujące metody i pola:

```
val artist = Artist()

companion object {
  //Wartość tych pól pobierana jest z obiektu artist
  val colors: ComposeColorPallet
  val isDark: Boolean
}

//Metoda mająca za zadanie ustawić motyw aplikacji dla danej grupy widoków
//Uruchamiana jest rekurencyjnie tak aby zadziałała dla każdego widoku w drzewie
private fun ViewGroup.applyTheme(p: ColorPallet) {
    for (i in 0 until this.childCount) {
        this.getChildAt(i).let {
            if (it is ViewGroup) it.applyTheme(p)
            it.defineType(p)
        }
    }

    this.defineType(p)
}

//Uruchamiaja docelową metodę odpowiedzialną za stylizację danego elementu
private fun View.defineType(p: ColorPallet) {
    when (this) {
        is RadioButton -> this.applyTheme(p)
        is Chip -> this.applyTheme(p)
        is RecyclerView -> this.applyTheme(p)
        is CircularSeekBar -> this.applyTheme(p)
        //itp.
    }
}

//Docelowe metody stylizujące
//Operują na wartości tag danego widoku aby odróżnić różne wersje stylistyczne
//Tag to dowolna wartość przypisana do widoku przez programistę
private fun FrameLayout.applyTheme(p: ColorPallet)
private fun ConstraintLayout.applyTheme(p: ColorPallet)
private fun LinearLayout.applyTheme(p: ColorPallet)
//itp.
```

06.6. Obiekty pomocnicze

W Kotlinie `object` to specjalny typ, który pozwala na tworzenie singletonów, czyli klas, które mają tylko jedną instancję. Umożliwia to łatwe tworzenie obiektów, które nie wymagają jawnego tworzenia instancji za pomocą konstruktora.

Ten dział jest poświęcony obiektom pomocniczym wykorzystanym w implementacji. Mają one na celu ustrukturyzowanie oraz uproszczenie pewnych funkcjonalności w kodzie.

06.6.1. Debug

Obiekt odpowiedzialny za centralizację wykorzystania narzędzi używanych do śledzenia błędów w aplikacji.

```
object Debug {
    //Dodaje dodatkowe niestandardowe informacje do raportu
    fun log(payload: String) {
        Sentry.addBreadcrumb(payload)
    }

    //Przesyła raport w przypadku wystąpienia błedu
    fun recordException(exception: Exception) {
        Sentry.captureException(exception)
    }
}
```

06.6.2. DialogBuilder

Obiekt ułatwiający tworzenie okien dialogowych dla interfejsu używającego XML.

```
object DialogBuilder {
    //Wyświetla dialog z pytaniem o potwierdzenie zadanej akcji
    inline fun Context.buildConfirm(
        message: String,
        label: String,
        textSize: Float = 20f,
        textAlign: Int = TEXT_ALIGNMENT_CENTER,
        crossinline onDeny: () -> Unit = {},
        crossinline onConfirm: () -> Unit
    )

    //Dostosowuje interfejs to wyświetlania dialogu
    //Między innymi przyciemnia to co wyświetla się za dialogiem
    fun Dialog.dialogSetup()
}
```

06.6.3. Pro

Obiekt służący do zarządzania licencjonowaniem aplikacji. Licencja służy jako weryfikacja, że użytkownik wykupił wersję płatną, bez ponownego korzystania z usług Google Play. Licencjonowanie polega na utworzenie pustego pliku w folderze aplikacji. Pomimo tego, że w teorii użytkownik nie ma dostępu do tego folderu, nie jest to duże utrudnienie. Zdeterminowany użytkownik ominie również te bardziej zaawansowane. Dlatego to wprowadzone ma jedynie w prosty sposób podstawowo przeciwdziałać omijaniu zabezpieczeń, nie uprzykrzając doświadczenia pozostałym.

```
object Pro {
    //Sprawdza czy plik licencji istnieje
    fun getLicenceStatus(): Boolean

    //Tworzy plik licenji
    fun createLocalLicence()

    //Usuwa plik licencji
    fun removeLocalLicence()
}
```

06.6.4. Icons

Obiekt grupujący ikony wykorzystywane w aplikacji.
Posiada pole `cats` będące listą dostępnych kategorii ikon oraz pole `icons` będące listą obiektów typu `Icon`.

```
data class Icon(
    val res: Int = 0, //ID zasobu
    val type: String = "", //Typ ikony
    val cat: String = "" //Kategoria ikony
)
```

06.6.5. Storage

Obiekt odpowiedzialny za serializację obiektów. Wykorzystuje mapę `path` zawierającą informację gdzie zapisywać obiekty danych klas.

```
aps.path = mapOf(
    Theme::class to "${aps.rootFolder}/theme",
    Settings::class to "${aps.rootFolder}/settings",
    Dashboard::class to "${aps.rootFolder}/dashboards"
)
```

Jego działanie opiera się wykorzystanie uniwersalnych generycznych metod umożliwiających serializację i deserializację dowolnych obiektów lub kolekcji tych obiektów. Jedynym warunkiem jest istnienie odpowiedniego wpisu w mapie `path`.

Metody pomocnicze i mapper wykorzystywany w obiekcie.

```
//Mapper dostarczany przez bibliotekę Jackson
val mapper: ObjectMapper

//Metoda serializująca obiekt dowolnej klasy do String
fun Any.prepareSave(): String = mapper.writeValueAsString(this)

//Metoda zwracająca zapisany obiekt danej klasy z pliku
inline fun <reified T> getSave() = try {
    FileReader(aps.path[T::class]).readText()
} catch (_: Exception) {
    ""
}
```

Metody służące serializacji obiektów.

```
//Metoda zapisująca obiekt dowolnej klasy do pliku
fun Any.saveToFile(save: String = this.prepareSave()) {
    try {
        val path = aps.path[this::class]
        File(path!!).writeText(save)
    } catch (_: Exception) {
    }
}

//Metoda zapisująca kolekcję obiektów dowolnej klasy do pliku
inline fun <reified T> Collection<T>.saveToFile(save: String = this.prepareSave()) {
    try {
        val path = aps.path[T::class]
        File(path!!).writeText(save)
    } catch (_: Exception) {
    }
}
```

Metody służące deserializacji obiektów.

```
//Metoda deserializująca obiekt dowolnej klasy z pliku
inline fun <reified T> parseSave(save: String = getSave<T>()): T? =
    try {
        mapper.readValue(save, T::class.java)
    } catch (_: Exception) {
        null
    }

//Metoda deserializująca kolekcję obiektów dowolnej klasy z pliku
inline fun <reified T> parseListSave(save: String = getSave<T>()): MutableList<T> =
    try {
        mapper.readerForListOf(T::class.java).readValue(save)
    } catch (_: Exception) {
        mutableListOf()
    }
```

W praktyce wykorzystanie powyższych metod wygląda następująco.

```
//Proces serializacji obiektu typu dashboard
val dashboard = Dashboard()
dashboard.saveToFile()

//Proces deserializacji obiektu typu dashboard
//Jeżeli typ pola jest określony metoda sama "domyśli się"
//jaki ma być zwrócony typ bez dodatkowych oznaczeń
dashboard = Storage.parseSave() ?: Dashboard()

//Alternatywnie
val theme = Storage.parseSave<Theme>() ?: Theme()
```

