

Dokumentacja projektu „Reflekto”

Michał Kwiecień
Michał Wójcik

29 maja 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 Realizacja komunikacji	5
4.5 Umieszczanie informacji na wyświetlaczu	6
4.6 Planowany rozwój	6
5 Opis aplikacji systemu iOS	6
5.1 Konfiguracja	6
5.2 Ekran główny	11
5.3 Działanie w tle	12
5.4 Połączenie oraz problem długiego czasu rozłączania w iOS	12
5.5 Programowanie funkcyjne	13
5.6 Pobierane dane i sposób ich pobrania	13
5.7 Testy jednostkowe	13
6 Sposób replikacji	14
6.1 iOS	14
6.2 Embedded	14
7 Podsumowanie	14

1 Ogólny opis projektu

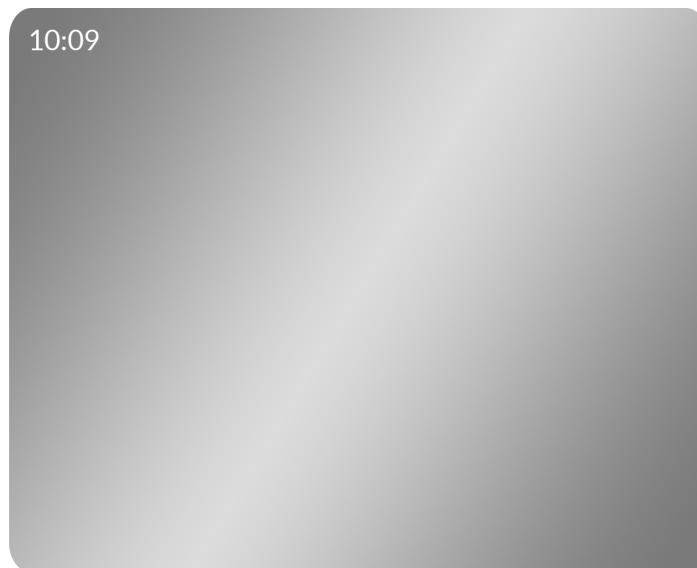
Założeniem projektu jest stworzenie inteligentnego lustra, które podczas wykonywania codziennej czynności umożliwia podgląd najświeższych i spersonalizowanych informacji. Informacje te zostaną wyświetcone na wyświetlaczu umieszczonym za lustrem weneckim, dzięki czemu będą widoczne jednocześnie obok odbicia.

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 kaźdorazowo 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)



