

Dokumentacja projektu „Reflekto”

Michał Kwiecień
Michał Wójcik

7 czerwca 2017

Streszczenie

Dokumentacja projektu inteligentnego lustra w konwencji IoT komunikującego się ze smartfonem z użyciem interfejsu Bluetooth Low Energy. Projekt powstał na potrzeby konkursu Nordic Semiconductor Student Contest.



Spis treści

1 Ogólny opis projektu	3
2 Prezentacja działania	3
3 Możliwości personalizacji	4
4 Opis aplikacji systemu wbudowanego nRF52	5
4.1 Włączenie urządzenia	5
4.2 Zasilanie	5
4.3 Sposób rozgłaszania	5
4.4 Zaimplementowane servisy	5
4.5 Obsługa odbieranych danych	6
4.6 Zabezpieczenia i charakterystyka konfiguracyjna	6
4.7 Umieszczanie informacji na wyświetlaczu	7
5 Opis aplikacji systemu iOS	7
5.1 Konfiguracja	8
5.2 Ekran główny	13
5.3 Działanie w tle	13
5.4 Połączenie oraz problem długiego czasu rozłączania w iOS	14
5.5 Programowanie funkcyjne	14
5.6 Pobierane dane i sposób ich pobrania	14
5.7 Testy jednostkowe	15
6 Sposób replikacji	15
6.1 iOS	15
6.2 Embedded	15
7 Podsumowanie	15

1 Ogólny opis projektu

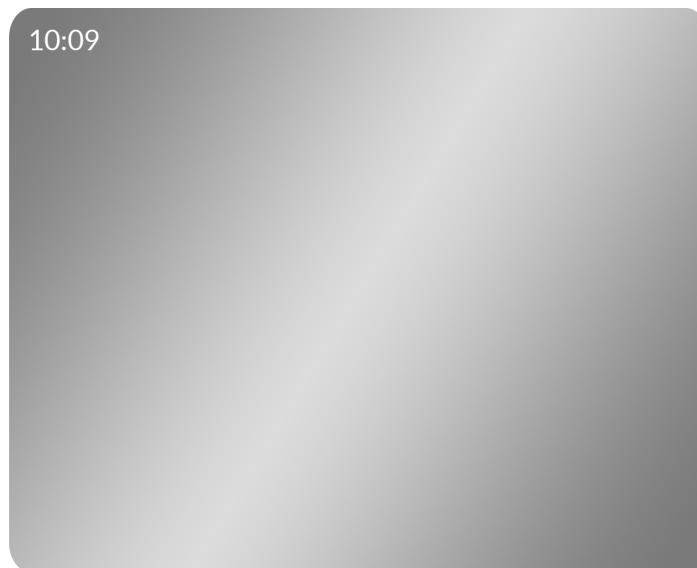
Założeniem projektu jest stworzenie inteligentnego lustra, które podczas wykonywania codziennej czynności umożliwia podgląd najświeższych i spersonalizowanych informacji. Informacje te zostaną wyświetcone na wyświetlaczu umieszczonym za lustrem weneckim, dzięki czemu będą widoczne jednocześnie obok odbicia.

Działanie lustra opiera się na przekazaniu danych poprzez moduł Bluetooth ze smartfona do modułu nRF52 i umieszczeniu ich na podłączonym ekranie. Aktywacja lustra nastąpi w momencie zbliżenia się do niego użytkownika. Z lustra może korzystać wielu użytkowników, gdyż każdorazowo przesyłane są indywidualne dane dla każdego z nich.

W celu wygenerowania danych stworzona została dedykowana aplikacja dla systemu iOS. Po wstępnej konfiguracji umożliwia ona zautomatyzowanie procesu i przesyłanie wiadomości w tle bez późniejszych ingerencji użytkownika.

