

A G H

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

Katedra Automatyki i Inżynierii Biomedycznej

Praca dyplomowa magisterska

Projekt i wykonanie interfejsu dla inteligentnego domu z wykorzystaniem technologii Qt

Design and implementation of interface for smart home using Qt technology

Autor: Adam Styrc
Kierunek studiów: Automatyka i Robotyka
Opiekun pracy: Dr inż. Paweł Rotter

Kraków, 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

1 Table of Contents

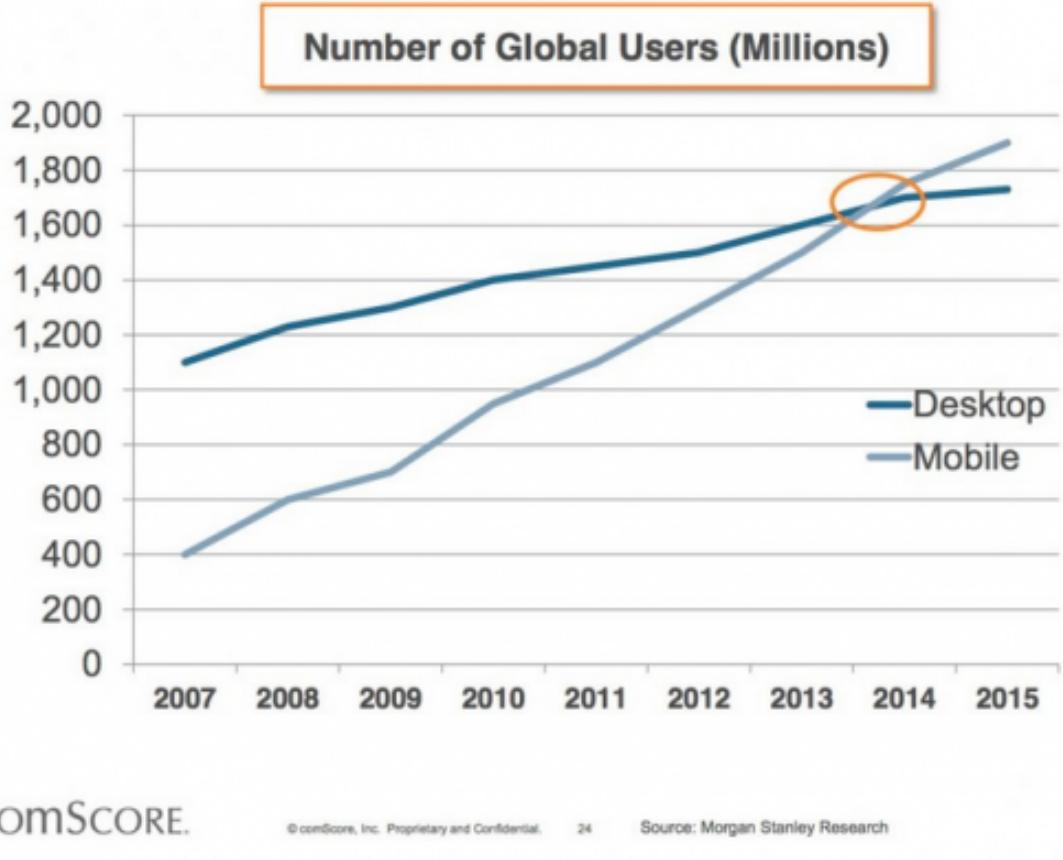
2 Wstęp	4
2.1 Obecne rozwiązania	6
2.2 Cross-platform tools	7
2.2.1 Apache Cordova (Phonegap)	7
2.2.2 Appcelerator (Titanium)	8
2.2.3 Adobe AIR	8
2.2.4 Xamarin.....	9
2.2.5 Unity	9
2.3 Qt 5 – nowe narzędzie do cross-kompilacji.....	10
3 Cel pracy	11
3.1 Przepływ (nawigacja)	12
3.2 Serwer centralny.....	13
3.3 Dashboard	13
3.4 Pokoje (Rooms).....	14
3.5 Urządzenia (Devices)	16
3.6 Formularz dodawania/edykcji urządzeń.....	17
3.7 Kamera	18
3.8 Ustawienia (Settings)	19
4 Implementacja.....	19
4.1 Podstawy	19
4.2 Przepływ (nawigacja)	20
4.3 Inicjalizacja	23
4.4 Dashboard	24
4.5 Ustawienia (Settings)	27
4.6 Urządzenia (Devices).....	31
4.7 Pokoje (Rooms).....	34
4.7.1 Strategia 1- i 2-kolumnowa.....	38
4.8 Formularz dodawania/edykcji urządzenia	39
4.9 Kamera	43
4.10 Odświeżanie	45
4.10.1 PullToRefresh	45
4.11 WebService.....	46
4.11.1 Obsługa błędów.....	47
4.12 Serwer centralny.....	48
4.13 Układ wykonawczy	49
5 Podsumowanie	53
5.1 Wnioski.....	53
5.2 Wkład autora	55
6 Bibliografia	55

2 Wstęp

We współczesnym świecie mamy do czynienia z powszechną komputeryzacją społeczeństwa. Jednak od konstrukcji pierwszych komputerów w połowie XX wieku technologia bardzo szybko i prężnie się rozwinęła, a swój dalszy rozwitk przeżywa do dziś. W dzisiejszych czasach większość ludzi ma dostęp nie tylko do komputerów stacjonarnych ale i urządzeń mobilnych, czyli tzw. smartfonów czy tabletów. Dlatego tworząc oprogramowanie, które ma być wygodne i ma na celu dotrzeć do jak największej liczby odbiorców, trzeba dane rozwiążanie informatyczne obsłużyć na wielu systemach operacyjnych. Z reguły istotne są te najpopularniejsze takie jak desktopowe Windows, OS X i Linux oraz mobilne Android, iOS. Czasem nawet może zajść potrzeba implementacji na systemy wbudowane RTOS dla dedykowanych urządzeń.

Stawia to wszystkich twórców aplikacji przed dużym wyzwaniem, ponieważ dane aplikacje muszą działać pod wieloma systemami operacyjnymi. Współcześnie, najczęściej zmusza to firmy do zatrudnienia osobnych sztabów informatycznych odpowiedzialnych za daną platformę tudzież do rezygnacji z grupy odbiorców użytkujących konkretny system, w przypadku decyzji aby go nie wspierać. Pierwsze rozwiązanie generuje dodatkowe - często spore - koszty, a drugie powoduje spadek dochodu.

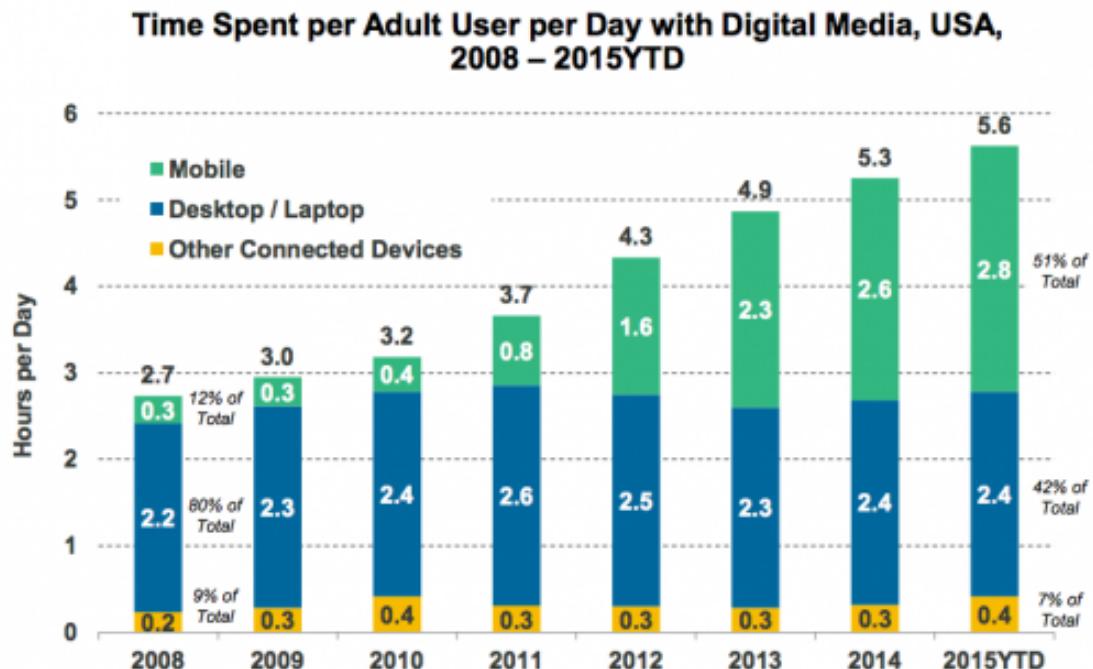
Warto jednak przyjrzeć się statystykom, gdyż udział w dostępie do danych przez urządzenia różnego typu dosyć znaczco się zmienił na przestrzeni ostatnich lat:



Rysunek 1 Liczba użytkowników komputerów stacjonarnych i urządzeń mobilnych

Otoż w ostatnich latach mamy do czynienia z gwałtownym wzrostem użytkowników urządzeń mobilnych. Na wykresie widać, że ich liczba przekroczyła liczbę użytkowników komputerów stacjonarnych już w 2014 roku! Nie oznacza to *stricte* wzrostu osób podłączonych do internetu – co też zapewne ma miejsce - ale świadczy o tym, że ludzie w ciągu doby są skłonni korzystać z obydwu rodzajów urządzeń. Bardziej obrazuje to z kolei poniższy wykres:

Internet Usage (Engagement) Growth Solid +11% Y/Y = Mobile @ 3 Hours / Day per User vs. <1 Five Years Ago, USA



@KPCB

Source: eMarketer 9/14 (2008-2010), eMarketer 4/15 (2011-2015). Note: Other connected devices include OTT and game consoles. Mobile includes smartphone and tablet. Usage includes both home and work. Ages 18+; time spent with each medium includes all time spent with that medium, regardless of multitasking.

14

Rysunek 2 Czas spędzony przez użytkownika na konkretnych urządzeniach

Istnieje na to proste wy tłumaczenie - ludzie częściej korzystają z komputerów stacjonarnych w pracy bądź w domu wieczorem – bo jest to wygodniejsze, szybsze i pozwala na bardziej zaawansowane działania. Jednak kiedy są poza biurem albo domem, to naturalnie wolą używać urządzeń mobilnych.

2.1 Obecne rozwiązania

Jednym z możliwych rozwiązań stanowi ograniczenie się tylko do aplikacji webowej, czyli strony internetowej. Powinna ona zostać wyświetlona zasadniczo przez każde urządzenie posiadające dostęp do internetu i wyposażone w przeglądarkę. Jednak to rozwiązanie nie jest idealne z kilku przyczyn.

Po pierwsze, w dalszym ciągu wyzwaniem jest obsłużenie responsywności interfejsu użytkownika. Strona internetowa musi być wyskalowana na wszelkiego formatu monitorach, tabletach i telefonach. Zwykle generuje to zmiany w układzie strony, a często nawet w jej

nawigacji, czyli sposobie poruszania się po witrynie. Nad intuicyjnym i przyjaznym użytkownikowi interfejsem graficznym aplikacji czuwają dodatkowe osoby - specjaliści od UX, czyli User Experience.

Ponadto, to podejście zakłada w sobie dostępność internetu, a nie zawsze lub nie w każdych warunkach jest to możliwe. Bez internetu pobranie danych potrzebnych chociażby do zbudowania UI nie może się zaistnieć. Co więcej, dochodzi opóźnienie działania aplikacji online wynikające z czasu odpowiedzi serwerów, a to jest często nie do przyjęcia.

Trzecim kontrargumentem jest zasadniczy brak wsparcia dla natywnych bibliotek systemu. Nie możemy wywołać natywnych funkcji i użyć wszystkich możliwości wbudowanych w platformę. Aplikacje webowe mogą być zatem tylko prostymi aplikacjami, które wyświetlają treści, np. nowiny, artykuły, blogi, a w najbardziej skomplikowanym stopniu pozwalającymi na wypełnienie formularza i przesłanie danych użytkownika.

Dlatego też jeżeli chcemy tworzyć bardziej zaawansowane, szybkie i działające w każdych warunkach programy, musimy korzystać z natywnych dla danej platformy narzędzi, czyli tzw. Natywnego API (Access Programming Interface) lub SDK (Standard Development Kit).

2.2 Cross-platform tools

W odpowiedzi na stawiany problem powstał szereg narzędzi programistycznych kryjących się pod wspólną nazwą "cross-platform tools". Są to narzędzia umożliwiające pisanie jednego kodu, który będzie wykonywalny na wielu platformach. Jest to ogromna oszczędność, ponieważ dzięki temu nie trzeba implementować tych samych algorytmów wielokrotnie w zależności od liczby platformy. Przyjrzyjmy się zatem współczesnym najpopularniejszym rozwiązaniom:

2.2.1 Apache Cordova (Phonegap)

Jest to jeden z czołowych frameworków cross kompilujących. Niewątpliwą zaletą jest korzystanie za darmo. Wymaga od znajomości HTML, JavaScripta i CSS, ponieważ kod napisany przy użyciu tych języków jest następnie osadzany w WebView – komponentem będącym bardzo prostą przeglądarką i będącym w stanie wykonać instrukcje języków

webowych. To poprzez javascriptowe funkcje zyskujemy dostęp do natywnego SDK danej platformy.