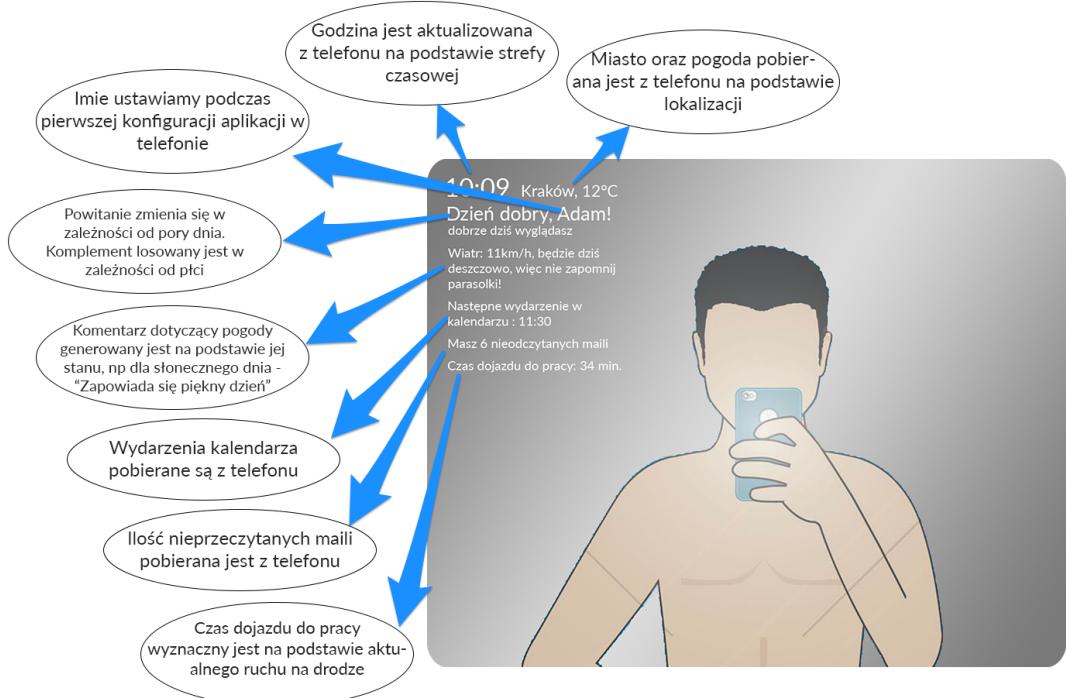
Rysunek 1: Lustro w stanie wyłączone

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

ftowy

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 wlutowania odpowiedniej zworki w płytę wyświetlacza. W związku z tym zdecydowano o pozostawieniu przez cały czas aktywnego zegara, jako jedynej 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 40dB (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 z urządzeniem.

Układ rozgłasza się pod wcześniej zdefiniowaną nazwą i UUID: "this is reflektō", co w systemie szesnastkowym przedstawia się następująco:

1 74686973-2069-73 20 72 65 66 6c 65 6b 74 6f

4.4 Realizacja komunikacji

Układ w momencie otrzymania danych wysyła je do komputera przez port szeregowy (w celu debugowania), oraz rzutuje je na wektor typu *char*, który przekazywany jest do kolejnej funkcji realizującej umieszczenie danych na wyświetlaczu.

Ze względu na ograniczenie ilości danych w pakiecie do 20 bajtów zastosowano specjalne kodowanie, gdzie pierwsze dwa bajty analizowane są w następujący sposób:

- bajt „0”

Wskazuje na typ otrzymanych danych (np. data, godzina, pogoda, kalendarz, e-mali)

- bajt „1”

4 bity górne – numer aktualnie otrzymanego pakietu

4 bity dolne – ilość wszystkich pakietów danego typu

4 bity
można
użyć do
„szyfró-
wania”
transmi-
sji

Wynika z tego maksymalna długość przesyłanych danych jednego typu do 288 znaków (16 pakietów po 18 użytecznych bajtów). Podział na paczki realizowany jest przez aplikację na smartfonie. Założono także obsługę błędnych pakietów przez SoftDevice'a, dlatego nie uwzględnione zostały bajty na sumę CRC.

4.5 Umieszczanie informacji na wyświetlaczu

Po otrzymaniu wszystkich paczek danego typu, dane „drukowane” są na wyświetlaczu (ze względu na metodę komunikacji niemożliwe jest przesłanie jednorazowo całej ramki obrazu). Wyświetlacz podzielony został na sekcje (od góry):

- 20% – aktualna data i godzina
- 20% – aktualna pogoda
- 60% – kalendarz, czas dojazdu do pracy, emaile itd. wyświetlane cyklicznie bądź jednocześnie

4.6 Planowany rozwój

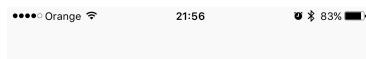
W planach jest obsługa danych konfiguracyjnych, które ustalałyby m.in. sposób wyświetlania informacji lub działania układu w trybie nieaktywnym. Dodatkowo w przypadku pomyślnego ukończenia projektu przed czasem, możliwe jest ponowne jego napisanie bez użycia protokołu UART, co zmniejszyłoby narzut na mikroprocesor.

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.

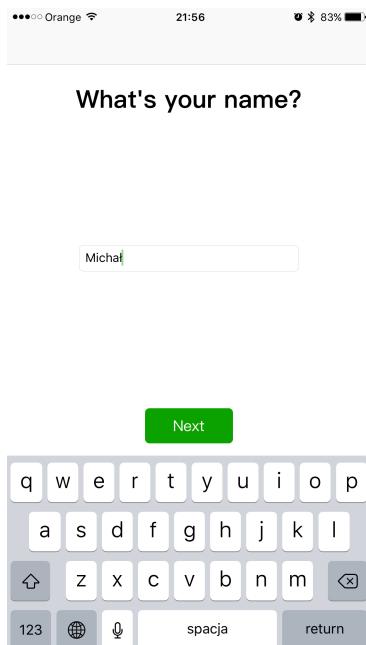
5.1 Konfiguracja

1. Konfiguracja rozpoczyna się od powitania użytkownika oraz poinformowania go o wymaganym procesie konfiguracji. (Rys. 4)