2 Prezentacja działania

Kiedy lustro nie jest w bliskim zasięgu jednego ze sparowanych telefonów, wyświetlana jest godzina lub pozostaje ono wyłączone w zależności od ustawień (Rys. 1)



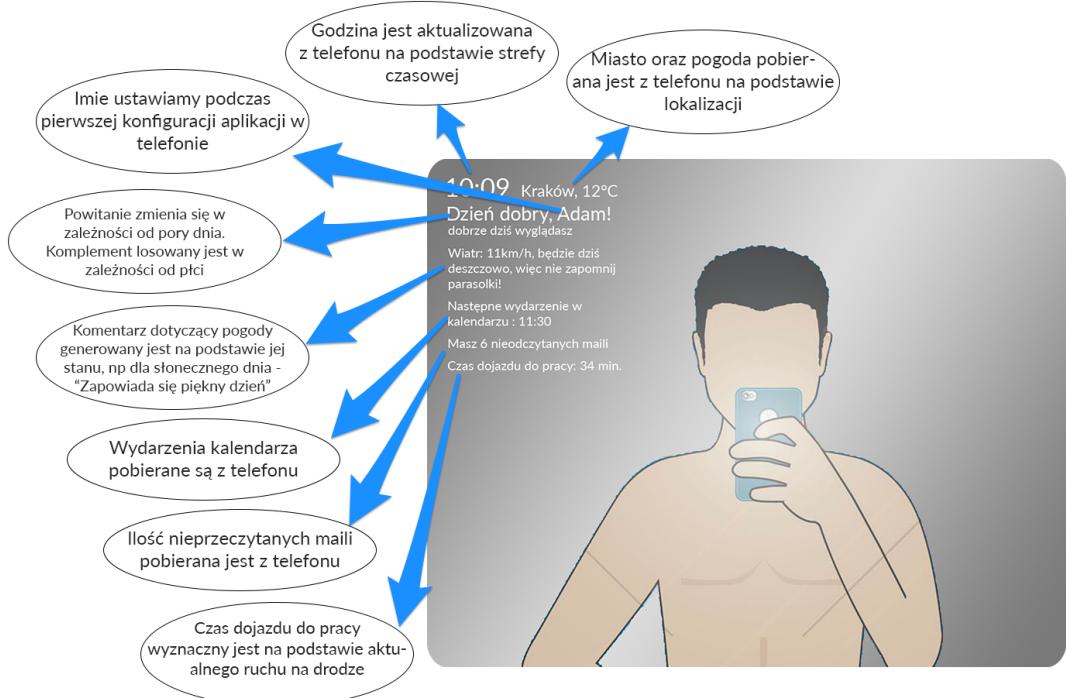
Rysunek 1: Lustro w stanie wyłączone

W momencie zbliżenia się użytkownika następuje transmisja danych i wyświetlenie aktualnych informacji (Rys. 2).



Rysunek 2: Poglądowy rysunek lustra w stanie aktywnym

3 Możliwości personalizacji



Rysunek 3: Poglądowy rysunek możliwości personalizacji

4 Opis aplikacji systemu wbudowanego nRF52

4.1 Włączenie urządzenia

Urządzenie po uruchomieniu inicjalizuje wszystkie niezbędne usługi: timery, stos BLE, serwisy oraz obsługę wyświetlacza. Dalej rozpoczyna się rozgłaszczenie (pod nazwą „nRF_Reflektō”) i urządzenie pozostaje w stanie oczekiwania aż do wyłączenia zasilania.

4.2 Zasilanie

Ze względu na użycie wyświetlacza TFT, wymagane jest zasilanie przez zasilacz zewnętrzny bądź port USB (bateria pastylkowa nie pokrywa zapotrzebowania na prąd). Szacowany pobór energii w stanie aktywnym wynosi 400mW. W stanie nieaktywnym pobór energii przez wyświetlacz może zostać zredukowany do zera (sterowanie PWM), jednakże wymaga to fizycznej ingerencji w płytę wyświetlacza. W związku z tym zdecydowano o pozostawieniu przez cały czas aktywnego zegara, jako zawsze aktualnej informacji.

4.3 Sposób rozgłaszania

Do urządzenia w tym samym czasie może być podłączony jeden smartfon, ale z punktu widzenia użytkownika z lustra może korzystać kilka osób jednocześnie. Zostało to osiągnięte poprzez wprowadzenie 4-sekundowych slotów czasowych w przesyłaniu danych przez każdy smartfon. Wstępnie moc rozgłaszania została obniżona o 30dB (potrzebna jest późniejsza kalibracja w docelowym urządzeniu). Dzięki temu oszczędzana jest bateria w smartfonie jak i lustrze – dane przesyłane są jak tylko lustro zostanie wykryte przez smartfona i nie jest wymagane ciągła aktywność w tle z urządzeniem.