Niestety w dalszym ciągu brakuje wielu implementacji. Lwia część projektu opiera się na pluginach. Jest on typu open-source w związku z czym jesteśmy skazani ich na łaskę i niełaskę kontrybutów projektu. Dodatkowym minusem była w ostatnich latach słaba wydajność wynikająca z wykonywania kodu na silniku technologii webowych, co siłą rzeczy musi być wolniejsze niż wykorzystanie natywnych komponentów UI platformy. W celu uzyskania płynności, musiały być stosowane optymalizacje. Jednak nie zawsze jest to proste, zwykle zaciemnia kod i może generować błędy.

2.2.2 Appcelerator (Titanium)

Chyba drugi co do popularności framework, który również wykorzystuje API javascriptowe. Jednak w tym przypadku UI jest budowany w oparciu o natywne dla platformy komponenty UI, co powoduje wzrost wydajności i płynniejsze działanie aplikacji w porównaniu do Phonegapa.

Z drugiej strony, wprowadza to fragmentację kodu, czyli rozbieżność implementacji dla konkretnych platform, ponieważ musimy skupić się na jej natywnym SDK. Mimo, iż złamanie zasady uniwersalności kodu zachodzi tylko w niektórych miejscach, należy wciąż mieć to na uwadze.

Do produktu dołączone jest solidne środowisko Titanium IDE. Poza tym, Titanium oferuje dostęp do usług w chmurze, dzięki czemu jesteśmy w stanie w prosty sposób utworzyć backend aplikacji.

2.2.3 Adobe AIR

Framework jest produktem firmy Adobe i opera się na innych wynalazkach Adobe, czyli Adobe Flash, Flexie i ActionScripcie. Dodatkowo – za wyjątkiem platform mobilnych – można posiłkować się HTML'em i JS'em. Framework wydaje się dosyć dojrzały, wspiera szeroką gamę platform oraz pozwala na budowanie ciekawych i zaawansowanych elementów graficznych, a zwłaszcza animacji.

Najbardziej niekorzystnym czynnikiem oddziaływanym na ten framework jest obecnie panująca tendencja do odchodzenia od technologii Flash na korzyść innych. Dodatkowo sam projekt wydaje się być molochem. Zawiera nad wyraz wiele niepotrzebnych funkcji i staje się przez to wysoce skomplikowany.

2.2.4 Xamarin

Xamarin jest własnością firmy Microsoft. Stosowanym językiem programowania jest C#, co niewątpliwie jest zaletą, z uwagi na licznosć istniejących bibliotek dla tego języka oraz samą jego nowoczesność.

Choć obsługuje tylko platformy mobilne, to trzeba przyznać robi to bardzo dobrze. Wspiera wiele ze specyficznych tylko dla nich technologii, np. technologia Beacon czy notyfikacje Push. W konstrukcji aplikacji również może dochodzić do punktów krytycznych, kiedy kod musi zostać ulec fragmentacji. Jednak według statystyk kod wspólny stanowi przeciętnie 60-70%. Jeżeli dodatkowo zastosuje się pełne podejście MVC (lub MVVM) może to być nawet 90%.

Xamarin ma własne IDE o nazwie Xamarin Studio, które jest bardzo dobrze wyposażone. Można korzystać także ze standardowego Visual Studio, dla którego przygotowano specjalne pluginy.

W zapiszu Xamarina znajduje się inne potężne narzędzie - Xamarin Test Cloud, które umożliwia przetestowanie aplikacji i jej wyglądu na ogromnej liczbie różnych urządzeń. Usługa chwali się, że można symulować ich nawet 1800!

2.2.5 Unity

To cross-platformowy silnik do tworzenia gier utworzony przez Unity Technologies, obecnie w wersji Unity 5. Tworzone w nim gry wideo mogą działać na PC-tach, konsolach, urządzeniach mobilnych i stronach internetowych. Wersja Unity Personal jest darmowa, jednak jeżeli chcemy wykorzystywać Unity w celach komercyjnych wymagana jest płatna wersja Unity Pro.

Unity opiera swoje działanie na Direct3D na systemach Windows i Xbox, OpenGL na Mac/Windows czy OpenGL ES na Androidzie i iOS-ie.

Tworzenie aplikacji bazuje na wykorzystaniu języka Mono - implementacji .NET'a, UnityScripta (wywodzący się z JS), C# czy Boo.

Unity jest bardzo potężnym narzędziem i chyba największym faworytem, jeżeli chodzi o tworzenie gier. Jednak jest to jedyny cel, w jakim można go wykorzystać, gdyż próby utworzenia standardowej aplikacji biznesowej są karkołomne.

2.3 Qt 5 – nowe narzędzie do cross-kompilacji.

Qt jest znany od lat środowiskiem utworzonym pod język C++, zwłaszcza do tworzenia GUI. Jednak dopiero od wersji 5, jego twórcy zdecydowali się uczynić z niego również narzędzie do cross-kompilacji. Mimo, że technologia 5.0 została wydana na przełomie 2012 i 2013 roku, to wciąż prężnie się rozwija. Od tego czasu doczekaliśmy już wersji 5.4 (już niedługo 5.5), a w tym czasie od podstawy technologii znaczco rozszerzono liczbę wspieranych platform i dodano bardzo wiele funkcjonalności i optymalizacji.

Wspierane platformy [1]:

- Windows XP, Vista, 7, 8 (32- i 64-bit)
- Linux Red Hat Enterprise, Ubuntu
- OSX 10
- Embedded Linux arm-gnueab Ubuntu
- QNX
- Windows Embedded 7 – armv4i
- iOS
- Android
- Windows Phone.

Qt5 to przede wszystkim prostota. Do tworzenia aplikacji wykorzystuje się Qt Quick, czyli moduł składający się z opracowanego przez twórców Qt języka QML oraz jego silnika. Język QML przypomina składnię javascript i służy przede wszystkim do tworzenia UI. Sam w sobie jest językiem deklaratywnym, a nie imperatywnym, przez co nie musimy pisać kodu jako instrukcji krok po kroku, ale zawrzeć w kodzie tylko warunki, które mają zostać spełnione, a

wszystkie zmiany zmiennych będą śledzone i zgłasiane poprzez emisję sygnałów. W rezultacie pisany kod jest uproszczony, krótszy i przejrzysty.

Oprócz języka QML, możemy korzystać z czystego JavaScripta, a nawet C++. Wymaga to utworzenia odpowiednich pomostów pomiędzy technologiami, jednak okazuje wielce pomocne przy pisaniu bardziej zaawansowanej logiki i algorytmów.

Warto nadmienić, że QML wspiera odtwarzanie multimedii, czyli plików audio oraz video. Pozwala na to specjalna biblioteka QtMultimedia 5.0. Posiada także Qt.WebView i Qt.WebEngine, czyli komponenty do odtwarzania treści z internetu.

Jedną z największych zalet Qt są jednak możliwości silnika graficznego i wydajność tworzonego interfejsu użytkownika. Rysowanie interfejsu użytkownika (UI) korzysta bowiem z OpenGL, w związku z czym za wyświetlanie komponentów odpowiada karta graficzna urządzenia. Responsywność i płynność tworzonych widoków oraz animacji jest naprawdę zauważalna.

Co ważne, Qt wspiera obsługę sensorów, o ile takie znajdują się na płycie głównej urządzenia. Za pomocą wbudowanych kontrolerów możemy otrzymywać odczyty z akcelerometrów, żyroskopów, termometrów oraz modułów GPS, dzięki czemu tworzone aplikacje mogą być interaktywne i bardziej zaawansowane.

Ukoronowaniem zalet Qt jest jego własne IDE, czyli środowisko programistyczne o nazwie Qt Creator. Znaczco przyspiesza proces tworzenia aplikacji. Posiada dedykowane narzędzia, np. edytor graficzny do tworzenia GUI oraz wsparcie dla wielu systemów kontroli wersji (Git, Mercurial itp.). Umożliwia debugowanie wszelkiego kodu (QML, Javascript, C++), a skonstruowaną implementację szybko przeniesiemy na dowolne urządzenie obsługiwane przez Qt.

3 Cel pracy

Celem pracy jest utworzenie panelu sterowania inteligentnym domem przy użyciu technologii Qt. Jedna implementacja ma być wykonywalna na platformach desktopowych i mobilnych, a także ma poprawnie wyświetlać się na ekranach różnych wymiarów i różnego upakowania

pikseli na ekranie - zarówno w tzw. zorientowaniu wertykalnym jak i horyzontalnym (urządzenia mobilne).

Podstawowa implementacja panelu sterowania będzie zawierać możliwość zdalnego zarządzania światłami, które uprzednio zdefiniujemy w systemie inteligentnego domu, oraz odczyt sensorów, np. termometra, ugotowanie wody poprzezłączenie czajnika, uzbrojenie i rozbicie alarmu, a także odtwarzanie obrazu z kamery. W tym celu zaplanowane zostały konkretne widoki oraz przepływ (nawigacja) aplikacji.

3.1 Przepływ (nawigacja)

Aplikacja będzie składać się z paru ekranów realizujących pewne funkcjonalności opisane poniżej. Każdy ekran musi być łatwo identyfikowalny, dlatego potrzebne będzie umieszczenie jego tytułu w widocznym miejscu. W tym celu zdecydowałem, że potrzebny będzie tzw. **Toolbar**, czyli charakterystyczny, oddzielony od treści pasek na górze ekranu, w którym można tytuł zawrzeć.

Oprócz tego w Toolbarze znajdzie się rozwiązanie nawigacji wstecznej. W aplikacji postanowiłem użyć popularnego z platform mobilnych przycisku **UP** znajdującego się w lewym górnym rogu każdego z zagłębionych widoków. Jego użycie będzie cofało ekran do poprzedniego.

Poza tym, dobrze byłoby obsługiwać nawigację wsteczną również w alternatywny sposób, jak chociażby przy użyciu natywnego przycisku **BACK** na urządzeniu z systemem Android lub **SPACE** na komputerze typu desktop.

Jako ostatnie, ważnym do wdrożenia będzie mechanizm odświeżania danych na żądanie. Jeżeli bowiem z aplikacji będą korzystać dwie lub więcej osób jednocześnie, użytkownik będzie musiał mieć możliwość aktualniania stanów urządzeń.

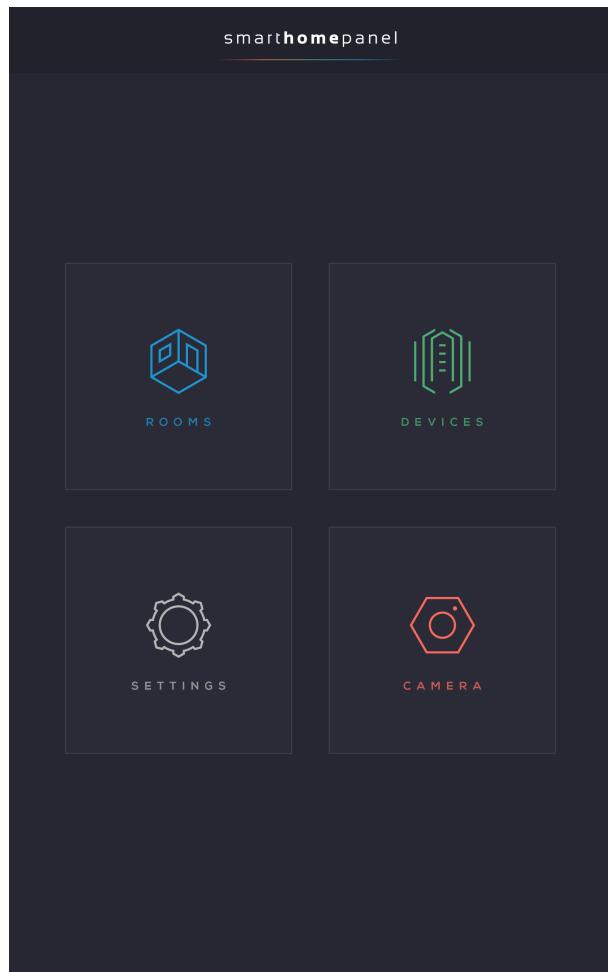
W tym celu na każdej liście lub gridzie zostanie wdrożony tzw. „Pull to refresh”, czyli interakcja użytkownika, dzięki której poprzez przeciągnięcie listy w górę zostanie wywołane zapytanie do serwera z prośbą o nowe dane. Gdy serwer zwróci odpowiedź, nastąpi przeładowanie danych.

3.2 Serwer centralny

W projekcie będzie konieczne wdrożenie dodatkowego sprzętu - serwera centralnego, który będzie posiadał bazę danych wszystkich **urządzeń wykonawczych** podlegających kontroli w inteligentnym domu (świateł, czujników itp.). Taki serwer umożliwi synchronizację stanów tych urządzeń, co jest kluczowe przy korzystaniu z systemu przez wielu ludzi (urządzenia typu desktop, mobilne). Serwer ma przede wszystkim za zadanie przechwytywać rozkazy wydawane przez zautoryzowanych użytkowników i przekazywać je na urządzenia wykonawcze. Oznacza to, że musi działać nieprzerwanie całą dobę, więc jego istotną zaletą byłby niski pobór prądu. Serwer z definicji musi być wpięty do sieci (chociażby lokalnej) i posiadać stały adres IP.

3.3 Dashboard

Pierwszym, powitalnym i nadziednym widokiem będzie tzw. **Dashboard**, z którego możliwe będą dalsze opcje sterowania, a także przejście do ustawień aplikacji, które pozwolą chociażby na autoryzację użytkownika.