06.6.6. Setup

Obiekt zawierający metody odpowiedzialne za kolejne etapy procesu inicjalizacji aplikacji.

```
object Setup {
    //Główna metoda rozpoczynająca inicjalizację
    fun initialize() {
        Debug.log("INIT")
        setFilesPaths()
        initializeBasicGlobals()

        CoroutineScope(Dispatchers.Default).launch {
            updateProStatus()
            checkBilling()
            checkBatteryStatus()
            configureForegroundService()
            initializeOtherGlobals()
            assignDaemons()
            finish()
        }
    }

  //Konfiguruje zmienne aps.rootFolder oraz aps.path
  private fun setFilesPaths()
  //Inicjalizuje zmienne globalne takie jak
  //aps.dashboard; aps.tile; aps.theme; aps.settings
  private fun initializeBasicGlobals()
  //Aktualizuje status licencji
  private fun updateProStatus()
  //Sprawdza czy nie ma przeprocesowanych płatności opóźnionych
  private fun checkBilling()

  //Sprawdza czy włączona jest optymalizacja baterii
  //Jeżeli tak to wyłącza pracę w tle
  private fun checkBatteryStatus()
```

Kontynuacja na drugiej stronie

```
    //Konfiguruje użycie foreground service
    //Zatrzymuje ją jeśli nie jest potrzebna
    //W innym przypadku uruchamia ją
    private suspend fun configureForegroundService() {
        //Usługa nie włączona ale działa
        if (!aps.settings.fgEnabled && service?.isStarted == true) {
            DaemonsManager.dischargeAll()
            ForegroundService.stop(app)
        }

        //Usługa jest włączona ale nie uruchomiona
        else if (aps.settings.fgEnabled && service?.isStarted != true) {
            //Zwalnia demony na wypadek jeżeli któryś pracuje
            DaemonsManager.dischargeAll()

            //Uruchamia usługę i czeka na jej start
            ForegroundService.start(app)
            ForegroundService.haltForService()
        }
    }

    //Konfiguruje pozostałe zmienne globalne
    private fun initializeOtherGlobals()

    //W zależności czy praca w tle jest włączona
    //Uruchamia demony z kontekstem aplikacji lub usługi
    private fun assignDaemons() {
        val context: Context = if (!aps.settings.fgEnabled) app
        else service!!

        DaemonsManager.assignAll(context)
    }

    private fun finish() = aps.isInitialized.postValue(true)
}
```

06.7. Managers

Za wyjątkiem `DaemonsManager` (object) są to klasy dedykowane do zarządzania określonymi zasobami oraz procesami. Ich użycie ma na celu centralizację logiki oraz zmniejszenie redundancji w kodzie.

06.7.1. BillingManager

Klasa służąca do procesowania zakupów oraz płatności wewnątrz aplikacji z wykorzystaniem Google Play. Do zarządzania połączeniem używa `StatusManager`. Każda z kluczowych metod jest nieblokująca za sprawą wykorzystania `suspendCoroutine`.

Klasa `BillingManager` zawiera następujące metody i pola:

```
val context: Context

//Czy jest jest włączony/czy pracuje
internal var isEnabled = false

//Klient dostarczany przez bibliotekę płatności
internal lateinit var client: BillingClient

private val manager = Manager()

companion object {
    //ID produktów
    var PRO = "atom_dashboard_pro"
    var DON1 = "atom_dashboard_don1"
    var DON5 = "atom_dashboard_don5"
    var DON25 = "atom_dashboard_don25"
}

fun enable() {
    isEnabled = true
    manager.dispatch(reason = "enable")
}

fun disable() {
    isEnabled = false
    manager.dispatch(reason = "disable")
}

//Buduje klienta płatności
internal fun createClient()

//Metody pomocnicze
private fun Purchase.acknowledge()
private fun Purchase.consume()

//Obsługuje potwierdzenie zakupu
//Przydziela zakupione produkty
fun onPurchased(purchase: Purchase)

//Wyświetla komunikaty związane z końcem procesu zakupu lub
//powiadamia o płatnościach opóźnionych
fun onPurchaseProcessed(purchase: Purchase)
```

```kotlin
//Pobiera informacje na temat produktów
private suspend fun getProductDetails(id: String): MutableList<ProductDetails>?

//Pobiera listę zakupionych produktów
suspend fun getPurchases(timeout: Long = 2000): MutableList<Purchase>?

//Pobiera ceny produktów
suspend fun getPriceTags(ids: List<String>): Map<String, String>?

//Uruchamia proces zakupu
suspend fun lunchPurchaseFlow(id: String)

//Pobiera listę zakupionych produktów według filtru
//Czas wykonywania wynosi minimum wartości eta
suspend inline fun checkPurchases(
    eta: Long = 10000,
    filter: (Purchase) -> Boolean = { !it.isAcknowledged },
    onDone: (List<Purchase>?) -> Unit = {}
)

inner class Manager : StatusManager(100) {
    override fun isStable(): Boolean = when (client.connectionState) {
        CONNECTED -> isEnabled
        CONNECTING -> false
        DISCONNECTED, CLOSED -> !isEnabled
        else -> true
    }

    override fun makeStable() {
        if (client.connectionState == CLOSED) createClient()
        else if (!isEnabled) client.endConnection()
        else if (client.connectionState != CONNECTING) {
            client.startConnection(object : BillingClientStateListener {
                override fun onBillingSetupFinished(billingResult: BillingResult) {}
                override fun onBillingServiceDisconnected() {}
            })
        }
    }

    //Wstrzymuje pracę do momentu ustabilizowania połączenia
    suspend fun awaitDone(timeout: Long = 5000): Boolean =
        withTimeoutOrNull(timeout) {
            while (!isStable()) delay(100)
            return@withTimeoutOrNull client.isReady
        } ?: false
}
```

06.7.2. FragmentManager

Klasa zarządzająca nawigacją w aplikacji. Posiada następujące pola oraz metody:

```
//Stos fragmentów
var backstack = mutableListOf<Fragment>(MainScreenFragment())

//Obecnie wyświetlany fragment
private var currentFragment: Fragment = MainScreenFragment()

//Jeżeli zwraca prawdę to użycie systemowego przycisku/gestu cofania
//nie powoduje cofnięcia do poprzedniego fragmentu
var doOverrideOnBackPress: () -> Boolean = { false }

//Animacje tranzycji pomiędzy fragmentami
object Animations {
    val swap: (FragmentTransaction) -> Unit
    val fade: (FragmentTransaction) -> Unit
    val slideLeft: (FragmentTransaction) -> Unit
    val slideRight: (FragmentTransaction) -> Unit
    val fadeLong: (FragmentTransaction) -> Unit
}

//Zmienia obecnie wyświetlany fragment
fun replaceWith(
    fragment: Fragment,
    stack: Boolean = true,
    animation: ((FragmentTransaction) -> Unit?)? = swap
)

//Cofa do poprzedniego fragmentu
//Zwraca fałsz jeżeli stos fragmentów jest pusty
fun popBackstack(
    stack: Boolean = false,
    animation: ((FragmentTransaction) -> Unit?)? = swap
): Boolean
```

06.7.3. StatusManager

Abstrakcyjna klasa mająca za zadanie periodyczne uruchamianie metody `makeStable` od momentu wywołania metody `dispatch` do momentu gdy metoda `isStable` zwróci prawdę.