Układ rozgłasza się pod wcześniej zdefiniowaną nazwą z dwoma servisami. Pierwszy posiada UUID „this is reflektō” zapisane w systemie szesnastkowym (w celu unikalnej identyfikacji):

1 74686973 - 2069 - 7320 - 7265 - 666C656b746F

Drugi to ustandaryzowana usługa czasu o UUID 0x1805 zaimplementowana w urządzeniu.

4.4 Zaimplementowane servisy

W układzie zostały zaimplementowane cztery servisy *Bluetooth Low Energy* których struktura i adresy są następujące:

- 0x1805 – Usługa aktualnego czasu
 - 0x2A2B – read/write – czas w systemie unixowym
- 0x0010 – Usługa pogodowa
 - 0x0011 – aktualne miasto, temperatura i ikona pogody
 - 0x0012 – aktualna prędkość wiatru
 - 0x0013 – porada pogodowa
- 0x0020 – Usługa osobistych informacji
 - 0x0021 – następne wydarzenie z kalendarza
 - 0x0022 – informacja o nieodczytanych mailach

- 0x0023 – prognozowany czas dojazdu do pracy
 - 0x0024 – imię użytkownika
 - 0x0025 – powitanie użytkownika
 - 0x0026 – komplement
- 0xDEAD – Usługa konfiguracyjna
 - 0xBEEF – zapis informacji konfiguracyjnych

4.5 Obsługa odbieranych danych

Każda z wyświetlanych informacji przechowywana jest jako struktura zawierająca:

- łańcuch znaków o ustalonej wcześniej długości (100 znaków)
- licznik ilości zapisanych znaków
- flagę mówiącą czy odbiór danych dla tego typu informacji został zakończony (czy jest możliwość poprawnego wyświetlenia całej informacji)
- flagę informującą czy informacja zmieniła się od ostatnio otrzymanej (czy jest konieczność jej ponownego wyświetlenia)

Z racji narzuconego limitu wielkości jednego pakietu *BLE* do 20 bajtów, konieczne jest sklejanie poszczególnych paczek do jednego ciągu znaków. W tym celu skorzystano ze znaków *ASCII* oznaczonych jako *STX* – *Start of text* i *ETX* – *End of text*, które przedstawiają się formacie dziesiętnym jako cyfry 2 i 3.

W momencie otrzymania zapisu na jedną z charakterystyk obsługujących dane tekstowe, event przekazywany jest do funkcji łączącej dane. W przypadku otrzymania pakietu zaczynającego się od *STX*, flagi oraz licznik znaków są zerowane i następuje zapis do łańcucha znaków. Porównywany jest otrzymany łańcuch z tym zapisanym poprzednio, dzięki temu wiadomo czy konieczne jest ponowne wyświetlenie danych. Odbiór kończy się w przypadku otrzymania znaku *ETX* (na dowolnym miejscu). Ustawiana jest wtedy flaga zakończenia.

Przykładowo porada pogodowa „It will rain till tomorrow morning” zostanie podzielona w następujący sposób:

1. *STX*It will rain till t
2. omorrow morning*ETX*

Wielodniowe testy nie wykazały żadnych błędów przy stosowaniu powyższej metody transmisji i obsługi danych.

4.6 Zabezpieczenia i charakterystyka konfiguracyjna

Przy podłączeniu dowolnego urządzenia uruchamiany jest timer, który zezwala na pozostanie połączonym maksymalnie przez dwie sekundy. W praktyce pozwala to jedynie na odkrycie struktury urządzenia. W przypadku nie wpisania ustalonego hasła (*001220*) na charakterystykę konfiguracyjną, lub wpisaniu jakichkolwiek danych do systemu lustra przed podaniem go, połączenie zostaje natychmiast zerwane. Wpisanie hasła zatrzymuje timer i powoduje oczekiwanie na dane (np. w przypadku dłuższego niż zwykle pobierania danych z API).