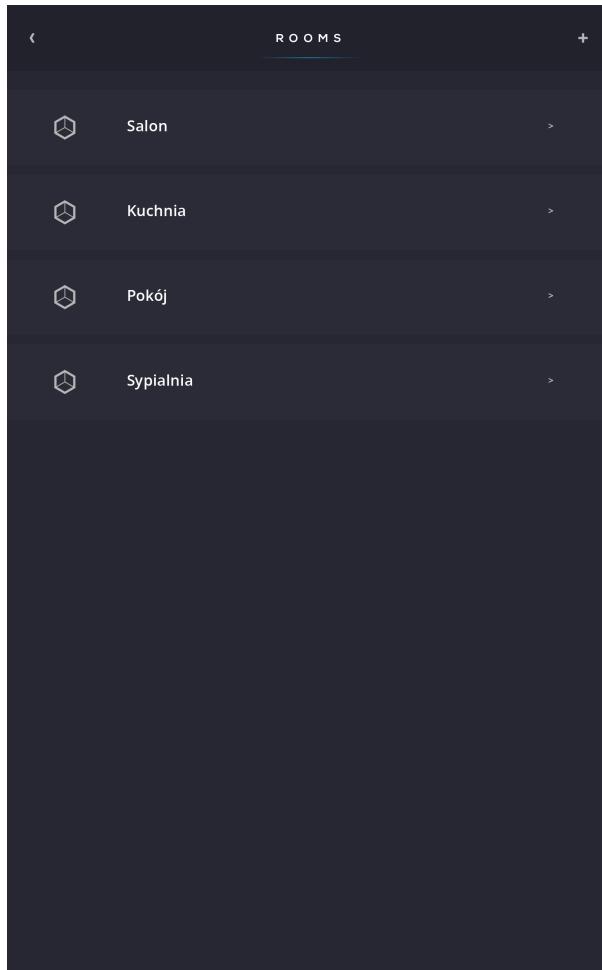
Rysunek 3 Dashboard - projekt

Będą stąd możliwe przejścia:

- do listy pokojów (**Rooms**), a której wybieramy pokój, którym chcemy zarządzać,
- do wszystkich urządzeń (**Devices**), które będą pogrupowane względem ich typu,
- do ustawień aplikacji (**Settings**),
- do odtwarzania streamowanego obrazu wideo z kamer (**Camera**).

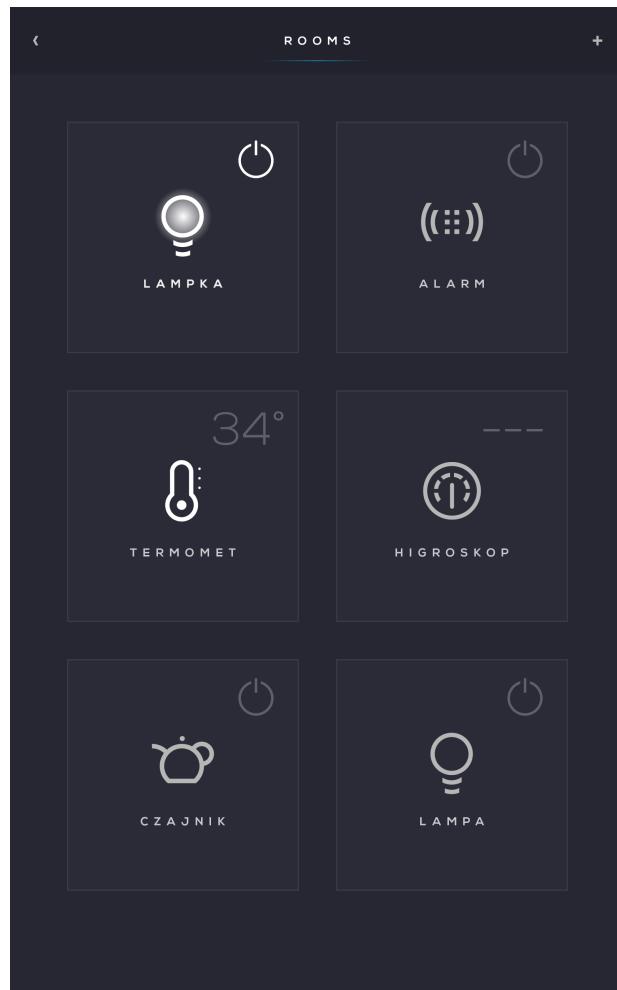
3.4 Pokoje (Rooms)

Zarządzanie pokojem. Z powodu dużej ilości urządzeń wykonawcznych, zdecydowałem, że przyjaznym dla użytkownika będzie widok, w którym wyświetla się wyłącznie urządzenia, znajdujące się w przestrzeni wybranego pokoju. Łatwiej bowiem kojarzyć przedmioty związane z konkretnym pokojem, takie jak na przykład światło główne w salonie i światło w główne kuchni. Pierwszym widokiem będzie lista wszystkich zdefiniowanych pokojów:



Rysunek 4 Rooms: Lista pokojów - projekt

A po wybraniu konkretnego:

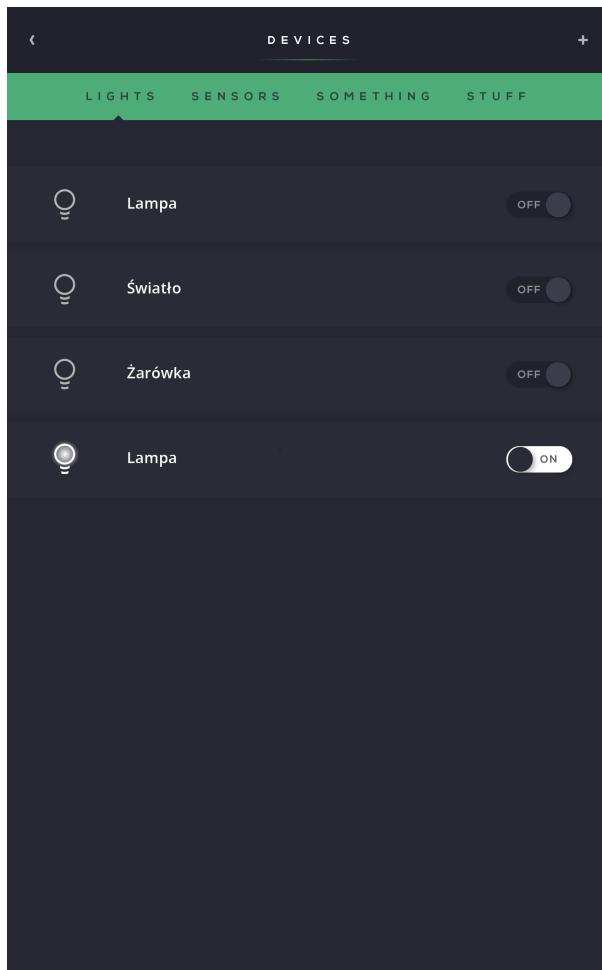


Rysunek 5 Rooms: panel urządzeń w pokoju - projekt

Tak więc będzie to tzw. grid, czyli lista elementów w postaci kafelek. Kafelki będą różnego typu, ponieważ możemy coś włączać/wyłączać – światła, alarmy, czajniki lub biernie odczytywać stan wskazań – termometr.

3.5 Urządzenia (Devices)

Z Dashboardu można oprócz ekranu zarządzania pokojem wejść w widok ze wszystkimi urządzeniami. Taka opcja istnieje na wypadek, jeżeli użytkownik zechce sterować grupą urządzeń tego samego typu – np. pozgaszać wszystkie światła, rozbroić wszelkie alarmy.



Rysunek 6 Devices - projekt - ekran z urządzeniami grupowanymi wg ich typu

U góry, na zielonym tle, będą dostępne zakładki służące do wyboru typu urządzeń – czyli światła, czujników itp.

3.6 Formularz dodawania/edykcji urządzeń.

Zarówno z widoku Rooms jak i Devices będzie dostępne tworzenie, a w zasadzie definiowanie nowego urządzenia. Opcja będzie dostępna z przycisku akcji „+” znajdującego się w prawym górnym rogu. Jest to standardowe umiejscowienie przycisków akcji na urządzeniach mobilnych. W moim przekonaniu jest ono wysoce intuicyjne i widoczne.

The screenshot shows a dark-themed user interface for adding a new device. At the top center, it says 'NEW DEVICE'. Below that are four input fields: 'Name' (empty), 'Type' (set to 'Light'), 'Room' (set to 'Kuchnia'), and 'IP' (empty). To the right of the 'Type' and 'Room' fields are small dropdown arrows. At the bottom center is a 'CONFIRM' button with a green checkmark icon.

Rysunek 7 Formularz dodawania/edykcji urządzenia - projekt

Aby zdefiniować urządzenie należy podać jego unikalną nazwę, typ, adres w systemie oraz do jakiego pokoju będzie należał.

Ponadto, ten sam formularz może służyć do edycji istniejących już urządzeń, jeżeli zajdzie taka potrzeba. Trwałe usunięcie urządzenia ma być dostępne z menu akcji, czyli z prawego górnego rogu.

3.7 Kamera

Będzie ekranem, który umożliwi podgląd obrazu z web kamer podłączonych do systemu. Obraz będzie streamowany na żywo.

3.8 Ustawienia (Settings)

Widok ustawień ma służyć do podania adresu serwera centralnego, obsługującego „inteligentny dom”, czyli adres IP wraz z portem aplikacji serwerowej. Ponadto, użytkownik będzie musiał podać login i hasło, które umożliwiają mu zalogowanie do systemu. Osoby niepowołane nie będą w ten sposób miały możliwości zarządzania intelligentnym domem.

4 Implementacja

4.1 Podstawy

Aby załadować aplikację z kodu QML należy w głównej klasie main.cpp zawrzeć poniższe instrukcje:

```
int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    QQmlContext* context = engine.rootContext();
    engine.load(QUrl(QStringLiteral("qrc:///main.qml")));

    return app.exec();
}
```

Dalszy kod aplikacji zależeć będzie wyłącznie od plików i zasobów QML. Pierwszym krokiem jaki należy podjąć to zdefiniowanie okna aplikacji:

```
ApplicationWindow {
    id: root
    width: 600
    height: 600
    ...
}
```

Wymiary 600x600 to tylko początkowe wymiary okna w pikselach na komputerach typu desktop. Na urządzeniach mobilnych nie możemy sterować wymiarami okna aplikacji, ponieważ zawsze są one pełnoekranowe.

Jeżeli chodzi o wymiarowanie graficzne, kluczową kwestią będzie zdefiniowanie wielkości **u** (unit - czyli jednostka aplikacji):

```
property int u: 1
```

Jest to o tyle ważne, iż tzw. upakowanie (zagęszczenie) pikseli jest zależne od ekranu. Zmusza nas to do zdefiniowania własnej jednostki miary, ponieważ operowanie na pikselach jest nieefektywne.

Zatem na podstawie wielkości dostarczanej przez komponenty Qt **Screen.logicalPixelDensity** (liczba pikseli na 1 mm) oraz wielu eksperymentach byłem w stanie zdefiniować jej optymalną wielkość, co ma miejsce przy załadowaniu aplikacji:

```
u = Screen.logicalPixelDensity
if (u > 6) {
    u = 6;
}
```

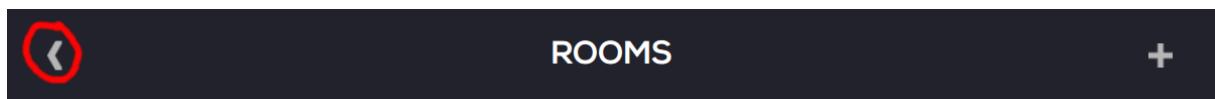
Dla wykorzystywanych w projekcie telefonów wynosi ona:

Tabela 1 Wartości logicalPixelDensity dla poszczególnych urządzeń oraz wprowadzona wartość u.

	logicalPixelDensity	u
Macbook Pro (OS X)	2.834645669291339	2
Samsung Galaxy Tab S (Android)	5.669291338582678	5
LG Nexus 5 (Android)	8.503937007874017	6

4.2 Przepływy (nawigacja)

Bardzo istotnym aspektem tworzenia aplikacji jest zdefiniowanie jej przepływu i narzędzi nawigacji. Rozpoczynając korzystanie aplikacji z widoku Dashboard będą możliwe przejścia w głąb. Kiedy takie przejścia nastąpią muszą istnieć jasne dla użytkownika opcje powrotu. Głównym i bardzo intuicyjnym rozwiązaniem jest przycisk znany z systemów mobilnych **przycisk UP**, znajdujący się w lewym górnym rogu każdego ekranu:



Rysunek 8 NavigationBar z zaznaczonym przyciskiem UP.

Ikona strzałki w lewo jest już dobrze znana użytkownikom na przestrzeni lat. Drugą najbardziej intuicyjną na urządzeniach mobilnych opcję będzie wykorzystanie **przycisku fizycznego BACK** np. na urządzeniach z Androidem oraz **przycisku SPACE** na desktopach.

By zrealizować implementację powyższych założeń kluczowe są 3 elementy:

1. **StackView** - jest to komponent, do którego możemy dokładać kolejne widoki, a te będą ułożone w formie stosu.

```
StackView {  
    id: stackView  
    anchors.fill: parent  
    initialItem: dashboardComponent  
    focus: true  
  
    Keys.onSpacePressed: {  
        flowManager.goBack();  
  
    }  
  
    Keys.onBackPressed: {  
        flowManager.goBack();  
  
    }  
}
```

Komentując kolejne linijki StackView: rozciągam na całą szerokości ekranu. Definiuję, że początkowym elementem ma być Dashboard. Aby przechwytywać eventy z przycisków muszę ściągnąć focus na ten element. Kolejne definicje to obsługa przycisków SPACE i BACK.