Zawiera następujące pola:

```
//Częstotliwość wywołania metody makeStable
private val interval: Long = if (BuildConfig.DEBUG) 1500 else 300

//Czy jest w trybie debugowania
private val debug: Boolean = false

private var job: Job? = null
var isWorking = false
```

Metody wykonywane w kluczowych momentach:

```
open fun onJobStart() {}
open fun onJobDone() {}
open fun onException(e: Exception) {}
```

Metody abstrakcyjne:

```
abstract fun isStable(): Boolean
abstract fun makeStable()
```

Główna metoda `dispatch`. Jej ponowne wywoływanie nie ma wpływu na pracę. Umożliwia opcjonalny restart obecnie uruchomionej pracy.

```
fun dispatch(cancel: Boolean = false, reason: String = "") {
    if (debug) Debug.log("SM_DISPATCH [$reason]")

    if (cancel) {
        if (debug) Debug.log("SM_CANCEL_JOB")
        job?.cancel()
    }

    if (job != null && job?.isActive == true) return

        job = GlobalScope.launch(Dispatchers.IO) {
        if (debug) Debug.log("SM_JOB_LAUNCH")
        try {
            //Pierwsza iteracja
            if (!isStable()) {
                isWorking = true
                onJobStart()
                makeStable()
                delay(interval)
            }

            //Pozostałe iteracje
            while (!isStable()) {
                makeStable()
                delay(interval)
            }

            isWorking = false
            onJobDone()
            if (debug) Debug.log("SM_SETTLE")
        } catch (e: Exception) {
            onException(e)
            delay(interval)
            dispatch(true)
        }
    }
}
```

06.7.4. DaemonsManager

Obiekt odpowiedzialny za przydzielanie oraz zwalnianie demonów. Posiada następujące metody:

```
fun assignAll(context: Context) = aps.dashboards.forEach { assign(it, context) }

fun dischargeAll() {
    Debug.log("DM_DISCHARGE_ALL")
    aps.dashboards.forEach { discharge(it) }
}

fun assign(dashboard: Dashboard, context: Context) = try {
    Debug.log("DM_ASSIGN")

    dashboard.apply {
        daemon = Daemon(context, this, type)
        daemon?.notifyAssigned()
    }
} catch (e: Exception) {
    Debug.recordException(e)
}

fun discharge(dashboard: Dashboard) = try {
    Debug.log("DM_DISCHARGE")
    dashboard.daemon?.notifyDischarged()
} catch (e: Exception) {
    Debug.recordException(e)
}
```

06.7.5. ToolbarManager

Klasa definiująca działanie paska narzędzi dostępnego z poziomu interfejsu dashboardu. Pełni rolę warstwy abstrakcji, tak aby uprościć implementację fragmentu odpowiedzialnego za tą część interfejsu użytkownika.

06.8. Interfejs użytkownika

06.8.1. XML

Znacząca część interfejsu użytkownika została zaimplementowana z wykorzystaniem Jetpack Compose. Nie całość, ponieważ niestety pewne funkcjonalności ekranów używających `RecyclerView` byłyby trudne do odtworzenia wykorzystując tą technologię zamiast XML.

`RecyclerView` jest to narzędzie dostępne jedynie z XML. Umożliwia ono tworzenie dynamicznych list które procesują jedynie elementy obecnie wyświetlane na ekranie.

Posiada ono również stosunkowo łatwą do implementacji możliwość obsługi gestu zmiany kolejności wyświetlanych elementów (ang. drag and drop) wraz z towarzyszącym temu animacjami.

Samodzielna implementacja takiej funkcjonalności przy listach jednowymiarowych nie stanowi problemu. Jednak sprawa znacząco się komplikuje w przypadku list dwuwymiarowych, gdzie elementy mają różne rozmiary, tak jak to jest w tym przypadku.

Z tego powodu, ekran główny wyświetlający listę dashboardów oraz ekran wyświetlający sam dashboard z dwuwymiarową listą kafelek wykorzystują XML, a co za tym idzie również interfejsy samych kafelek oraz dialogi wyświetlane na tych ekranach.

06.8.2. Fragmenty

Za każdy ekran aplikacji jest odpowiedzialny odrębny fragment:

DashboardFragment	ekran dashboardu
DashboardPropertiesFragment	ekran właściwości dashboardu
LoadingFragment	ekran ładowania
MainScreenFragment	ekran główny z listą dashboardów
SettingsFragment	ekran ustawień
ThemeFragment	ekran konfiguracji motywu
TileIconFragment	ekran konfiguracji ikony kafelki
TileNewFragment	ekran dodawania nowej kafelki
TilePropertiesFragment	ekran konfiguracji kafel

06.8.3. RecyclerView

Jest to narzędzie dostępne jedynie z XML. Umożliwia ono tworzenie dynamicznych list które procesują jedynie elementy obecnie wyświetlane na ekranie. Posiada ono również stosunkowo łatwą do implementacji możliwość obsługi gestu zmiany kolejności wyświetlanych elementów (ang. drag and drop) wraz z towarzyszącym temu animacjami. W przedstawianej aplikacji `RecyclerView` jest wykorzystywany między innymi do wyświetlania listy dashboardów na ekranie głównym oraz do wyświetlania siatki kafelek na ekranie dashboardu. W skład implementacji `RecyclerView` wchodzą trzy klasy opisane w kolejnych rozdziałach.

06.8.4. RecyclerViewAdapter

Generyczna klasa dziedzicząca po `ListAdapter`. Rolą adaptera jest zarządzanie wyświetlaną listą.

Poza metodami dziedziczonymi `RecyclerViewAdapter` został rozszerzony o:

1. Użytkowe funkcje lambda wywoływane w kluczowych momentach

```
var onBindViewHolder: (item, ViewHolder, Int) -> Unit = { _, _, _ -> }
var onItemClick: (item) -> Unit = {}
var onItemLongClick: (item) -> Unit = {}
var onItemRemoved: (item) -> Unit = {}
var onItemMarkedRemove: (Int, Boolean) -> Unit = { _, _ -> }
var onItemEdit: (item) -> Unit = {}
```

2. Metody pomocnicze

```
//Oznacza element jako ten do usunięcia
//Zmienia go wizualnie sygnalizując to użytkownikowi
fun markItemRemove(position: Int)

//Usuwa oznaczone przez użytkownika elementy listy
fun removeMarkedItems()

//Usuwa element z określonej pozycji listy
fun removeItemAt(pos: Int, notify: Boolean = true)
```

3. Rozszerzoną obsługę trybu edycji

```
var editMode = Modes()
open inner class Modes {
    //Obecny tryb
    private var mode = -1

    //Funkcja lambda wywoływana w momencie zmiany trybu
    var onSet: (Modes) -> Unit = {}

    //Czy znajduje się w danym trybie
    val isNone: Boolean
    val isSwap: Boolean
    val isRemove: Boolean
    val isEdit: Boolean

    //Metody ustawiające określony tryb edycji
    fun setNone()
    fun setSwap()
    fun setRemove()
    fun setEdit()
}
```

06.8.5. RecyclerViewItem

Abstrakcyjna klasa po której dziedziczy każda klasa która jednocześnie reprezentuje elementy danej listy w aplikacji. Jedną z takich klas jest klasa `Tile`.

W skład tej klasy wchodzą następujące metody i pola:

```
//Unikatowy identyfikator elementu
var id: Long

//Identyfikator zasobu XML będącego interfejsem graficznym elementu
abstract val layout: Int

//Odniesienie do ViewHolder danego elementu
@JsonIgnore
var holder: RecyclerViewAdapter.ViewHolder? = null

//Adapter do którego jest przypisany element
@JsonIgnore
var adapter: RecyclerViewAdapter<*>? = null

//Metody związane z RecyclerView

open fun onCreateViewHolder(
    parent: ViewGroup,
    viewType: Int
): RecyclerViewAdapter.ViewHolder

open fun onViewAttachedToWindow(holder: RecyclerViewAdapter.ViewHolder)

open fun onBindViewHolder(
    holder: RecyclerViewAdapter.ViewHolder,
    position: Int
)

//Metoda odpowiedzialna za stylizację elementu
open fun onSetTheme(holder: RecyclerViewAdapter.ViewHolder)

//Metody reagujące na zdarzenia związane z elementem
open fun onTouch(v: View, e: MotionEvent)
open fun onClick(v: View, e: MotionEvent)
open fun onEdit(isEdit: Boolean)
```

Dodatkowo klasa `RecyclerViewItem` posiada system flag określający w jakim stanie znajduje się element.

```
@JsonIgnore
var flag = Flags()

inner class Flags {
    //Obecna flaga
    private var flag = -1

    //Czy dana flaga jest ustawiona
    val isNone
    val isRemove

    //Ustawia daną flagę
    fun setNone() = setFlag(-1)
    fun setRemove() = setFlag(1)

    //Aktualizuje wizualnie element w zależności od flagi
    private fun show()
}
```

06.8.6. ItemTouchCallback

Ostatnia klasa wchodząca w skład implementacji `RecyclerView`. Odpowiada ona za gesty, animacje oraz zmiany kolejności elementów listy.