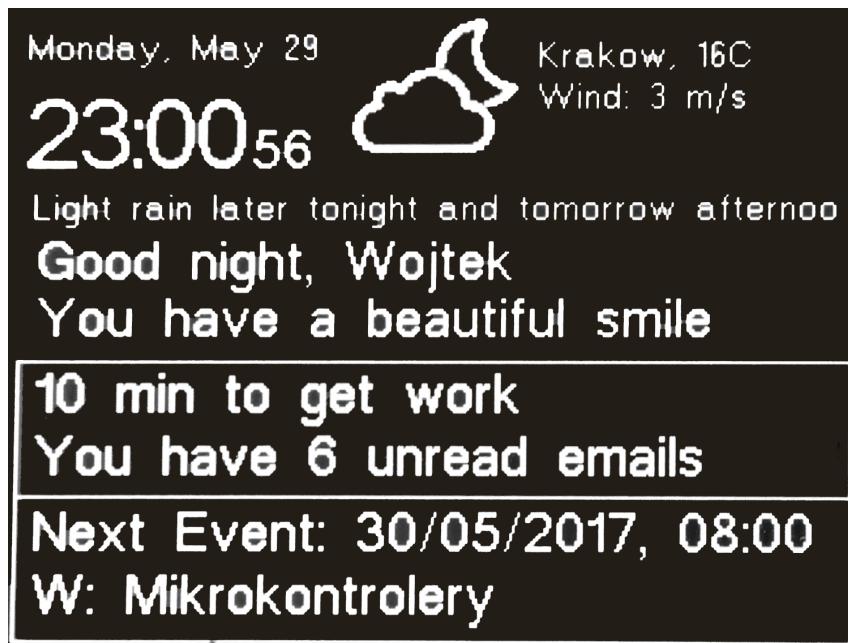
Ze względu na optymalizację poboru energii smartfona, wykorzystywane jest specjalne hasło pozwalające na natychmiastowe rozłączenie z urządzeniem z ominięciem ograniczeń systemu iOS.

4.7 Umieszczanie informacji na wyświetlaczu

Ze względu na rodzaj wyświetlacza oraz użytką komunikację do przesyłu danych po *SPI*, dostęp do wyświetlacza odbywa się pixel po pixelu. By zachować możliwość wybiórczego umieszczania i czyszczenia informacji, wyświetlacz został podzielony na sekcje:

- Sekcja Czasu – odświeżana selektywnie (sekundy co sekundę, minuty i godziny co 60 sekund, data w momencie zmiany dnia tygodnia)
- Sekcja Pogody – odświeżana pojedynczo w przypadku zmiany otrzymanych danych, obejmuje zbiór 10 ikon pogodowych drukowanych pixel po pixelu, aktualne miasto, temperaturę, wiatr i poradę pogodową
- Sekcja Powitania i Komplementu – odświeżana co 4 sekundy w przypadku obecności kilku użytkowników jednocześnie
- Sekcje e-mail, czasu dojazdu i kalendarza – odświeżane w przypadku zmiany

Wyświetlacz jest czyszczony w całości 30 sekund po rozłączeniu się ostatniego użytkownika. Odświeżanie zegara pozostaje wtedy bez zmian. Wygląd wyświetlacza przedstawia rysunek 4.