2. **FlowManager** - to mój własny komponent, w którym zawarłem wszystkie przejścia, czyli jakie widoki mają być dorzucane do StackView oraz jak ma działać powrót. A oto uproszczona funkcja js:

```
function goBack() {
    if (stackView.depth > 1) {
        ...
        stackView.pop();
        stackView.currentItem.refreshUI();
        ...
    } else {
        Qt.quit();
    }
}
```

Działa ona na zasadzie ściągania widoków ze StackView i odświeżania elementów poprzednich. refreshUI() - odświeżenie są dodatkową funkcją, jednakże niezmiernie ważną ze względu na zmiany stanów urządzeń wpiętych w "inteligentny dom", np. zmiany temperatury. Jeżeli natomiast dojdziemy do ostatniego widoku (Dashboard), to kolejny powrót spowoduje zamknięcie aplikacji.

3. **NavigationBar** - to również własny komponent, który będzie powielany na każdym widoku. Będzie także zawierał przycisk UP oraz ewentualne inne akcje.

```
Rectangle {
    anchors.top: parent.top
    anchors.left: parent.left
    anchors.right: parent.right
    height: 24*u
    SquareImage {
```

```
height: parent.height
width: parent.height
imgSource: "qrc:/img/img/icon_up.png"
imgFill: 0.30
visible: stackView.depth > 1

MouseArea {
    anchors.fill: parent
    onClicked: flowManager.goBack();
}

...
}
```

NavigationBar to prostokąt z własnym kolorem, który zawiera w sobie ikonę.

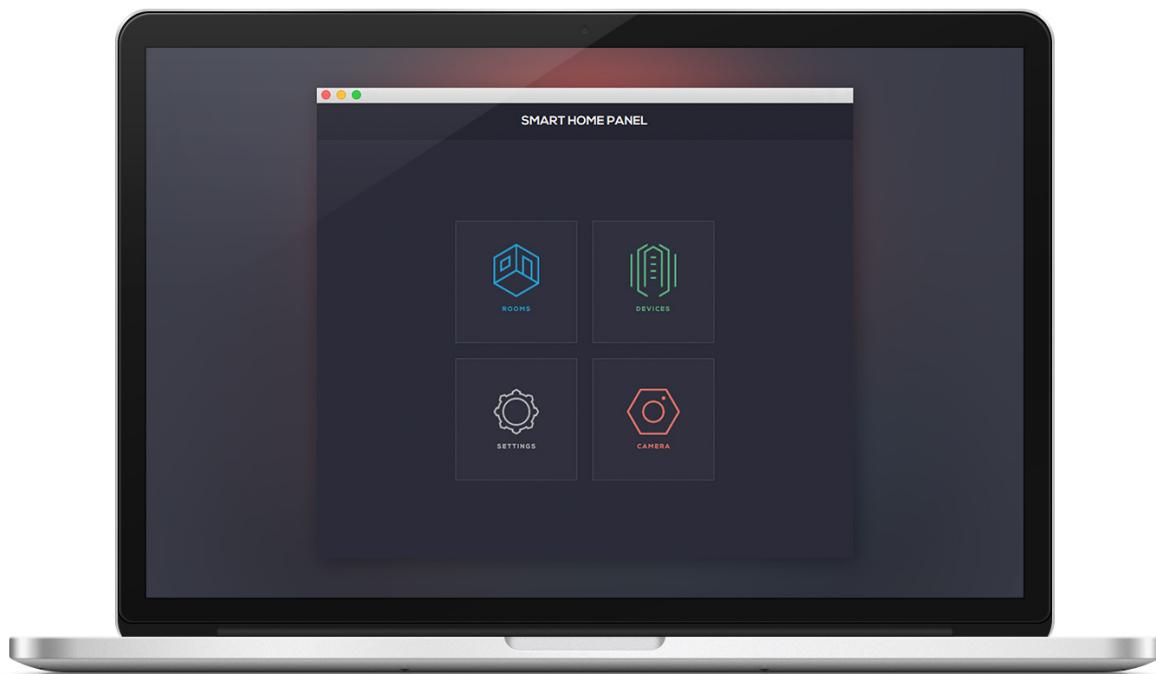
4.3 Inicjalizacja

W main.cpp następuje także inicjalizacja czcionek. Poprzez Loadery jest ona asynchroniczna i nie spowalnia rozruchu aplikacji.

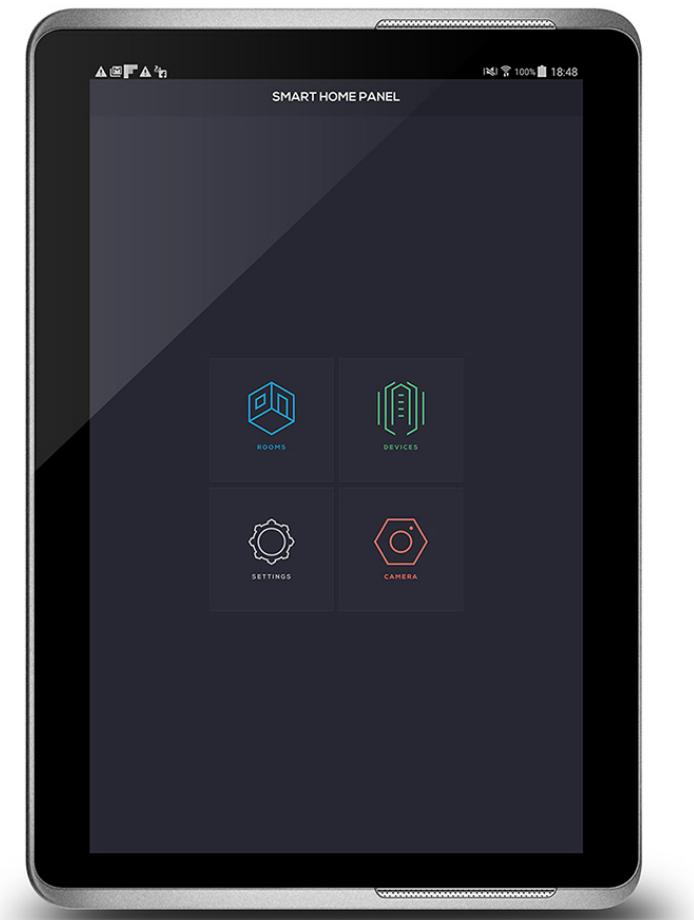
```
FontLoader{ id: nexaLight; source: "qrc:/font/fonts/Nexa_light.otf" }
FontLoader{ id: nexaBold; source: "qrc:/font/fonts/Nexa_bold.otf" }
```

Do części inicjalizacyjnej należy również zadeklarowanie komponentów wszystkich ekranów (widoków) w aplikacji, jednak warto tu zaznaczyć, że Qt nie rysuje widoków, które nie są widoczne. Jest to bardzo istotna optymalizacja zasobów procesora.

4.4 Dashboard



Rysunek 9 Dashboard - rezultat na Macbook Pro 15'



Rysunek 10 Dashboard - rezultat na tablecie Samsung Galaxy Tab S

Aplikacja charakteryzuje się 3 ważnymi funkcjonalnościami: Rooms, Devices oraz Camera. Potrzebna jest także opcja wejścia w ustawienia (Settings). Aby wyświetlić te cztery opcje w formie kafelek oddalonych od siebie w równych odstępach posłuży element **Grid**:

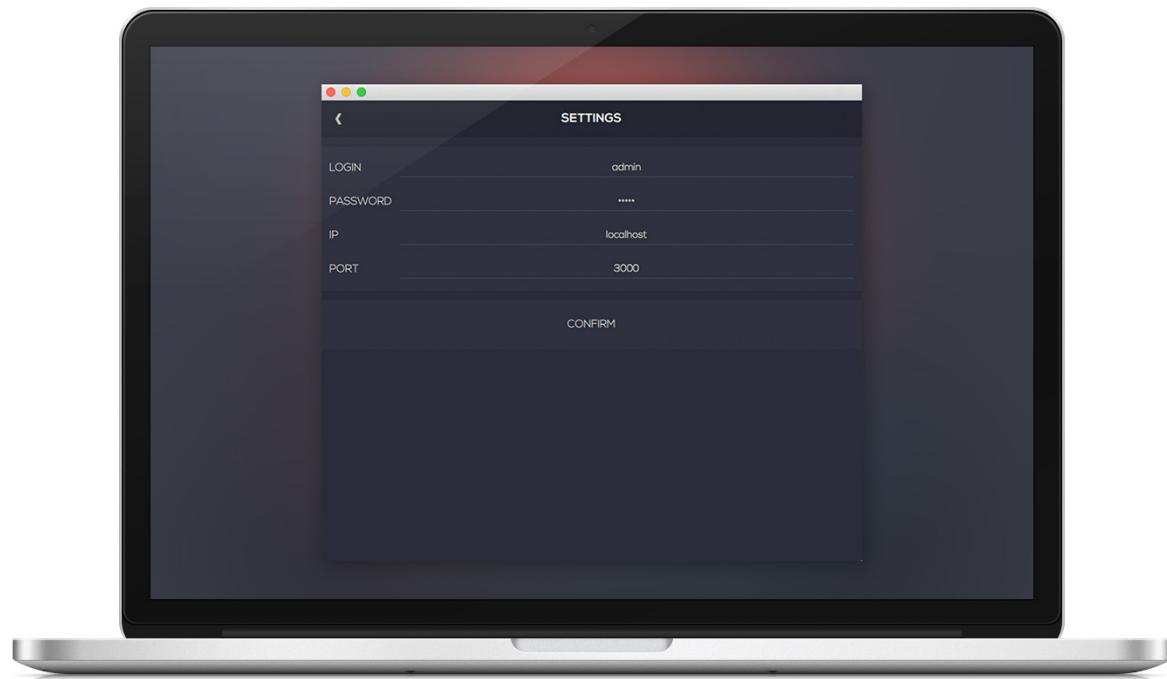
```
Grid {  
    id: grid  
    columns: 2  
    spacing: 20  
    anchors.centerIn: parent  
    ...  
}
```

Możemy do Grida wstawić dowolne elementy, a te zostaną poukładane w 2 kolumnach i będą miały między sobą odstęp 20px.

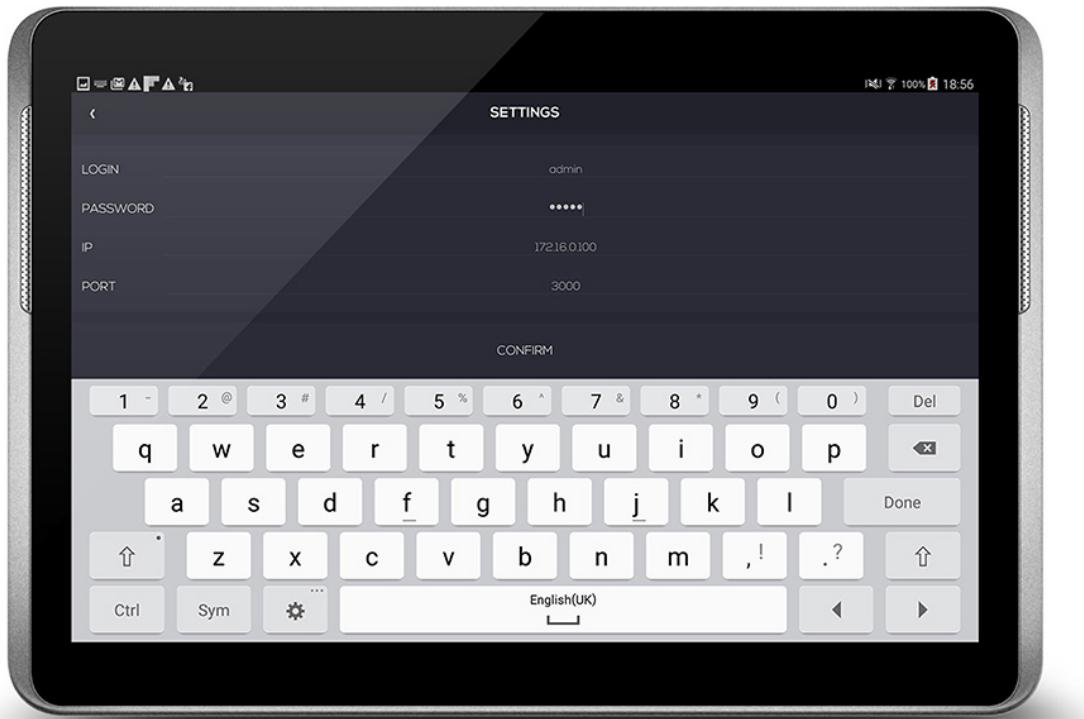
```
Card {  
    MouseArea {  
        anchors.fill: parent  
        onClicked: {  
            flowManager.showRooms();  
        }  
    }  
  
    Image {  
        id: roomsImage  
        width: 30*u  
        height: 35*u  
        mipmap: true  
        anchors.bottom: parent.bottom  
        anchors.bottomMargin: 30*u  
        anchors.horizontalCenter: parent.horizontalCenter  
        source: "qrc:/img/img/icon_rooms.png"  
    }  
  
    Text {  
        anchors.bottom: parent.bottom  
        anchors.bottomMargin: 20*u  
        anchors.horizontalCenter: parent.horizontalCenter  
        color: Color.LIGHT_BLUE  
        text: "R O O M S"  
  
        font.pixelSize: 4*u  
        font.family: nexaBold.name  
    }  
}
```

Zatem wstawiamy 4 elementy kafelek powyższego typu. Card - czyli tło kafelki to prostokąt z obwódką. Kliknięcie w kartę powoduje przeniesienie do innego ekranu przy pomocy FlowManagera. Na karcie umieszczamy obrazek tak, aby znajdowa się 30u licząc od dołu karty, a zaraz pod nim umieszczamy tekst z odpowiednim kolorem.