06.8.7. Wykrywanie gestów

Wykrywanie globalnych gestów w aplikacji opiera się o własną implementację `ConstraintLayout` oraz `LinearLayout`, tak aby uzyskać dostęp do metody `onInterceptTouchEvent` umożliwiającej przechwycenie zdarzeń związanych z dotykiem. Wykorzystanie tych klas jako `root layout` interfejsu użytkownika pozwoliło na implementację dowolnych gestów w aplikacji.

07. Faza testów końcowych i publikacji

07.1. Testy końcowe

07.1.1. Testy manualne

Testy manualne polegały na długoterminowym korzystaniu z aplikacji w różnych scenariuszach użytkowania. Skupiono się na następujących aspektach:

Interfejs użytkownika: Ocena intuicyjności i responsywności interfejsu, w tym łatwości nawigacji oraz dostępności kluczowych funkcji.

Scenariusze obejmowały:

- Sprawdzanie czy użytkownik może łatwo znaleźć potrzebne informacje.
- Testowanie reakcji aplikacji na różne rozdzielczości i orientacje ekranu.

Stabilność: Obserwacja zachowania aplikacji w dłuższych sesjach użytkowania, w tym sprawdzanie, czy aplikacja nie ulega awariom ani nie zawiesza się.

Scenariusze obejmowały:

- Śledzenie stanu aplikacji długo pracującej w tle.
- Dodawanie wielu jednocześnie pracujących dashboardów.
- Testowanie aplikacji w warunkach słabego połączenia internetowego.
- Obserwowanie jak aplikacja radzi sobie z utratą połączenia z internetem.
- Sprawdzanie czy aplikacja utrzymuje połączenie z brokerem.
- Testowanie czy aplikacja utrzymuje stałe połączenie z brokerem.
- Sprawdzanie czy aplikacja wznawia połączenie z brokerem po jego utracie.

Wydajność: Monitorowanie czasu ładowania danych oraz reakcji na interakcje użytkownika, co miało na celu ocenę płynności działania aplikacji.

Scenariusze obejmowały:

- Pomiar czasu ładowania aplikacji.
- Testowanie szybkości reakcji aplikacji na różne interakcje użytkownika, takie jak przewijanie, klikanie przycisków, czy wprowadzanie danych.
- Analiza zużycia zasobów systemowych podczas normalnego użytkowania.
- Sprawdzanie płynności list przy dużej ilości wyświetlanych elementów.

07.1.2. Testy z różnymi konfiguracjami serwera MQTT

W celu oceny kompatybilności aplikacji z różnymi konfiguracjami serwera MQTT, przeprowadzono testy z użyciem różnych protokołów oraz ustawień:

Obsługa różnych protokołów: Testowano aplikację z serwerami obsługującymi protokoły MQTT 3.1, 3.1.1 oraz 5.0, aby zweryfikować, czy aplikacja prawidłowo interpretuje i przetwarza wiadomości w każdym z tych standardów.

Obsługa różnych form uwierzytelniania: Testowano czy aplikacja poprawnie obsługuje wszystkie warianty oraz kombinacje uwierzytelniania: za pośrednictwem certyfikatów klienta, a także z wykorzystaniem hasła i loginu.

Obsługa certyfikatów self-signed: Testowano czy aplikacja poprawnie łączy się z serwerami skonfigurowanymi przy użyciu certyfikatów self-signed.

07.2. Publikacja na platformie Google Play

Aplikacja jest dostępna do pobrania na platformie Google Play pod nazwą **Atom dashboard - MQTT and IoT**. Na dzień 01.12.2024 aplikacja posiada **358** unikalnych użytkowników.

Poniżej znajduje się lista najpopularniejszych krajów:

Indonezja(25)	
Rosja(31)	
Niemcy(27)	
Brazylia(19)	
Stany Zjednoczone(20)	
Indie(16)	
Korea Południowa(13)	
Wielka Brytania(11)	
Malezja(13)	

08. Faza utrzymania

08.1. Napotkane trudności

08.1.1. Praca w tle

Jedną z większych napotkanych trudności było przystosowanie aplikacji do pracy w tle. Powiązane to jest z skomplikowaną naturą zasad na podstawie których Android zarządza cyklem życia aplikacji i jak to na nią dokładnie wpływa. Obecnie jest duży nacisk na wydłużanie czasu pracy na baterii urządzeń mobilnych, stąd z nieprzychylnością patrzy się na aplikacje długo pracujące w tle. Obecnie, aplikacja przy włączaniu tej funkcji wymaga od użytkownika wykluczenia jej z procesu optymalizacji czasu pracy na baterii. W teorii ma to przeciwdziałać sytuacji gdzie system zabija aplikację, jednak nie jest to gwarantowane rozwiązanie.

08.1.2. Foreground Service

W sierpniu 2024 Google wprowadziło wymaganie dotyczące określenia typu usługi działającej na pierwszym planie dla aplikacji kierowanych na system Android w wersji 14 wzwyż. Same typy są ściśle unormowane, część z wymaga zadeklarowania użycia specyficznych uprawnień. Konkretny wybór musi być poparty argumentami, a błąd w tej materii może zaowocować zablokowaniem naszej aplikacji na platformie Google Play. Niestety, żadne z tych dostępnych nie pasowały w tym przypadku, jedyną alternatywą pozostało użycie typu `specialUse`. To z kolei wymaga załączenia dokładnego wytłumaczenia tej decyzji w manifeście aplikacji wraz z nagraniem filmu, przesyłanego w momencie publikacji na platformie, w jaki sposób aplikacja wykorzystuje usługę pierwszoplanową.

08.1.3. Klient MQTT

Pierwszym wykorzystywanym klientem był ten dostarczany przez Eclipse Paho. Niestety nie jest ona już aktywnie wspierana przez twórców od czterech lat. W związku z tym nie obsługuje już ona MQTT w wersji 5 oraz co gorsze, jest niekompatybilna z nowszymi wersjami Androida. Pierwszym rozwiązaniem było naprawianie błędów powodujących tą niekompatybilność. Było to jednak rozwiązanie krótkoterminowe i na ten moment aplikacja została przepisana tak aby wykorzystywać asynchronicznego klienta dostarczanego przez HiveMQ.

08.1.4. Rzadkie błędy

Dużą trudnością są błędy w aplikacji które występują rzadko, w warunkach ciężkich do odtworzenia lub po długotrwałym używaniu aplikacji. Jednym z rozwiązań żeby zapobiegać występowaniu takich błędów jest użycie systemów zdalnego raportowania takich jak Firebase Crashlytics lub Sentry. Obecnie aplikacja używa obu takich systemów równolegle. Sentry jako ten charakteryzujący się lepszą telemetryką używany jest do śledzenia błędów i innych zdarzeń kluczowych a Firebase jest wykorzystywany z kolei do celów statystycznych.

08.1.5. Serializacja

Serializacja danych w aplikacji wielokrotnie powodowała nawracające błędy:

1. Problem z serializacją klas polimorficznych (np. lista kafelek)
Rozwiązań tego problemu jest wiele, często są one skomplikowane. Obecnie jest on rozwiązany prostym oznaczeniem klasy `Tile` odpowiednią anotacją mającą za zadanie automatyczne dodanie odpowiedniego pola zawierającej typ klasy przy serializacji.