Rysunek 4: Interfejs lustra

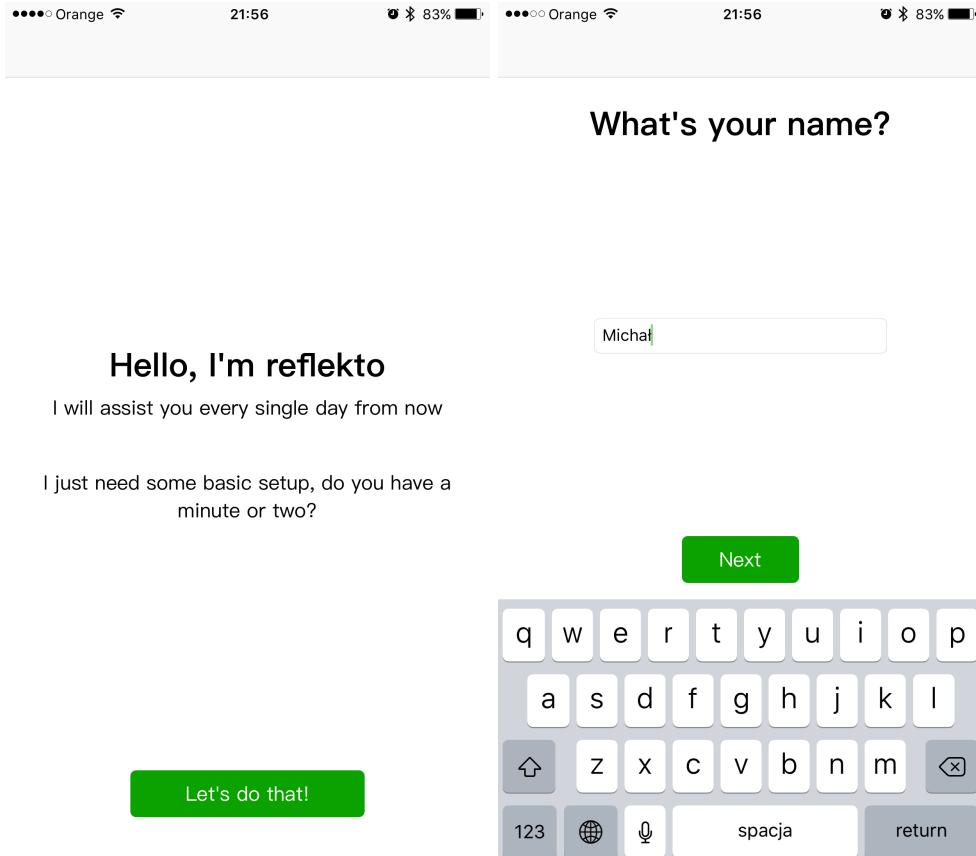
5 Opis aplikacji systemu iOS

Aplikacja mobilna po uruchomieniu sprawdza, czy został już przeprowadzny proces konfiguracji. Jeśli nie, nastąpi jej automatyczne włączenie. Użytkownik prowadzony jest kolejno przez kilka

ekranów, na których musi udzielić odpowiedzi na kilka pytań.

5.1 Konfiguracja

1. Konfiguracja rozpoczyna się od powitania użytkownika oraz poinformowania go o wymaganym procesie konfiguracji. Następnie użytkownik proszony jest o podanie swojego imienia. Będzie ono użyte później, do wyświetlenia odpowiedniego powitania na lustrze. Obok imienia pojawi się również tekst zależny od pory dnia. (Rys. 5)



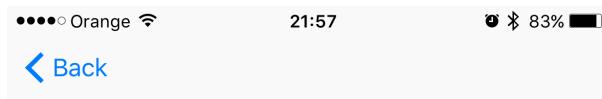
Rysunek 5: Wstęp do konfiguracji oraz pobranie imienia

2. Kolejnym krokiem jest wybór płci. Dzięki jej znajomości generowany jest jeden z wielu komplementów na temat użytkownika lustra. (Rys. 6)



Rysunek 6: Pobranie płci

3. Czwarty ekran wyświetla trzy przyciski, dzięki którym użytkownik zezwala aplikacja na dostęp do określonych danych w swoim telefonie. Jest to niezbędny krok z powodu bezpieczeństwa użytych w systemie iOS. Po przekazaniu wszystkich zgód ekran automatycznie przechodzi do kolejnego, nie ma potrzeby zatwierdzania. (Rys. 7)



You have to help me!

To do some magic I need your permission to
access your location

I love magic!

I can make even more magic, just give me
permission to access your calendar!

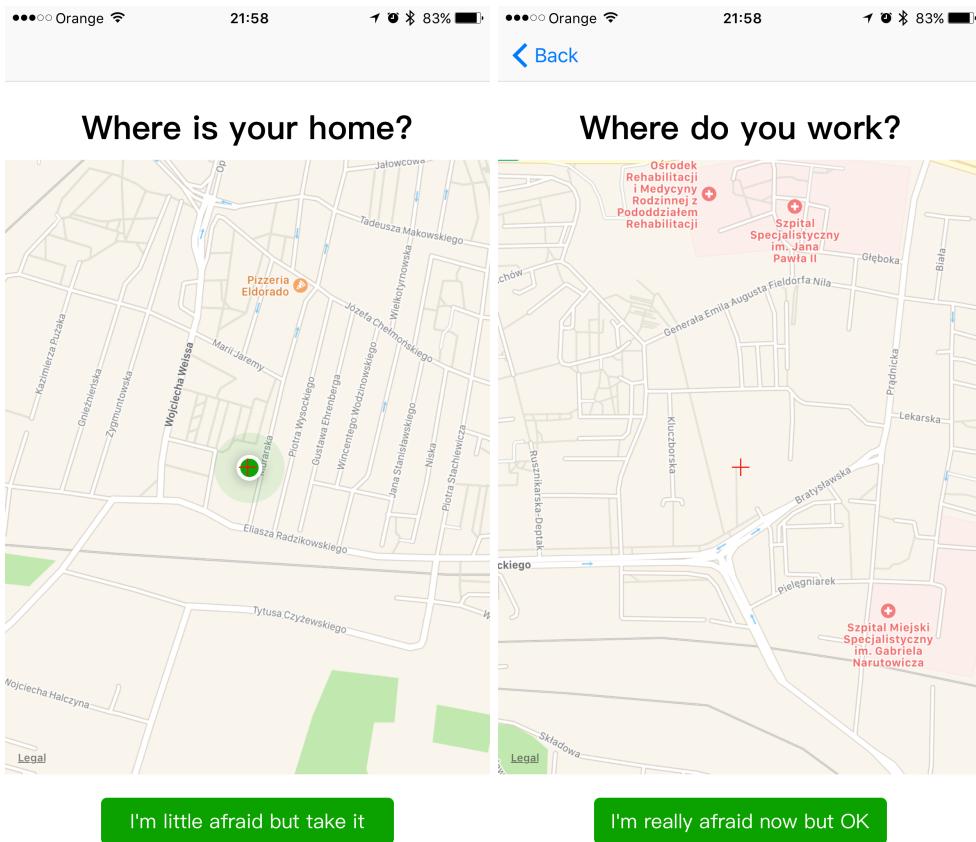
Take what you need!