Rysunek 4: Wstęp do konfiguracji

2. 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)



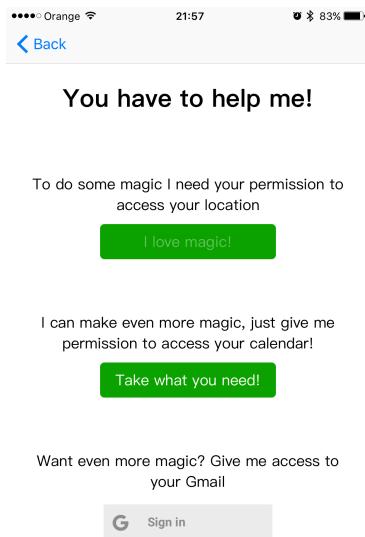
Rysunek 5: Pobranie imienia

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



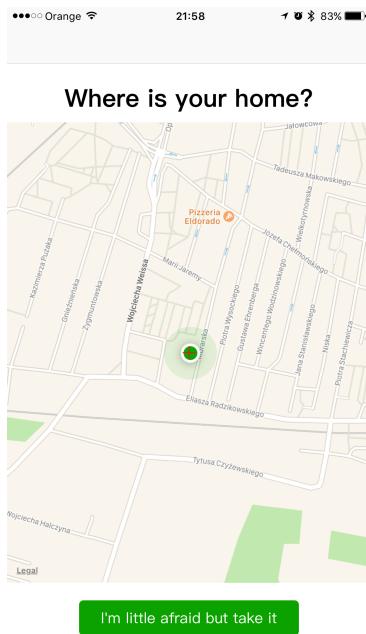
Rysunek 6: Pobranie płci

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

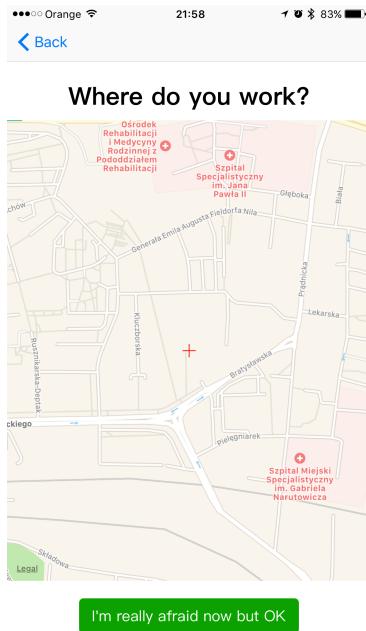


Rysunek 7: Prośba o zezwolenia

5. 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 znajduję. (Rys. 8) (Rys. 9)



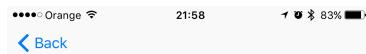
Rysunek 8: Wskazanie na mapie miejsce zamieszkania



Rysunek 9: Wskazanie na mapie miejsce pracy

6. Po wskazaniu miejsca zamieszkania i pracy aplikacja pokazuje ekran, w którym informuje o zakończeniu procesu konfiguracji. Naciśnięcie przycisku powoduje przejście do ekranu

głównego. (Rys. 10)



Rysunek 10: Potwierdzenia zakończenia procesu konfiguracji

5.2 Ekran główny

Po zakończeniu procesu konfiguracji lub każdorazowym późniejszym wejściu w aplikację użytkownikowi ukazuję się ekran główny (Rys. 11). 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 AppStore. Po włączeniu skanera, można już wyjść z aplikacji lub zablokować ekran - aplikacja będzie cały czas działała w tle.



Rysunek 11: 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 wyświetlane na lustrze. Dodając do tego bardzo słabą moc nadajnika całkowicie zabezpiecza to możliwość nieautoryzowanego użycia lustra. 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 schudny 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 nieodczytyanych 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

Tutaj to pan kopujesz hexa i działa, heja, dawać iPada.

7 Podsumowanie

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