2. Problem z deserializacją wynikający z braku odpowiedniego konstruktora
Jest to problem wynikający ze sposobu w jaki dane są deserializowane. W bazowej konfiguracji, na początku tego procesu tworzona jest instancja danej klasy. Z tego powodu musi być dostępny konstruktor klasy bez parametrów.

3. Problemy z deserializacją przy zmianach w projekcie
Jest to kluczowy aspekt wpływający na działanie aplikacji. W wyniku zmian w kodzie aplikacji, zmiany nazw pól lub zmiany pakietu, w jakim znajduje się dana klasa, mogą wystąpić problemy z deserializacją danych z poprzednich wersji aplikacji. Z oczywistych względów jest to poważny problem mający duży wpływ na użytkownika końcowego.

08.1.6. Weryfikacja konta Google Play Console

W roku 2023 Google ogłosiło nadchodzącą wymaganą weryfikację kont deweloperów na platformie Google Play Console. Dla kont personalnych wiąże się to z udostępnieniem wrażliwych danych takich jak: miejsce zamieszkania, imię, nazwisko, prywatne dane kontaktowe. Podawane informacje są weryfikowane przez przesłanie państwowych dokumentów, rachunków lub umów wynajmu. Niespełnienie tych wymagań wiąże się z blokadą konta.

08.2. Plany ulepszenia

08.2.1. Jetpack Compose

Wykorzystanie przestarzałego już XML do tworzenia interfejsu użytkownika wiąże się z dużymi ograniczeniami. Konwersja w pełni na Jetpack Compose przyniosła by wiele uproszeń, z tego powodu jest to jedno z ważniejszych celów w planach na ulepszenie projektu.

08.2.2. Protokoły

Aplikacja została napisana z myślą o przyszłe rozszerzenie o inne protokoły. Dla przykład, użycie Bluetooth umożliwiło by bezpośrednią komunikację z sterowanymi urządzeniami. Na ten moment ma to jednak niski priorytet z racji skomplikowania.

08.2.3. Testy automatyczne

Dodanie automatycznych testów wykluczających błędy na skutek wprowadzonych zmian lub aktualizacji dependencji aplikacji. Takie testy usprawniłyby i ustandaryzowały proces testowania przy publikacji nowej wersji aplikacji.

08.2.4. Dokumentacja

Stworzenie dokładnej dokumentacji jak korzystać z aplikacji jest jednym z kluczowych celów na ten moment.

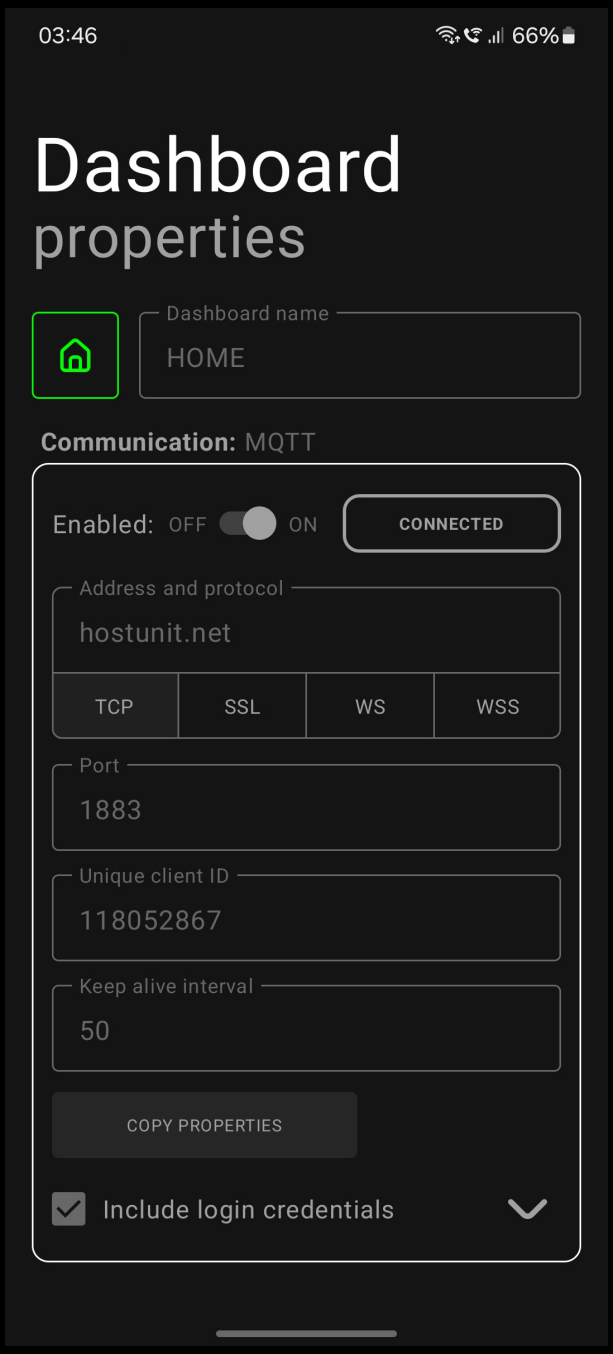
08.2.5. Wersja na system iOS

Planowane jest stworzenie wersji aplikacji na urządzenia z systemem iOS.

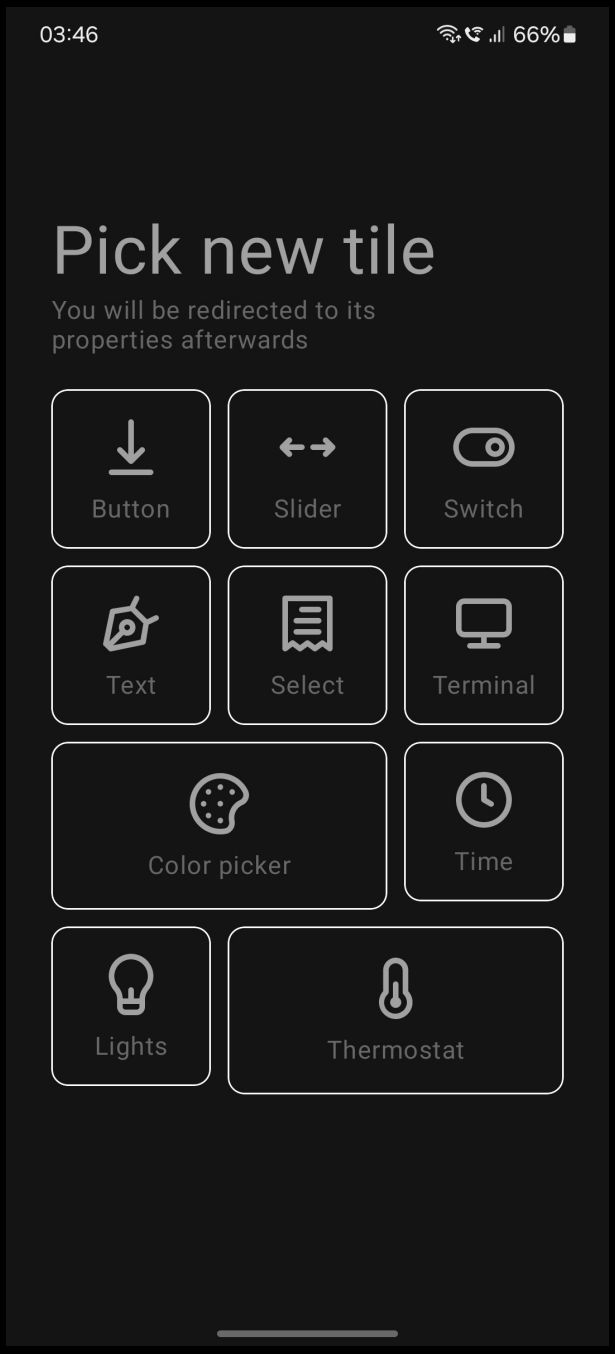
09. Prezentacja

Ten dział pracy poświęcony jest prezentacji części efektu końcowego, przedstawione zrzuty ekranu nie reprezentują wszystkich funkcjonalności.