Want even more magic? Give me access to
your Gmail

G Sign in

Rysunek 7: Prośba o zezwolenia

4. Ostatnim krokiem jest wskazanie aplikacji miejsca zamieszkania oraz pracy. Użyta została do tego mapa, która ułatwia ten proces. Automatycznie po wejściu w ekran pobierana jest lokalizacja użytkownika i wskazywane jest aktualne miejsce, w którym się znajduje. (Rys. 8)



Rysunek 8: Wskazanie na mapie miejsce zamieszkania oraz miejsca pracy

5. Po wskazaniu miejsca zamieszkania i pracy aplikacja pokazuje ekran, w którym informuje o zakończeniu procesu konfiguracji. Naciśnięcie przycisku powoduje przejście do ekranu głównego. (Rys. 9)



Rysunek 9: Potwierdzenia zakończenia procesu konfiguracji

5.2 Ekran główny

Po zakończeniu procesu konfiguracji lub każdorazowym późniejszym wejściu w aplikację użytkownikowi ukazuje się ekran główny (Rys. 10). Znajdują się na nim przyciski do włączenia skanera Bluetooth, wyłączenia go oraz do przeprowadzenia procesu konfiguracji ponownie. Przyciski do włączania i wyłączania skanera są niezbędne, aby aplikacja przeszła review podczas umieszczania jej w sklepie AppStore. Po włączeniu skanera, można już wyjść z aplikacji lub zablokować ekran – aplikacja będzie cały czas działała w tle.



Rysunek 10: Ekran główny

5.3 Działanie w tle

Z powodu bardzo restrykcyjnego podejścia systemu iOS do aplikacji działających w tle została ona maksymalnie zoptymalizowana, aby zużywała jak najmniej zasobów. Nadajnik Bluetooth w lustrze rozgłasza swoje pakiety z odpowienio niską mocą, aby połączenie odbywało się tylko

wtedy, gdy lustro faktycznie jest w pobliżu telefonu. Dodatkowo Informacje pobierane z API są cachowane w celu optymalizacji zużycia energii. Po wykryciu nadajnika, aplikacja ma 10 sekund na połączenie się z lustrem oraz pobranie wszystkich danych. Aplikacja przed rozpoczęciem kolejnego skanowania oczekuje 4 sekundy po rozłączeniu, dzięki czemu powstaje okno czasowe, podczas którego kolejne urządzenie w zasięgu ma szansę na połączenie i wysłanie swoich informacji.