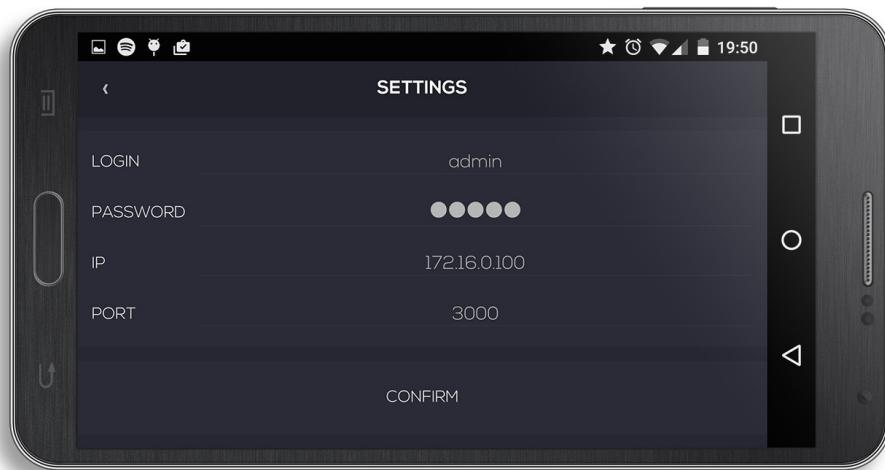
4.5 Ustawienia (Settings)



Rysunek 11 Settings - rezultat na Macbook Pro 15'



Rysunek 12 Settings - rezultat na Samsung Galaxy Tab S (układ horyzontalny) - wraz z otwartą klawiaturą



Rysunek 13 Settings - rezultat na LG Nexus 5 (układ horyzontalny)

Kolejnym, najprostszym do omówienia ekranem będą Ustawienia aplikacji. Do tego potrzebowałem utworzyć prosty formularz zbierający wartości tekstowe. Ponieważ ów fomularz stanowi powtarzający się układ elementów takich jak etykieta i pole tekstowe, dobrze będzie skorzystać z dwóch nowych komponentów: **Column** i **Row**. W Row (wierszu) ułożymy obydwa elementy tak jak chcemy, a Column (kolumna) zapewni nam ich równy odstęp:

```

Column {
    id: form
    ...
    anchors.margins: Dimension.SPACING*u
    spacing: Dimension.SPACING*u

    Row {
        width: parent.width
        height: 16*u

        ShpLightText {
            id: txtLogin
            anchors.verticalCenter: parent.verticalCenter
            text: "Login"
            width: Dimension.FORM_LABEL_WIDTH*u
        }

        ShpTextField {
            id: tfLogin
            anchors.right: parent.right
            anchors.left: txtLogin.right
            anchors.leftMargin: Dimension.SPACING*u
            anchors.verticalCenter: parent.verticalCenter
            text: settings.login
            onTextChanged: {
                settings.login = text
            }
        }
    }
    ...
}

```

Wykorzystywane są tu elementy tekstowe Shp. W celu nie powielania kodu w całej aplikacji, można zenkapsulować styl komponentów tekstowych aplikacji (Shp pochodzi od SmartHomePanel) do osobnej klasy.

ShpLightText.qml

```
Text {  
    color: Color.WHITE  
    font.pixelSize: 7*u  
    font.capitalization: Font.AllUppercase  
    font.family: nexaLight.name  
}
```

To Text wykorzystujący niestandardową czcionkę nexaLight dostarczoną do aplikacji, koloru białego i z domyślną wielkością 7u.

ShpTextField.qml

```
TextField {  
    font.family: nexaLight.name  
    horizontalAlignment: TextInput.AlignHCenter  
  
    style: TextStyle {  
        textColor: Color.LIGHT_GRAY  
        background: Item {  
            Rectangle {  
                anchors.bottom: parent.bottom  
                height: 1  
                width: parent.width  
                color: Color.COMPONENT_BORDER  
            }  
        }  
    }  
}
```

To z kolei TextField, również z czcionką nexaLight, ale kolorem delikatnie ciemniejszym. Poza tym cały styl TextField polega na dodaniu białej poziomej kreski na całej długości komponentu, aby podkreślić edytowalność tego pola.

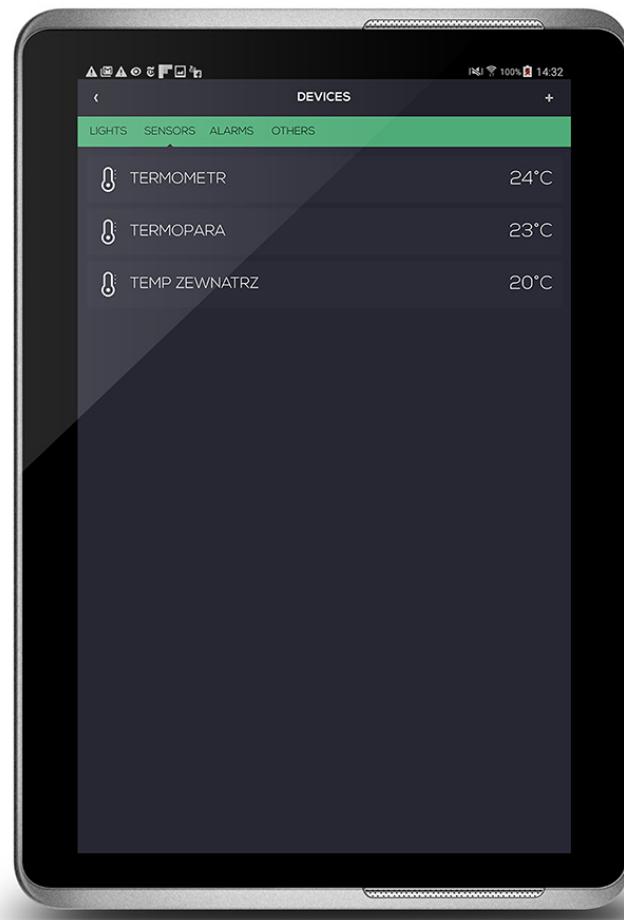
W ekranie ustawień należy podać login i hasło do systemu, oraz adres serwera, który zarządza "inteligentnym domem". Zatwierdzenie przyciskiem "Confirm" spowoduje wysłanie na serwer zapytania o autoryzację użytkownika. Gdy login i hasło okażą się prawdziwe, zarządzanie "inteligentnym domem": zostanie udostępnione. Wprowadzone dane ustawień zostaną zapisane w trwałej pamięci aplikacji tak, aby użytkownik nie musiał ich podawać przy ponownym rozruchu aplikacji. Do tego nadaje się dedykowany komponent **Settings**:

```
Settings {  
    id: settings  
  
    property string login: ""  
    property string password: ""  
  
    property string ip: "172.16.0.105"  
    property string port: "3000"  
    property string hostname: "http://" + ip + ":" + port  
}
```

4.6 Urządzenia (Devices)



Rysunek 14 Devices - rezultat na Macbook Pro 15' (zakładka z temperaturami)



Rysunek 15 Devices - rezultat na Samsung Galaxy Tab S (zakładka z temperaturami)



Rysunek 16 Devices - rezultat na LG Nexus 5 (zakładka ze światłami)

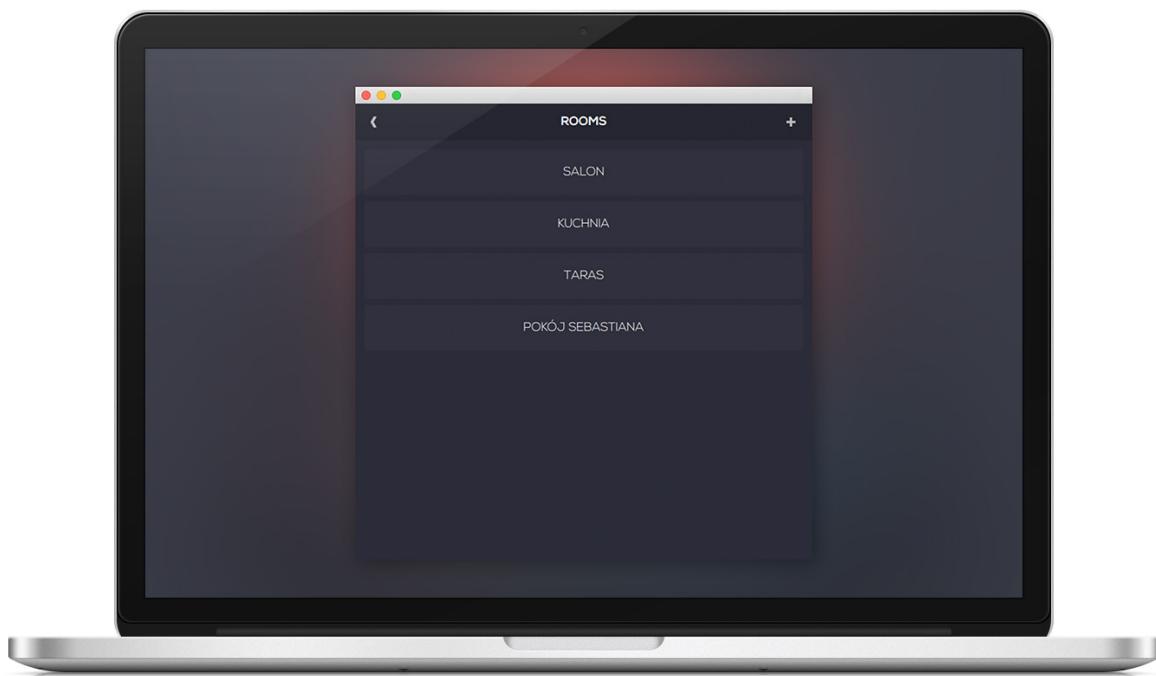
Ekran wyświetlający grupę urządzeń danego typu. Aby utworzyć górny pasek z zakładkami, specjalnie został utworzony komponent TabsView. Zakładki typów urządzeń wykonawczych zostały wyświetlane przy wykorzystaniu komponentu **Repeater**, dla którego - jak w przypadku ListView - konieczne jest zdefiniowanie *modelu* i *delegatu*.

```
Repeater {  
    id: repeater  
    model: deviceTypesModel  
    delegate:  
        ...  
}
```

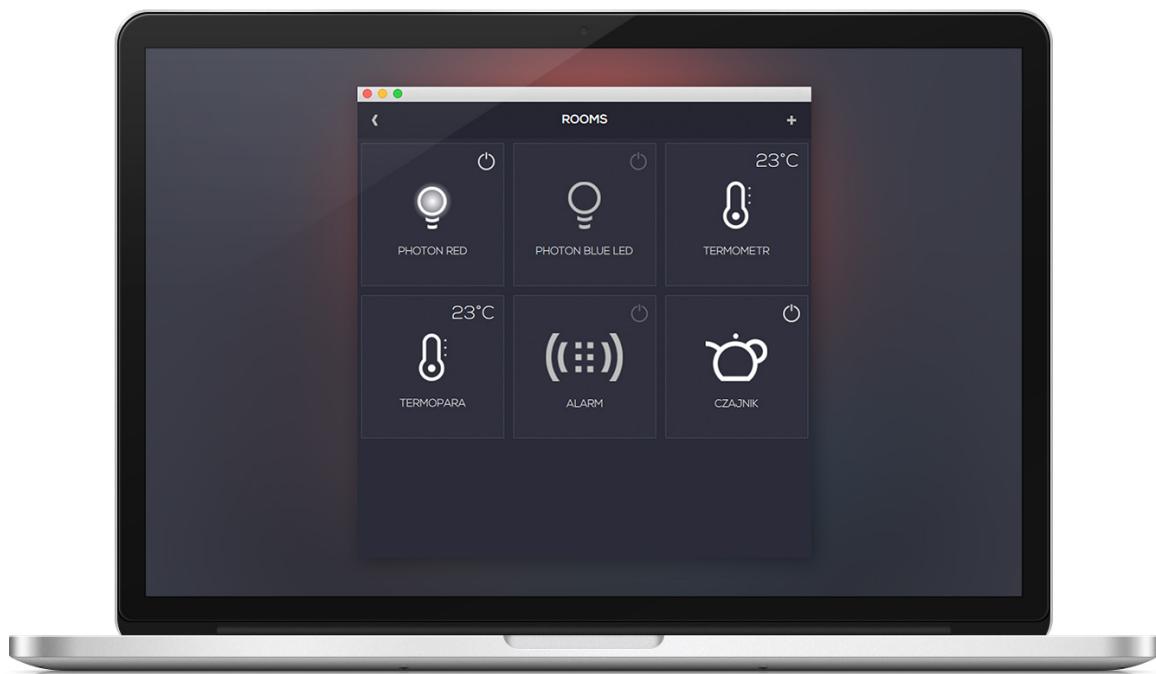
Samo wyświetlanie urządzeń wykonawczych jest rozwiązane elementem ListView, dla którego model stanowi dostarczana z serwera centralnego lista urządzeń o zadanym typie. Z racji tego,

iż wizualizacje elementów listy będą się różnić w zależności od wybranego typu (chociażby ikoną) bądź komponentem po prawej stronie (przycisk typu On/Off lub tekst), podmianie będzie również podlegać pole ListView.component.

4.7 Pokoje (Rooms)



Rysunek 17 Rooms - rezultat na Macbook Pro 15'. Zawężone okno aplikacji. Lista pokojów.



