

Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I  
I N A U K I N F O R M A C Y J N Y C H



# Praca dyplomowa inżynierska

na kierunku Informatyka i Systemy Informacyjne

System Smart Home oparty na systemie wbudowanym

**Adrian Cieśla**

Numer albumu 313207

**Michał Czapnik**

Numer albumu 313209

**Jan Palmer**

Numer albumu 313429

promotor

dr inż. Janusz Rafałko

WARSZAWA 2024



## Streszczenie

### System Smart Home oparty na systemie wbudowanym

Rozwiązania w obszarze inteligentnych domów w ostatnich latach nabierają popularności, stając się coraz bardziej niezawodnymi, tańszymi oraz łatwiejszymi w obsłudze. Kilka kliknięć na urządzeniu mobilnym wystarcza do zapalenia światła, otworzenia garażu, czy też uzyskania informacji o warunkach pogodowych przez podłączone czujniki. Niniejsza praca odkrywa możliwości implementacji zdalnie sterowanych urządzeń tego typu na systemie wbudowanym. Raspberry Pi 4B został użyty jako zdalny hub, do którego podłączone zostały czujniki, diody LED, czujniki ruchu oraz małe rotory do obsługi blokowania drzwi. Ponadto, utworzona została aplikacja mobilna, aby zapewniać użytkownikom dostęp do danych, czy też do sterowania całym systemem inteligentnego domu. Oba powyższe moduły zostały połączone przez serwer w chmurze, aby umożliwić sprawdzanie stanów urządzeń bez względu na region świata, w którym użytkownik się aktualnie znajduje. Całość została przygotowana jako forma demonstracyjna produktu wraz z makietą domu wyposażoną w zdalnie kontrolowane urządzenia.

**Słowa kluczowe** — Smart Home, Raspberry Pi, serwer w chmurze, tworzenie aplikacji mobilnych, makietą domu

## Abstract

Smart Home system based on an embedded system

Smart Home solutions are rising in popularity in recent years, as they are being made more reliable, cheaper and easier to use. A few clicks on a phone can turn on lights, open a garage door, or display weather conditions through connected sensors. This paper explores the possibilities of implementing such remote-controlled devices on an embedded system. Raspberry Pi 4B has been used as a remote hub, to which sensors, LED diodes, motion sensors and small motors, to use as door locks, have been connected. A mobile application has been made to provide data and control inputs to users. Both have been connected by a cloud server to allow checking of device statuses from all over the world. All of this has been prepared as a proof of concept for a product, together with a demonstrative model of a house with remotely controlled appliances.

**Keywords** — Smart Home, Raspberry Pi, cloud backend server, mobile development, house model



## Spis treści

<b>1. Wstęp</b>	<b>5</b>
1.1. Nazewnictwo	6
<b>2. Projekt systemu</b>	<b>7</b>
2.1. Wymagania funkcjonalne	7
2.2. Wymagania нефункционалне	10
2.3. Analiza ryzyka	12
2.4. Perspektywa biznesowa	13
2.5. Podział zadań	14
<b>3. Opis rozwiązania</b>	<b>16</b>
3.1. System Wbudowany	18
3.1.1. Struktura systemu wbudowanego	18
3.1.2. Opis użytych urządzeń	21
3.1.3. Przykład podłączenia urządzeń	22
3.1.4. Biblioteki	26
3.2. Serwer w chmurze	27
3.2.1. Opis ogólny	27
3.2.2. Wybór technologii	27
3.2.3. Struktura projektu serwera	28
3.2.4. API serwera	31
3.2.5. Przeprowadzone testy jednostkowe	31
3.2.6. Przeprowadzone testy integracyjne	31
3.2.7. Przeprowadzone testy akceptacyjne	32
3.3. Aplikacja mobilna	34
3.3.1. Opis ogólny	34
3.3.2. Wybór technologii	34
3.3.3. Architektura aplikacji	34
3.3.4. Struktura projektu aplikacji	35