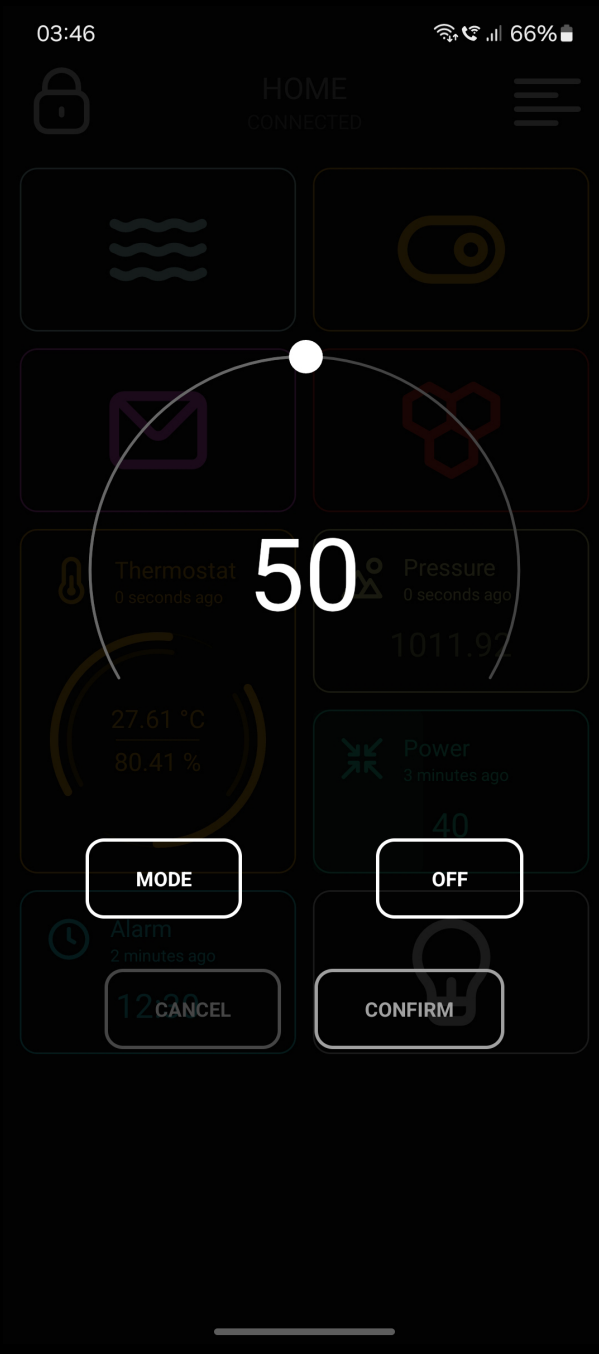
Rys. 5. Konfiguracja ikony elementu



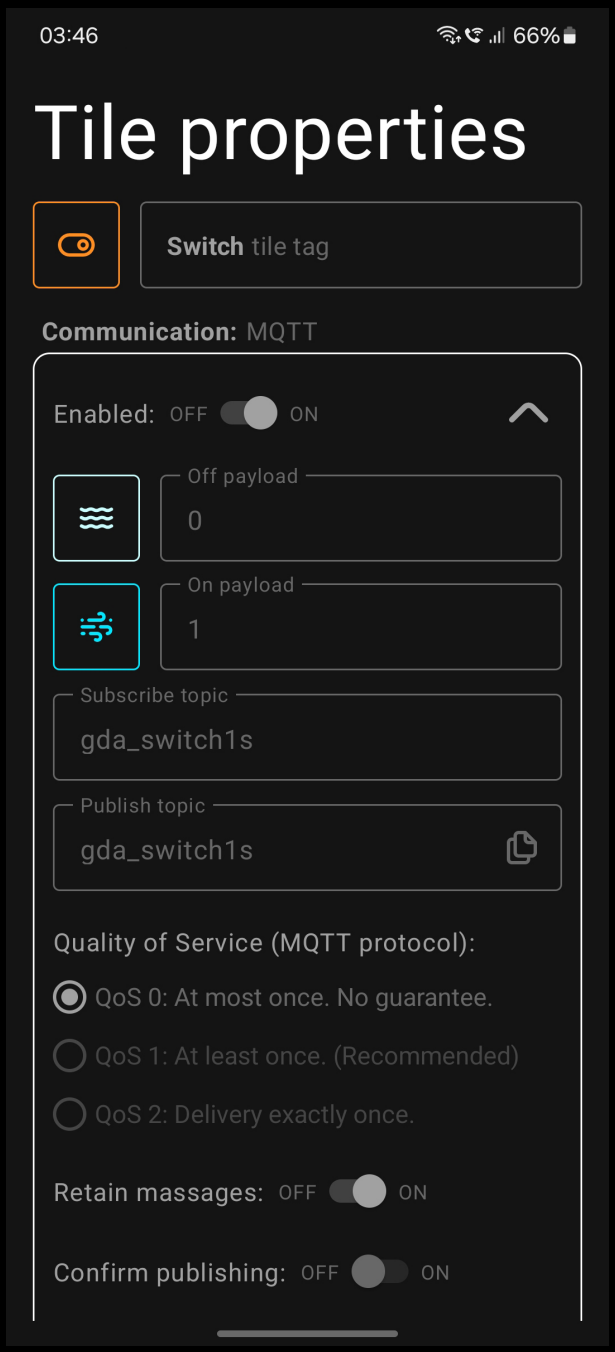
Rys. 6. Konfiguracja dashboardu



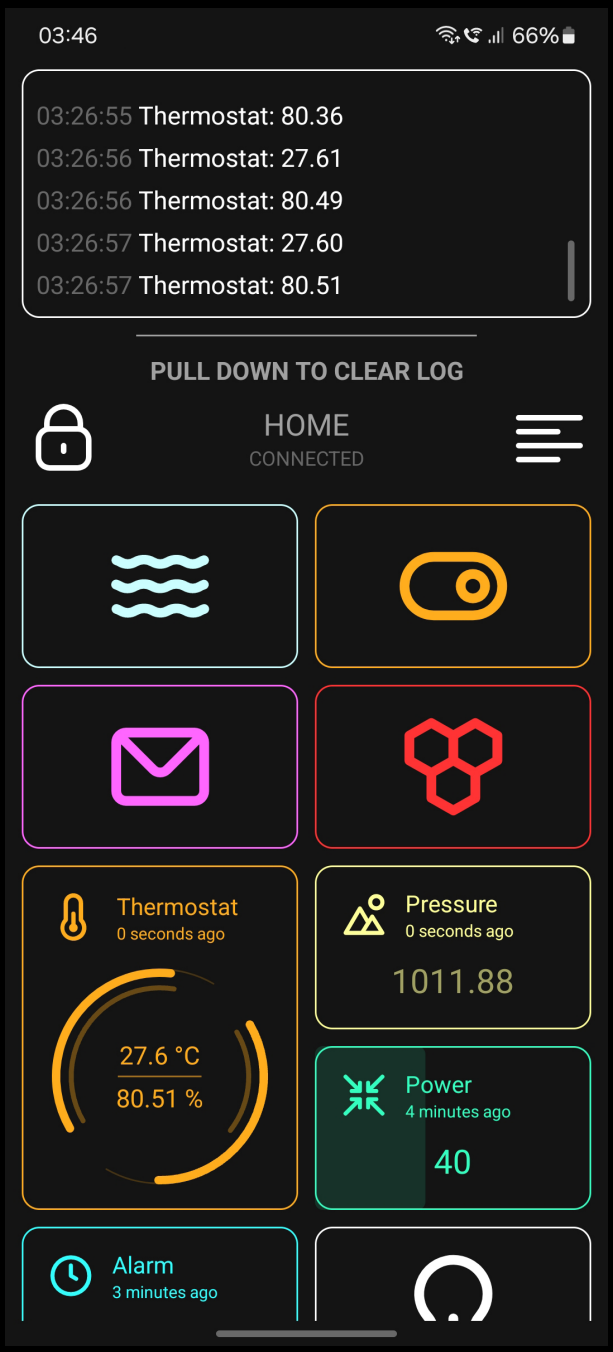
Rys. 7. Wybór nowej kafelki



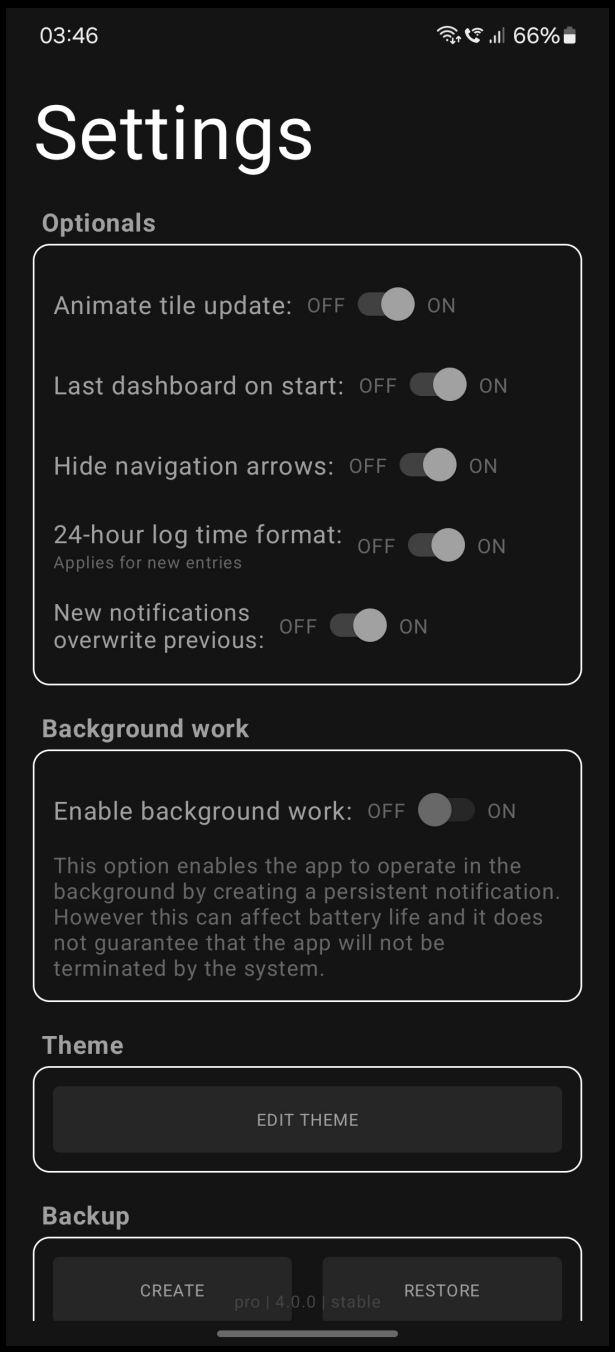
Rys. 8. Interfejs kafelki `Lights`



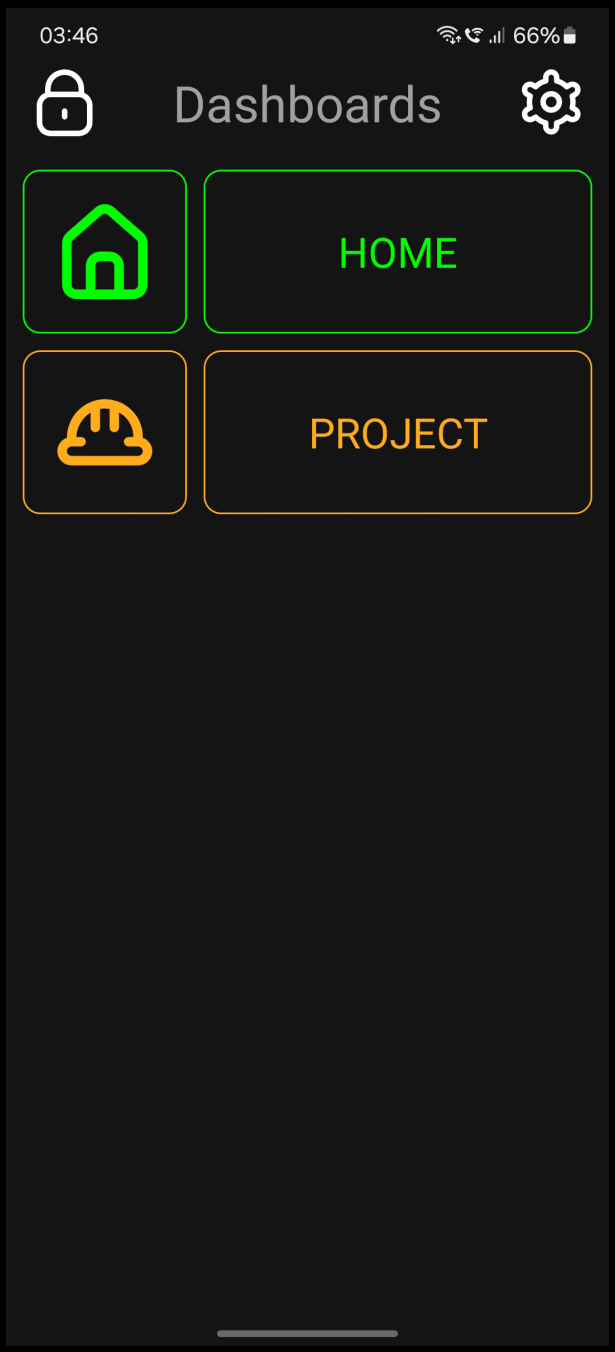
Rys. 9. Konfiguracja kafelki



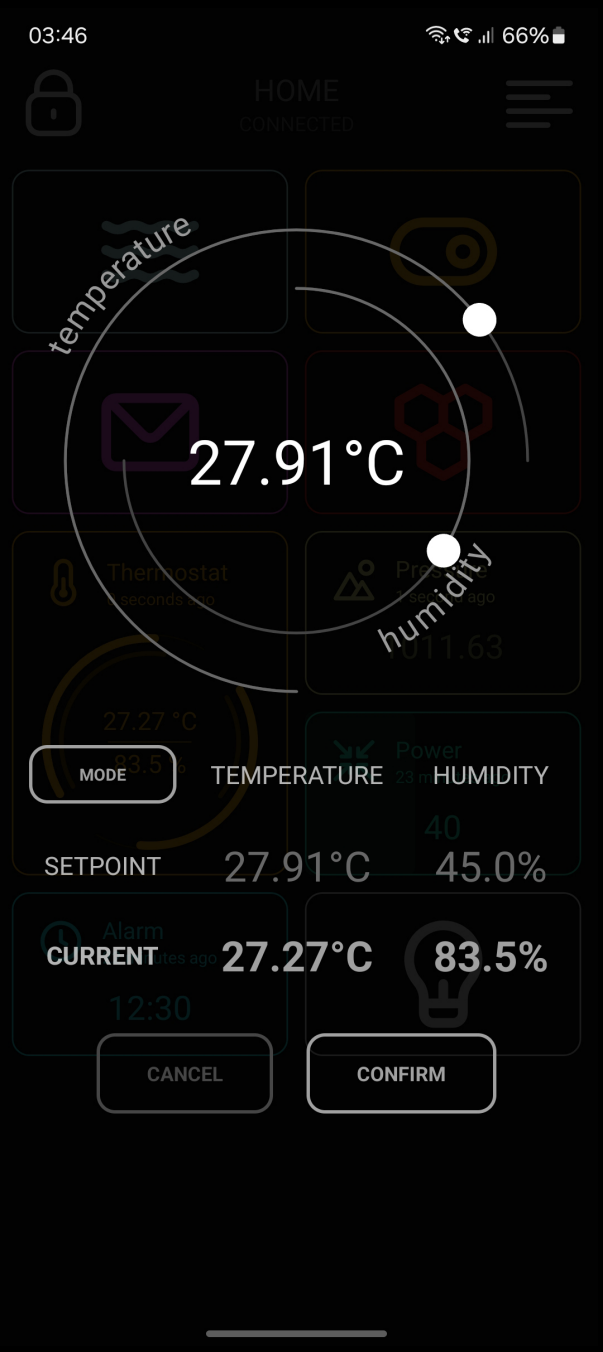
Rys. 10. Funkcja dziennika



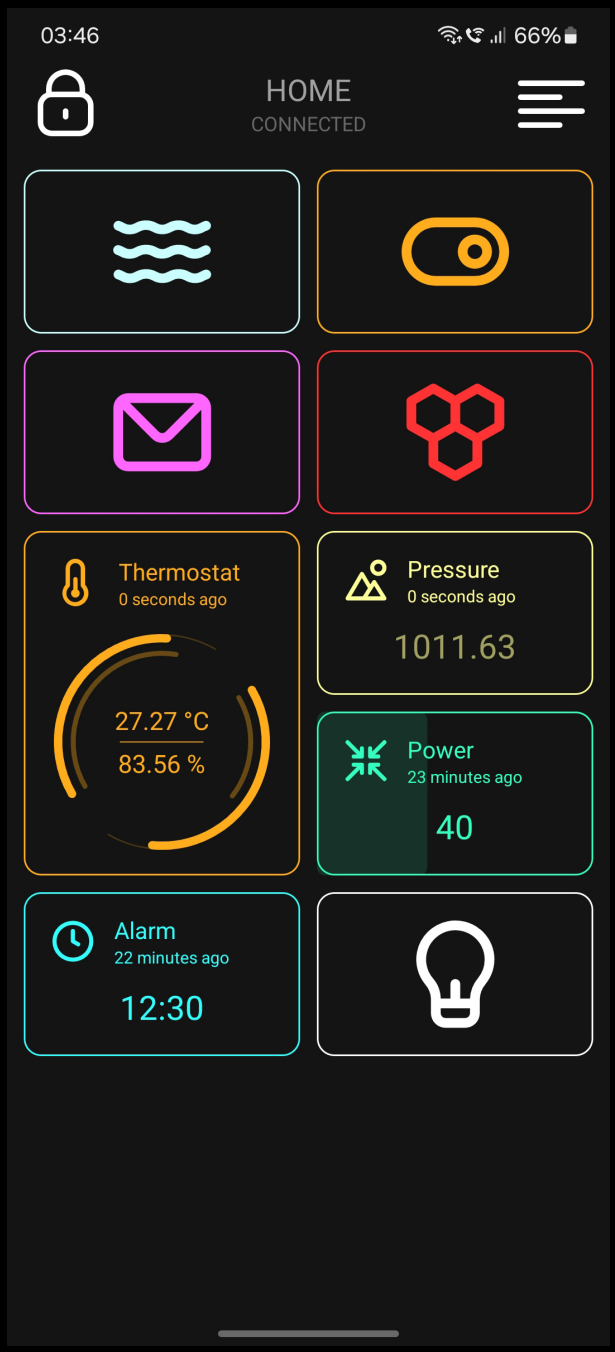
Rys. 11. Ustawienia



Rys. 12. Ekran główny



Rys. 13. Interfejs kafelki Thermostat



Rys. 14. Przykładowy dashboard

10. Podsumowanie

Prezentowany projekt powstawał na przestrzeni kilku lat. Głównym założeniem było nabycie doświadczenia oraz nauka. Projekt pozwolił na zdobycie umiejętności programowania aplikacji mobilnych w nowym języku, bliższe zrozumienie programowania obiektowego oraz naukę projektowania interfejsów użytkownika.

Efektem jest aplikacja która jest intuicyjna, elastyczna i może być dostosowana do różnych potrzeb. Dzięki modularnej budowie można ją łatwo rozwijać i integrować z nowymi technologiami.

W trakcie realizacji projektu napotkano różne wyzwania. Rozwiązanie tych problemów pozwoliło zdobyć cenne doświadczenie, które może być wykorzystane w przyszłości.

Podsumowując, praca nad aplikacją była okazją do stworzenia praktycznego narzędzia oraz do rozwoju wiedzy i umiejętności w wielu dziedzinach informatyki.