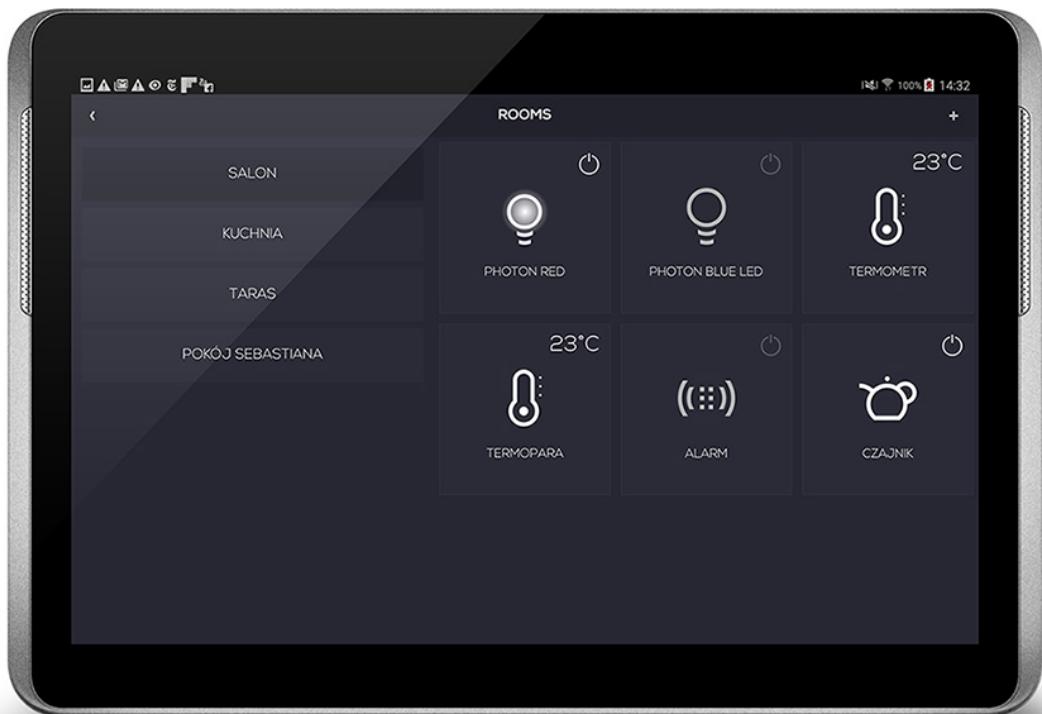
Rysunek 18 Rooms - rezultat na Macbook Pro 15'. Zawężone okno aplikacji. Panel pokoju.



Rysunek 19 - Rooms - rezultat na Macbook Pro 15'. Poszerzone okno aplikacji.



Rysunek 20 Rooms - rezultat na Macbook Pro 15'. Bardzo poszerzone okno aplikacji.



Rysunek 21 Rooms - rezultat na Samsung Galaxy Tab S (układ horyzontalny)



Rysunek 22 Rooms - rezultat na LG Nexus 5.

Ekran zarządzania pokojami. Do utworzenia listy pokojów nadał się komponent **ListView**, któremu do pola *model* należy podać listę pokojów z serwera, a do pola *delegate* komponent UI, który będzie wyświetlony dla każdego elementu pokoju.

Z kolei do wyświetlania urządzeń wykonawczych przynależących do pokoju wykorzystałem inny, bardziej ogólny komponent - **Flickable** - z niego dziedziczy bowiem ListView. We Flickable, dzięki któremu uzyskałem skrolowalność ekranu, osadziłem komponent **Flow** odpowiadający za uporządkowanie wyświetlanych elementów od lewej do prawej:

```
Flickable {  
    flickableDirection: Flickable.VerticalFlick  
    anchors.fill: parent  
    contentWidth: parent.width; contentHeight: grid.implicitHeight
```

```
Flow {  
    id: grid  
    anchors.fill: parent  
    flow: Flow.LeftToRight  
}  
}
```

Do zdefiniowanego Flow następnie należy dodać widoki urządzeń. Dla każdego typu urządzenia zdefiniowałem osobny komponent QML. Ponieważ liczba urządzeń jest dynamiczna i mogą one się różnić typem, widoki w zależności od potrzeby również są tworzone dynamicznie:

```
var lightCard = Qt.createComponent("LightCard.qml");  
var temperatureCard = Qt.createComponent("TemperatureCard.qml");  
var alarmCard = Qt.createComponent("AlarmCard.qml");  
var kettleCard = Qt.createComponent("KettleCard.qml");
```

Ciekawym aspektem jest przeliczanie długości boków kafelek, które musi spełniać założenie responsywności. Aby kafelki dobrze się wyświetlały należy uwzględnić szerokość kontenera, w którym są osadzone, i odpowiednio rozłożyć kolumny, zakładając pewną minimalną długość kafelki:

```
var columnsCount = Math.floor(containerWidth / (minWidth * u));  
cardWidth = (containerWidth / columnsCount);
```

Każdorazowe odświeżenie wiąże się z usunięciem wszystkich widoków zawartych w kontenerze i dynamicznym dodaniem nowych, które zwróci serwer.

4.7.1 Strategia 1- i 2-kolumnowa

W widokach, w których chcemy zaprezentować listę elementów oraz kontener wyświetlający treść zależną od wyboru elementu listy, powszechną strategią w przypadku szerokich ekranów jest wyświetlenie dwóch kolumn: listy i kontenera; a w przypadku węższych ekranów, np. smartfonów wyświetlenie najpierw listy, a po dopiero po wybraniu elementu listy kontenera.

Aby zrealizować tak zaplanowany UI, wykorzystano poniższą implementację:

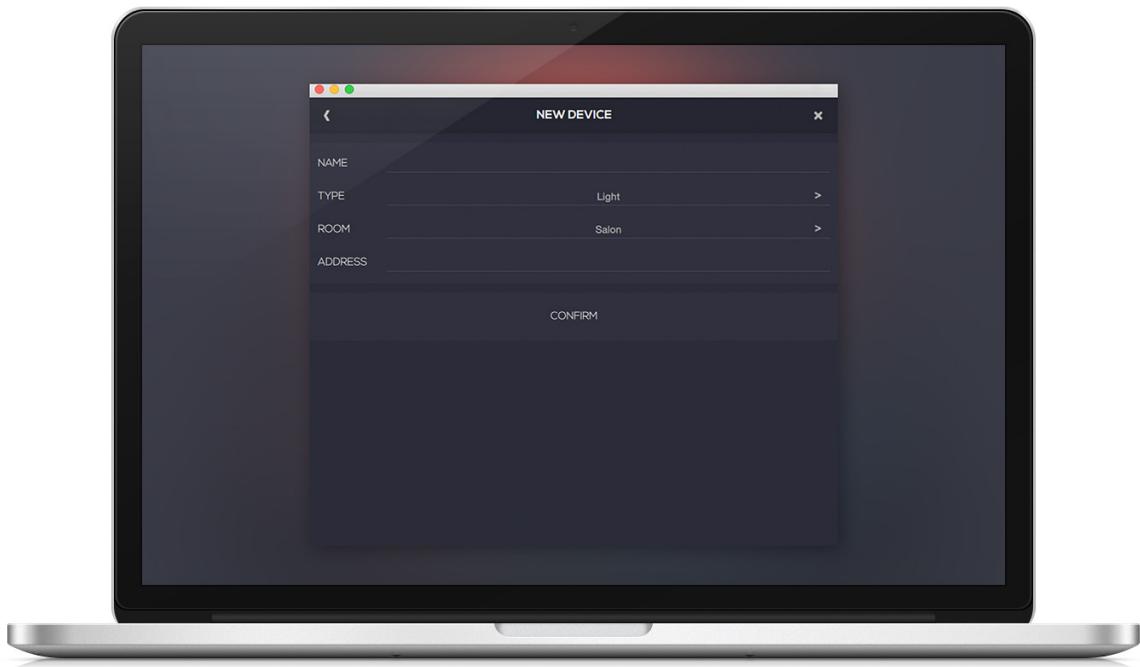
```
RoomsList {  
    id: roomsList  
    anchors.left: parent.left  
    width: (flowManager.isTwoPane ? Dimension.TWO_PANEL_LEFT_CONTAINER : 1)  
* parent.width  
    ...  
    visible: flowManager.isTopLevel || flowManager.isTwoPane  
}  
  
RoomPanel {  
    id: roomPanel  
    anchors.right: parent.right  
    width: (flowManager.isTwoPane ? Dimension.TWO_PANEL_RIGHT_CONTAINER : 1)  
* parent.width  
    ...  
    visible: !flowManager.isTopLevel || flowManager.isTwoPane  
}
```

Algorytm decyzyjny, którą strategię należy zastosować zamyka się w 1 linijce:

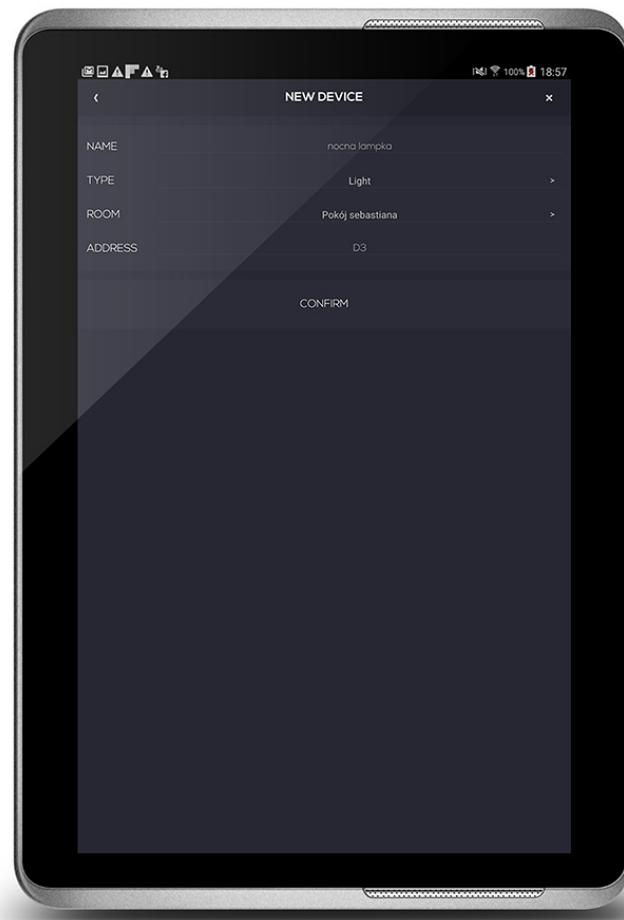
```
property bool isTwoPane: (stackView.width / u) > 320;
```

Użyta wartość 320 została dobrana empirycznie na podstawie oceny wizualnej na realnych sprzętach.

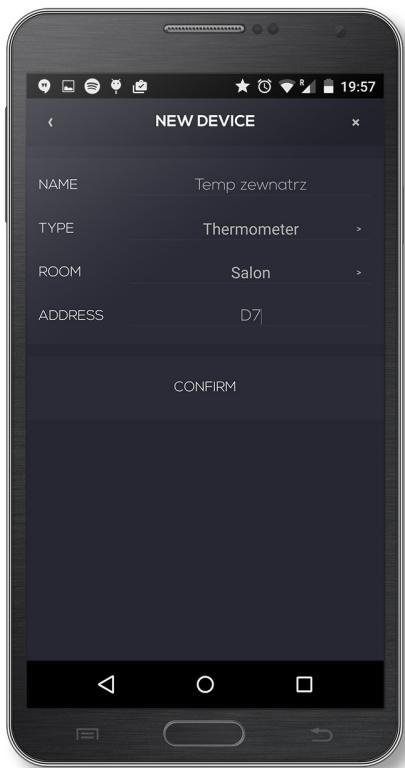
4.8 Formularz dodawania/edykcji urządzenia



Rysunek 23 Formularz dodawania urządzenia - rezultat na Macbook Pro 15'



Rysunek 24 Formularz dodawania urządzenia - rezultat na Samsung Galaxy Tab S.



Rysunek 25 Formularz dodawania urządzenia - rezultat na LG Nexus 5.

Ekran dodawania/edykcji urządzenia jest konstrukcyjnie dosyć podobny do ekranu Ustawień, ponieważ obydwa są formularzami. Dlatego użyłem tego samego podejścia. Zachowuję zasadę **MVC** czyli **Model-View-Controller**, logika UI jest odseparowana od logiki wprowadzanych danych. Elementami, które się różnią są na pewno kontrolery tych widoków. Kontrolerem w przypadku ekranu Ustawień był komponent Qt.Settings i miał on za zadaanie przechowywać proste dane.

W przypadku dodawania/edykcji urządzeń wykonawczych, logika jest bardziej skomplikowana. Został w tym celu przygotowany specjalny komponent **DeviceController** typu Qt.Item, który nie stanowi jednak elementu graficznego, ale enkapsuluje logikę formularza. Odseparowano do niego między innymi funkcje do załadowania z serwera danych o urządzeniu, dostępnych typach i pokojach (metoda load()) oraz do zapisu urządzenia na serwerze (metoda save()).