5.4 Połączenie oraz problem długiego czasu rozłączania w iOS

Po połączeniu z lustrem, aplikacja ma 2 sekundy na przesłanie odpowiedniego ciągu bajtów do charakterystyki konfiguracyjnej. Jeśli tego nie zrobi zostanie automatycznie rozłączona, a żadne dane przez nią przesłane nie zostaną wyświetcone na lustrze. Dodając do tego obniżoną moc nadajnika całkowicie zabezpiecza to możliwość nieautoryzowanego użycia lustra przez osoby trzecie. Znanym problem, który występuje w systemie iOS jest jego czas rozłączania się z urządzeniem. Po wydaniu komendy rozłączenia nie jest to robione natychmiastowo, lecz dopiero po około 7 sekundach. Rozwiązaliśmy ten problem wpisaniem do charakterystyki konfiguracyjnej odpowiedniego ciągu bajtów, który zaraz po ich otrzymaniu rozłącza urządzenie. Rozłącznie po stronie sprzętu realizowane jest natychmiast, stąd nie występuje problem długiego rozłączania.

5.5 Programowanie funkcyjne

Aplikacja pobiera dane asynchronicznie, jest to wykonane w kodzie w bardzo schludny sposób dzięki zastosowaniu programowania funkcyjnego. Wykonane zostało to z użyciem biblioteki RxSwift. Poniżej fragment kodu ilustrujący, jak łatwo połączyć kilka różnych informacji pobieranych asynchronicznie w jeden element, który można przesyłać do urządzenia. Reactive Functional Programming to w ostatnich czasach bardzo popularne i pożadne podejście podczas programowania aplikacji mobilnych.

```
1 Observable.zip(  
2     DataManager.timestamp,  
3     DataManager.weather,  
4     DataManager.nextEvent,  
5     DataManager.name,  
6     DataManager.greeting,  
7     DataManager.compliment,  
8     DataManager.unreadMailsCount,  
9     DataManager.travelToWorkTime)  
10    .subscribe(onNext: { [weak self]  
11        (timestamp, weather, nextEvent, name, greeting,  
12            compliment, unreadMailsCount, travelWorkTime) in  
13            //wysłanie danych do lustra
```

5.6 Pobierane dane i sposób ich pobrania

- Godzina
- Miejsce, pogoda – na podstawie wyznaczonej lokalizacji telefon wysyła zapytanie do Dark Sky API podając w parametrze m. in. długość oraz szerokość geograficzną. Otrzymany w odpowiedzi JSON parsowany jest na obiekt, z którego później wybierana jest temperatura. Miejsce wyznaczone jest również na podstawie szerokości i długości geograficznej używając wbudowanego API Apple Maps.

- Wiatr – pobierana z Dark Sky API
- Pogoda - dodatkowa informacja. – pobierana z Dark Sky API
- Powitanie
- Imię
- Komplement
- Czas dojazdu do pracy – wyznaczany na podstawie prawdziwego ruchu na ulicach w aktualnym momencie
- Ilość nieodczytanych mail
- Następne wydarzenie w kalendarzu

5.7 Testy jednostkowe

Jednym z trudniejszych algorytmów zastosowanych w aplikacji jest algorytm podziału danych na paczki. Z tego powodu zostały napisane do niego testy jednostkowe a sam algorytm powstał dzięki TDD (Test Driven Development)

6 Sposób replikacji

6.1 iOS

Po sklonowaniu repozytorium z brancha master, należy w konsoli przejść do folderu z projektem oraz uruchomić komendę

```
1 pod install
```

Jeśli komenda nie jest dostępna należy zainstalować oprogramowanie CocoaPods

6.2 Embedded

Najprostszym sposobem umieszczenia oprogramowania na DevKicie nRF52 jest skopiowanie pliku *nRF_Reflektō.hex* do urządzenia J-LINK wyświetlanego przez komputer. Folder z wygenerowanymi plikami został specjalnie pozostawiony w repozytorium:

```
1 /reflektō-embedded/nRF_Reflektō_Services/Output/nrf52832_xxaa/Exe/
```

Wymagane jest wcześniejsze posiadanie zainstalowanego SoftDevice na płytce. Można tego dokonać poprzez aplikację nRFgo Studio. Aplikacja była rozwijana i testowana z użyciem SoftDevice

```
1 s132_nrf52_4.0.2_softdevice.hex
```

Inną możliwością jest własnoręczne skompilowanie plików źródłowych. Projekt rozwijany był z użyciem IDE Segger Embedded Studio 3.12. Po pobraniu repozytorium wystarczy uruchomić plik projektu *nRF_Reflektō.emProject* i wybrać z menu *Build/Build and run*.

7 Podsumowanie

Projekt rozwijany był z użyciem systemu kontroli wersji GitHub. Wszystkie postępy prac dostępne są pod adresem: <https://github.com/Solstico>