11. Bibliografia

Podane adresy URL zostały sprawdzone dnia 1 grudnia 2024.

[1] 2022 Survey Shows MQTT Adoption is High for Industry 4.0
https://www.hivemq.com/blog/2022-survey-shows-mqtt-adoption-is-high-for-industry

[2] Agile Manifesto
https://agilemanifesto.org

[3] Android Studio
https://developer.android.com/develop

[4] BBC - AWS: Amazon web outage breaks vacuums and doorbells
https://www.bbc.com/news/technology-55087054

[5] FasterXML Jackson Module Kotlin - dokumentacja
https://github.com/FasterXML/jackson-module-kotlin

[6] Foreground service types are required
https://developer.android.com/about/versions/14/changes/fgs-types-required

[7] Get started with Firebase Crashlytics
https://firebase.google.com/docs/crashlytics/get-started?platform=android

[8] Git - dokumentacja
https://git-scm.com/docs

[9] Google Play - developer content policy
https://play.google/developer-content-policy

[10] Gradle - dokumentacja
https://docs.gradle.org/current/userguide/userguide.html

[11] HiveMQ MQTT Client - dokumentacja
https://hivemq.github.io/hivemq-mqtt-client

[12] Introduction to MQTT 5 Protocol - MQTT 5 Essentials Part 1
https://www.hivemq.com/blog/mqtt5-essentials-part1-introduction-to-mqtt-5/

[13] Jetpack Compose - dokumentacja
https://developer.android.com/develop/ui/compose/documentation

[14] Kotlin - dokumentacja
https://kotlinlang.org/docs/home.html