Formularz edycji urządzenia został wzbogacony również o nowy element UI **ComboBox**, który także został rozszerzony do enkapsulującego komponentu **ShpComboBox**:

```
ComboBox {  
    style: ComboBoxStyle {  
        background: Item {  
  
            SquareImage {  
                anchors.right: parent.right  
                anchors.verticalCenter: parent.verticalCenter  
                height: parent.height  
                width: parent.height  
                imgSource: "qrc:/img/img/icon_select.png"  
                imgFill: 0.80  
            }  
  
            Rectangle {  
                anchors.bottom: parent.bottom  
                height: 1  
                width: parent.width  
                color: Color.COMPONENT_BORDER  
            }  
        }  
  
        label: Text {  
            verticalAlignment: Text.AlignVCenter  
            horizontalAlignment: Text.AlignHCenter  
            text: control.currentText  
            color: Color.LIGHT_GRAY  
        }  
    }  
}
```

4.9 Kamera

Obraz z kamery został udostępniony w formie streamowanego wideo na HTTP dostępnego pod endpointem **/stream** lub bezpośrednio, np:

- 172.16.0.105:8080/stream
- 172.16.0.105:8080/stream/video.mjpeg

Streaming wideo jest poprawnie odtwarzany w takich przeglądarkach jak Google Chrome, Mozilla Firefox, Safari.

Pierwszą najprostszą próbą implementacji było wykorzystanie komponentu **Video** z wbudowanej biblioteki QtMultimedia 5.0:

```
Video {  
    id: video  
    anchors.fill: parent  
    source: settings.ip + "/stream/video.mjpeg"  
}
```

Niestety żaden z podanych adresów nie zakończył się sukcesem. Wystąpił błąd: "Video Content-Length musi be non 0".

Drugim podejściem było wykorzystanie **WebView** z biblioteki QtWebKit 3.0:

```
WebView {  
    url: settings.ip + "/stream"  
    anchors.fill: parent  
}
```

Jednak i to podejście nie rozwiązało problemu. W oknie WebView nie ładowała się żadna treść. Ponadto, to rozwiązanie nie zdałoby egzaminu na Androidzie i iOS, ponieważ w Qt nie wspiera własnego WebView, argumentując że owe platformy mają własne, bardziej doskonałe implementacje tych komponentów. Za to dostarcza do nich dostęp w pakiecie *QtAndroidExtras*.

Dla pewności sprawdziłem inne streamy wideo na żywo, np. wizję z Watykanu na YouTube:
<https://www.youtube.com/embed/86SemtwG6Lc>.

I w obydwu przypadkach ponownie nie zdały egzaminu. Wygląda na to, że Qt nie jest jeszcze gotowy do obsługi streamowanego wideo.

Ten problem ma rozwiązywać biblioteka **QtGStreamer**, jednak dotychczas doczekała się ona implementacji jedynie na komputery typu desktop.

Zatem implementacja WebCamery była niemożliwa.

4.10 Odświeżanie

Kolejnym ważnym aspektem jest odświeżanie danych aplikacji. Stany urządzeń mogą się często zmieniać, zwłaszcza jeżeli zarządzają nimi również inni użytkownicy. Dlatego też istnieje potrzeba wyświetlania najświeższych danych.

Aby rozwiązać ten problem, na ekranach, w których dane są pobierane z serwera, zaimplementowałem funkcję refreshUI(). Ma ona za zadanie pobrać odpowiednie dane z serwera i nanieść je na interfejs graficzny. Wywołanie onRefreshUI() ma miejsce przy każdym wejściu w widok.

Dodatkowo może on być wywoływany na ekranie co zadany okres czasu (np. 20 s). Aby zrealizować takie podejście można się posłużyć elementem **Timer**:

```
Timer {  
    interval: 20000  
    repeat: true  
    running: parent.visible  
    onTriggered: refreshUI();  
}
```

4.10.1 PullToRefresh

Jednak należy również zapewnić użytkownikowi odświeżenie danych na żądanie. Jeżeli miałby to robić wychodząc i wchodząc w dany ekran, byłoby to bardzo nieefektywne rozwiązanie. Bardzo modnym współcześnie rozwiązaniem jest tzw. mechanizm PullToRefresh, wykorzystywany jest chociażby przez aplikacje Facebook, Twitter, Evernote i wiele innych

zarówno webowych jak i mobilnych. Najpowszechniej występuje na listach z danymi, polegającym na wykorzystaniu gestu przeciągnięcia kursorem/palcem w dół, tak jakby chcielibyśmy zobaczyć dane powyżej, czyli intuicyjnie najświeższe.

Taki pattern PullToRefresh został właśnie zaimplementowany na widokach Rooms i Devices, gdzie mamy do czynienia z listami urządzeń. Realizacja takiego podejścia sprowadza się do poniższego kodu:

```
onDragEnded: {  
    if (atYBeginning) {  
        console.log("PullToRefresh:")  
        refreshUI();  
    }  
}
```

Dodawanego do komponentów ListView czy ogólniej Flickable. Niezmiernie przydatna okazuje się właśnie wcześniejsza implementacja refreshUI().

4.11 WebService

WebService to klasa JS utworzona w celu komunikacji z serwerem centralnym. Ponieważ serwer centralny jest serwerem w REST-owym i udostępnia dane w postaci JSON, gdy odpytane zostaną odpowiednie endpointy HTTP, klasa WebService posiada implementację komunikacji na zasadzie request-response. W tym celu zostały wykorzystane elementy wbudowane w JS XMLHttpRequest. Oto realizacja requesta:

```
var xhr = new XMLHttpRequest();  
xhr.open(verb, BASE + (endpoint? '/' + endpoint:''));  
xhr.setRequestHeader('Content-Type', 'application/json');  
xhr.setRequestHeader('Accept', 'application/json');  
var data = obj?JSON.stringify(obj):''  
console.log("Body: " + data);  
xhr.send(data);
```

A

realizacja

response'a:

```
xhr.onreadystatechange = function() {
```

```

if(xhr.readyState === XMLHttpRequest.DONE) {
    var responseJson;
    if(xhr.status === 200 && onSuccess) {
        ...
    } else if (onError) {
        ...
    }
}

```

Jeżeli wszystko pójdzie poprawnie odpowiedź przyjdzie z serwera ze statusem 200. Jeżeli gdziekolwiek po drodze wystąpi błąd, to trzeba go będzie obsługiwać.

4.11.1 Obsługa błędów

Cała logika walidacji danych w przypadku logowania użytkownika czy formularzy została umieszczona na serwerze centralnym. Jeżeli dane są niekompletne lub niepoprawne, serwer zwraca odpowiedź ze statusem błędu HTTP 400.

W celu obsługi błędów w przypadku braku połączenia z serwerem lub wspomnianych walidacji danych został utworzony komponent UI **ErrorBar**.

```

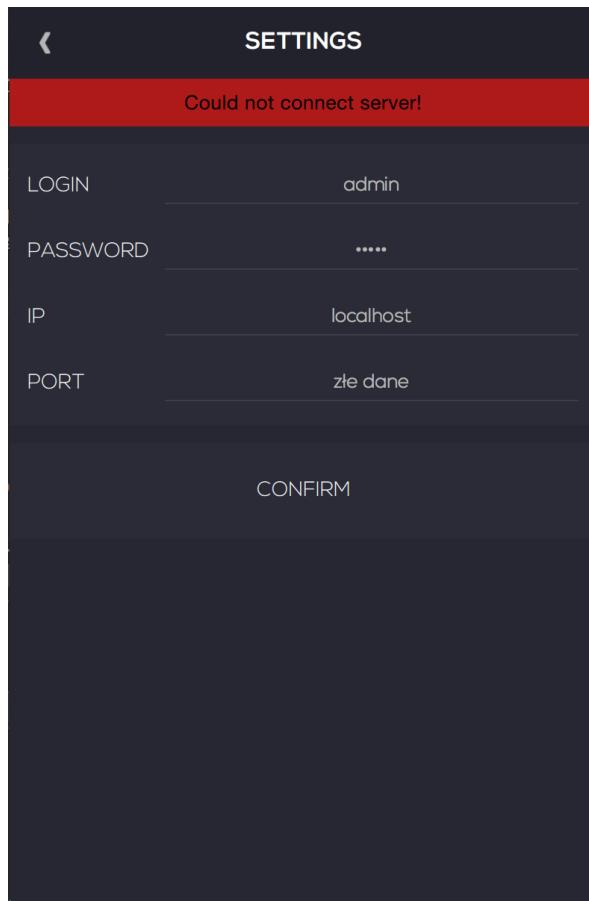
Rectangle {
    anchors.top: parent.top
    height: visible ? 16*u : 0
    width: parent.width
    color: Color.VALIDATION_RED
    visible: error !== ""
    z: 2

    property string error: ""

    Text {
        anchors.centerIn: parent
        text: error
    }
}

```

Jest dopinany w niemal każdym ekranie, a domyślnie niewidoczny, ale gdy tylko wystąpi błąd, zostanie on wpisany w pole *error* i na ekranie zagości powiadomienie w tej postaci:



Rysunek 26 ErrorBar - pasek błędu wyświetlony przy braku połączenia z serwerem.

4.12 Serwer centralny

Jako serwer centralny wykorzystałem komputer **RaspberryPI** z systemem Raspbian z rodziny Linux. Idealnie realizuje on postawione wcześniej założenia, zwłaszcza że potrzebuje tylko prądu o napięciu 5V, co zaoszczędza koszty.

Na RaspberryPI został utworzony **serwer REST** - służący do komunikacji z urządzeniami sterującymi. Wymieniane dane są serwowane w formacie JSON. Przykładowo pobrane dane do edycji urządzenia wyglądają następująco:

```
{  
  "_id": "55adf42f11493a565bb6909e",  
  "state": "24°C",  
  "name": "Termometr",
```

```
        "ip": "D5",
        "typeId": "2",
        "roomId": "5564cf5859e2d63aa63a1936"
    }
```

Serwer opiera się na technologii **Node.js** przy użyciu frameworka **Express**. Działa on na porcie 3000. Z kolei wszystkie informacje i stany urządzeń są zapisane w bazie danych **Mongo DB**. Dzięki istniejącym już bibliotekom Node.js możliwe jest operowanie na rekordach tej bazy.

Serwer REST realizuje następujące endpointy:

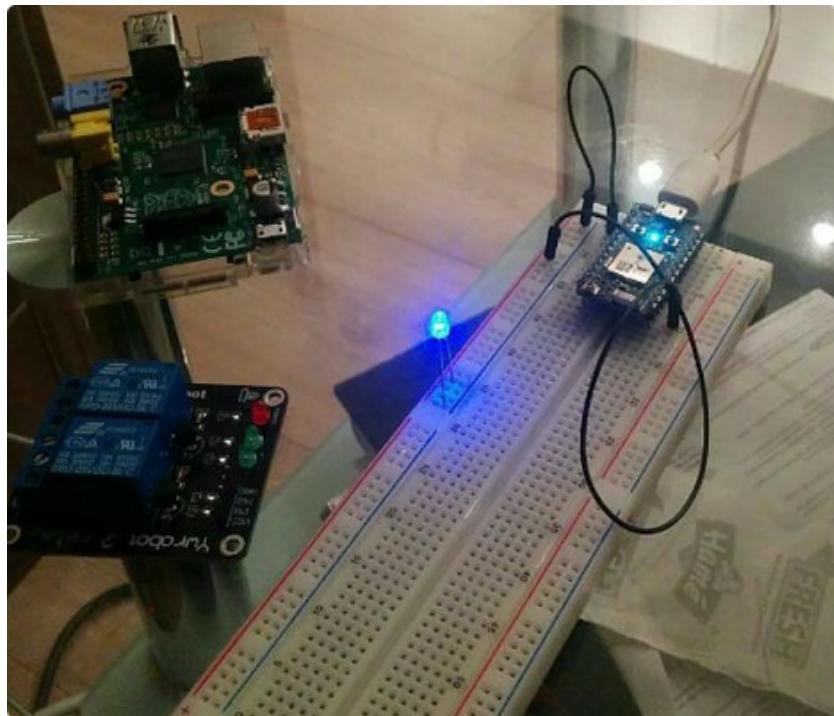
- /auth (GET) - umożliwia logowanie
- /devices (GET, POST, PUT, DELETE) - umożliwia pobranie listy urządzeń, konkretnego urządzenia, dodawanie nowego /edycję urządzenia (wraz walidacją) oraz jego usunięcie
- /rooms (GET, POST, PUT, DELETE) - umożliwia pobranie listy pokojów, dodanie pokoju, edycję pokoju oraz usuwanie pokoju.

4.13 Układ wykonawczy

Sterowane urządzenia muszą otrzymywać sygnał przez pewne medium. Do przekazywania rozkazu z serwera bezpośrednio na sterowane urządzenia zdecydowałem się użyć układu **Particle Photon**. Jest to nowość na rynku elektronicznym - płytka ukazała się na świetle dzienne w maju 2015 roku. Particle Photon to mikrokontroler podobny do mikrkontrolera Arduino - zarówno w konstrukcji jak i implementacji własnych algorytmów - ale jest połączony do sieci dzięki zawartemu na płytce modułowi WiFi. Napisany program również wgrywa się przez internet dzięki usłudze Particle Cloud.

Rozkazy serwera centralnego można było przekazywać na Particle Photon poprzez użycie firmowego SDK o nazwie ParticleJS (technologia Node.js). Funkcje tej biblioteki mogłem wywoływać z aplikacji serwera, która również była pisana w Node.js.

Zaczynając od rzeczy najprostszych byłem w stanie sterować diodą LED poprzez połączenie do odpowiednich wyjść cyfrowych (pinów) na płytce:



Rysunek 27 Podłączenie diody LED to Particle Photon.

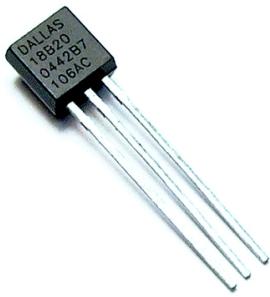
Chcąc sterować czymś większym, np. lampką biurkową, zasilaną prądem 220V, trzeba by użyć cyfrowego wyjścia Photona jako sterowania przekaźnikiem. Ponieważ dysponowałem starą lampką, wpięłem jej rozcięte przewody do przekaźnika YwRobot 2 Relay. Można również spróbować bardziej eleganckiego rozwiązania poprzez użycie przekaźnika o charakterze wkładanego do kontaktu gniazdka, aby nie przecinać kabli rzeczywistych urządzeń.



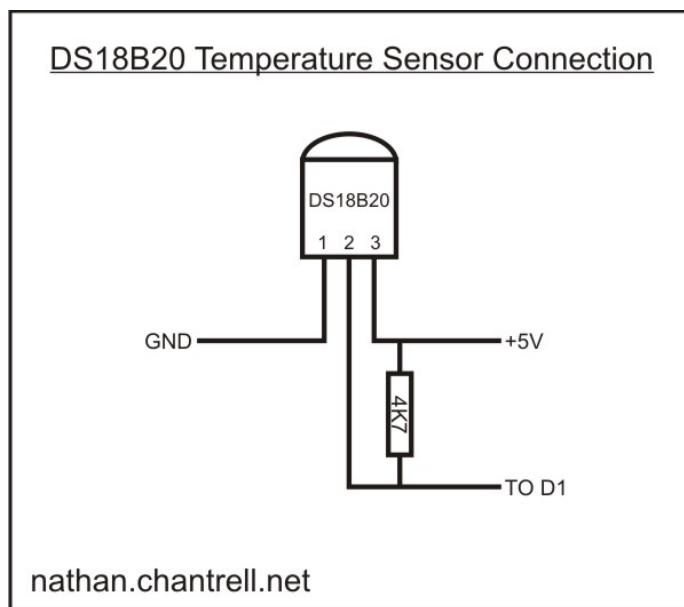
Rysunek 28 Podłączenie lampki biurkowej do Particle Photon.

W ten sposób, jeżeli układ wykonawczy jest w stanie sterować lampami (5V i 220V) o charakterze włącz/wyłącz, to można bez większego problemu podpiąć szereg innych podobnie działających urządzeń takich jak alarmy, czajniki itp.

Oprócz oświetlenia, podłączyłem do układu **termometr DS18B20**. Wymagał on podpięcia do wejścia cyfrowego przy zastosowaniu techniki OneWire. Aby odczytywać temperaturę, na Photonie musi być zaimplementowany specjalny algorytm konwerujący sygnały cyfrowe na stopnie Celcjusza. W tym celu wykorzystałem już gotowe biblioteki z języka C przygotowane na mikroprocesor Arduino.

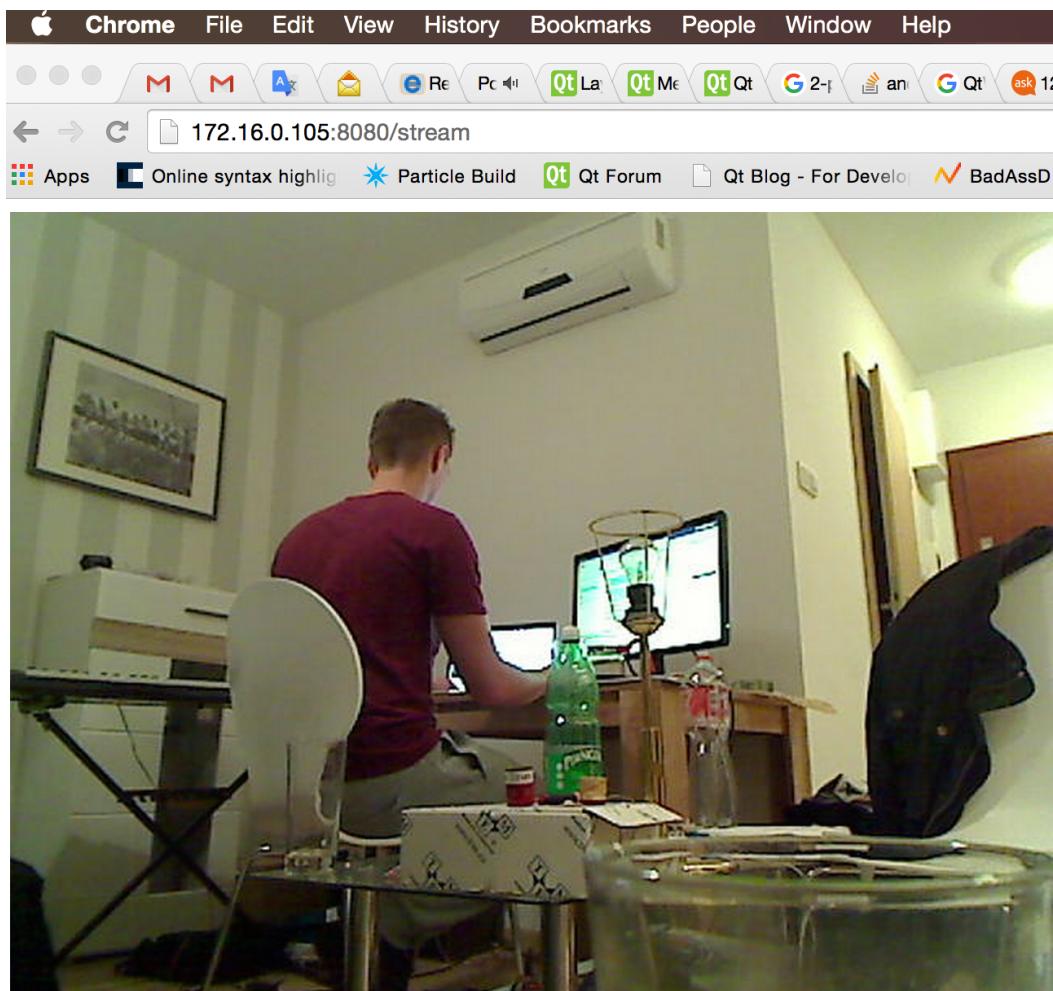


Rysunek 29 Termometr DS18B20



Rysunek 20 Schemat podłączenia termometra DS18B20 do Particle Photon.

Ponadto, do serwera centralnego została podłączona kamera **USB Logitech QuickCam Pro 9000**. Streaming obrazu wideo w czasie rzeczywistym został zrealizowany przy użyciu programu **Motion**. Obraz dostępny był na porcie 8080 serwera w postaci pliku .JPEG.



Rysunek 31 Obraz z kamery dostępny na porcie 8080

5 Podsumowanie

5.1 Wnioski

Aplikacja SmartHomePanel została wykonana niemal w całości z założonym projektem. Posiada wiele rozmaitych ekranów, jednak każdy z nich jest obecnie modny, funkcjonalny i powszechny na wielu platformach. W pracy ukazano szereg wyzwań i problemów implementacyjnych interfejsu graficznego, czyli np. podział ekranu na jedno- i wielokolumnowy w zależności od szerokości ekranu (Rooms), popularny widok z zakładkami (Devices), formularze danych występujące powszechnie w aplikacjach. Ze wszystkimi skutecznie sobie poradzono. Przede wszystkim pragnę podkreślić spełnienie założenia projektowego co do responsywności i skalowalności GUI do wielu ekranów. Pomyślnie bowiem aplikacja działa na Macbooku Pro (OS X), Nexusie 5 i Samsungu Galaxy Tab S (Android), co zostało udokumentowane na niniejszej pracy magisterskiej.

Prezentowane przeze mnie fragmenty kodu nie stanowią całości kodu aplikacji, ale jego sporą i najbardziej kluczową część. To wszystko przy pomocy QML, który jest potężnym a z drugiej strony prostym językiem. Moim zdaniem jego *deklaratywny* charakter oszczędza mnóstwo linijek kodu, które musiałbym napisać w przypadku korzystania z języka *imperatywnego*. Wszystkie zmiany w modelach danych musiałbyły być wówczas obsługiwane w tzw. callbackach, a byłoby ich wiele z uwagi na liczbę różnych endpointów serwera REST-owego. To zwiększa wysiłek programistyczny i generuje możliwe błędy. Co więcej zmiany w kodzie byłyby trudniejsze, ponieważ trzeba by je obsługiwać w większej liczbie miejsc. Qt natomiast każdą zmianę zmiennej QML propaguje w postaci *sygnalów* i dzięki temu widoki ekranu i dane same się odświeżają.

W szeregu jego zalet wpisuje się przyjazna składnia. QML wywodzi się z JavaScripta, którego składnia jest dosyć prosta i powszechnie znana; co więcej może z nim bardzo dobrze współpracować, co również zostało ukazane w projekcie.

Do wad Qt w przypadku tej aplikacji można zaliczyć stosunkowo długi czas rozruchu aplikacji na systemie Android sięgający 2-3 sekund. Ponadto, czasem na tablecie Samsung Galaxy Tab S spotykałem się z zawieszeniem działania aplikacji i konieczne było awaryjne zamknięcie. Jednak może to wynikać z wysokiej fragmentacji Androida, jest bowiem mnóstwo sprzętów różnymi odmianami tej platformy, stąd nie należy tego traktować jako duży błąd ze strony Qt, ponieważ bardzo ciężko zapewnić wsparcie dla tak szerokiej liczby urządzeń.

Za negatywny aspekt framework'u uważam również porażkę przy implementacji odtwarzania obrazu video na żywo. Niewątpliwie powinno być to jak najszybciej rozwiązane przez twórców Qt.

Podsumowując, pisanie aplikacji projektowanej na wiele platform jest przyjemnością. Bardzo szybko możemy uzyskać pożądany efekt. Na pewno może służyć do prototypowania aplikacji. Ponadto, Qt zrealizuje logikę większości aplikacji na rynku, czyli takich w których poprzez ładny i animowany interfejs użytkownika zaprezentujemy listy danych, treści czy formularze do zbierania danych i prześlemy je przez internet. Na obecnym etapie nie zdecydowałbym się go jednak używać przy bardziej wymagających projektach, ponieważ może się okazać, że nie wspiera wszystkich wymaganych funkcjonalności, jak choćby w przypadku tej pracy - streamingu video. Ogólnie rzecz ujmując Qt 5.5 jest rzecznym, solidnym i przyjazdym

narzędziem deweloperskim, dzięki któremu można zaoszczędzić czas programistów i koszty tworzenia aplikacji na różne platformy. Na pewno należy do czołówki frameworków kross kompilujących.

5.2 Wkład autora

Framework do kross kompilacji Qt jest stosunkowo młody. Książek na ten temat jest bardzo mało, a całą wiedzę na ten temat czerpie się raczej ze sporadycznych artykułów w internecie oraz oficjalnej dokumentacji komponentów Qt. Jednak w porównaniu do konkurencyjnych narzędzi programistycznych czy języków programowania, nie istnieje jeszcze dostatecznie dużo zasobów, z których można by się uczyć. Brakuje również opublikowanych przykładów projektów czy konkretnych rozwiązań problemów programistycznych.

Zawarte implementacje są jedynie wybranymi - według mnie najlepszymi - rozwiązaniami postawionymi przez mnie zagadnień. Tworzenie aplikacji opierało się w dużej mierze na metodzie prób i błędów, podczas których okazywało się, że obrane podejście jest niedoskonałe, zbyt skomplikowane, nieczytelne lub wręcz niemożliwe do zrealizowania. Zdobyte podczas tych prób wiedza i doświadczenie nie tylko pozwoliły mi osiągnąć cele projektu, ale także utworzyć precyzyjne i przejrzyste rozwiązania.

Tworzenie interfejsu użytkownika na wiele platform jest problemem nietrywialnym ze względu na swoją złożoność. W konstrukcji takiego interfejsu raczej nie mówi się o algorytmach, jakie należy wdrożyć, ale za to koniecznie jest obranie trafnej strategii, która jak najlepiej pomoże zrealizować założony projekt. Przeliczenia i implementacje użyte w projekcie są nowatorskie oraz przyczyniają się zatem do poszerzenia bazy wiedzy na temat Qt.

6 Bibliografia

- [1] "http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/" [online]. [Data uzyskania dostępu: wrzesień 2015]
- [2] "http://doc.qt.io/QtSupportedPlatforms/index.html" [online]. [Data uzyskania dostępu: sierpień 2015]
- [3] Jesse J. Garrett, "The Elements of User Experience: User-Centered Design for the Web and Beyond", Berkeley New Riders, 2010

- [4] "<http://simpleprogrammer.com/2013/07/01/cross-platform-mobile-development/>" [online].
[Data uzyskania dostępu: wrzesień 2015]
- [5] Symeon Huang, Qt 5 Blueprints, 2015, Birmingham Packt Publishing, 2015
- [6] Johan Thelin, "Qt5 Cadaques", 2015. Adres "<http://qmlbook.github.io/>"
- [7] "<http://qmlbook.github.io/en/ch10/index.html>", [online]. [Data uzyskania dostępu: wrzesień 2015]
- [8] Wojciech Bednarski, "Learning JavaScriptMVC", Birmingham Packt Publishing, 2013
- [9] "<https://docs.particle.io/>" [online]. [Data uzyskania dostępu: wrzesień 2015]
- [10] Ilya Grigorik, "High Performance Browser Networking", chapter 15, O'Reilly Media, 2013
- [11] "The Gang of Four", "Design patterns, software engeneering, object-oriented programming", Addison-Weseley 1994
- [12] "<http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>" [online]. [Data uzyskania dostępu: wrzesień 2015]
- [13] Evan M. Hahn, "Express in Action", MEAP, 2014
- [14] "<http://openmymind.net/mongodb.pdf>" [online], [Data uzyskania dostępu: wrzesień 2015]