[15] Latest Research Shows MQTT is Seeing Increased Adoption in IoT
https://www.hivemq.com/blog/latest-research-shows-mqtt-seeing-increased-adoption-iot-iiot

[16] Material Design 2
https://m2.material.io/

[17] Material Design 3
https://m3.material.io/

[18] Mosquitto - dokumentacja
https://mosquitto.org/documentation

[19] MQTT 3.1.1 Specification
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[20] MQTT 5 Specification
https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[21] MQTT 5 Vs. MQTT 3 – MQTT 5 Essentials Part 2
https://www.hivemq.com/blog/mqtt5-essentials-part2-foundational-changes-in-the-protocol/

[22] Number of IoT connected devices worldwide from 2019 to 2033
https://www.statista.com/statistics/1194682/iot-connected-devices-vertically

[23] Number of users of smart homes worldwide from 2019 to 2028
https://www.statista.com/forecasts/887613

[24] Sentry Android - dokumentacja
https://docs.sentry.io/platforms/android

[25] The GNU Manifesto - Why All Computer Users Will Benefit
https://www.gnu.org/gnu/manifesto.en.html%5C#benefit

[26] The Precision Agriculture Based on Wireless Sensor Network with MQTT Protocol
https://iopscience.iop.org/article/10.1088/1755-1315/207/1/012059

[27] Waterfall Methodology: A Complete Guide
https://business.adobe.com/blog/basics/waterfall

12. Załaczniki

1. Praca dyplomowa w formacie PDF
2. Kod źródłowy aplikacji