3.3.5.	Prototyp interfejsu użytkownika . . . . .	36
3.3.6.	Przebieg implementacji . . . . .	40
3.3.7.	Testy akceptacyjne . . . . .	43
3.4.	Komunikacja pomiędzy modułami . . . . .	44
3.4.1.	Protokół komunikacji . . . . .	48
3.4.2.	Konfiguracja sieci . . . . .	48
3.4.3.	Specyfikacja API . . . . .	49
<b>4.</b>	<b>Makieta domu . . . . .</b>	<b>50</b>
4.1.	Opis makiety . . . . .	50
4.2.	Układ elektroniczny wewnątrz makiety domu . . . . .	52
<b>5.</b>	<b>Możliwości dalszego rozwoju projektu . . . . .</b>	<b>57</b>
5.0.1.	Wsparcie dla działania wielu układów Raspberry Pi jednocześnie . . . . .	57
5.0.2.	Aktualizowanie stanów urządzeń na bieżąco . . . . .	58
5.0.3.	Powiadomienia o wykryciu ruchu w aplikacji mobilnej . . . . .	58
5.0.4.	Zmiana nazw urządzeń . . . . .	59
5.0.5.	Niestandardowe grupowanie urządzeń . . . . .	59
5.0.6.	Rozpoznawanie głosu . . . . .	60
5.0.7.	Łączenie działania urządzeń . . . . .	60
5.0.8.	Cyberbezpieczeństwo . . . . .	60
<b>6.</b>	<b>Podsumowanie i wnioski . . . . .</b>	<b>61</b>
6.1.	Trudności stojące za wytwarzaniem systemów inteligentnego domu . . . . .	61
6.2.	Doświadczenia z projektem . . . . .	62
6.2.1.	System wbudowany . . . . .	62
6.2.2.	Serwer w chmurze . . . . .	62

## 1. Wstęp

Na początku XX wieku wizja inteligentnego domu, w którym codzienne czynności zamiast człowieka wykonują maszyny, wydawała się być odległą od rzeczywistości. Z biegiem lat jednak technologia ulegała coraz szybszemu postępowi. Rozpoczęto konstrukcję urządzeń takich jak lodówki, telewizory, czy komputery, znacznie zwiększając komfort przebywania w swoim miejscu zamieszkania. Z czasem nowe rozwiązania w tym obszarze zaczęły koncentrować się na świecie cyfrowym i umożliwiły wykonywanie wielu codziennych czynności, niegdyś uciążliwych, z wykorzystaniem jedynie kliknięć, czy też komend głosowych. Perspektywa dalszego rozwoju w obszarze inteligentnych domów jest ogromna. Arsenal narzędzi technologicznych, które można wykorzystać w tej dziedzinie wciąż bardzo szybko się rozwija i daje wiele możliwości do wytwarzania nowych, innowacyjnych rozwiązań. Atrakcyjnym więc może się wydawać próba wejścia na rynek systemów inteligentnych domów ze swoim własnym, konkurencyjnym systemem.

Obecnie istnieje niewiele popularnych i dobrze rozwiniętych systemów Smart Home:

- Google Home [1]
- Home Assistant [2]
- openHAB [3]

Każdy z nich ma swoje wady. W przypadku Google Home użytkownicy często skarżą się na nieprzewidziane błędy systemu, czy też powolne działanie. W przypadku Home Assistant konfiguracja całego systemu jest zbyt skomplikowana dla osób nieposiadających wiedzy w zakresie elektroniki i informatyki. Podobnie jest też w przypadku openHAB. Mając świadomość faktu, jak duży potencjał ma rozwój technologii Smart Home i tego, jak słabo rozwinięty obecnie jest to rynek, nasz zespół postanowił zrozumieć, w jaki sposób można zaradzić powyższym problemom próbując stworzyć od podstaw swój własny system inteligentnego domu.

Wizja naszego systemu opiera się o zapewnienie użytkownikowi możliwie największego komfortu. Chcieliśmy, żeby sposób korzystania z systemu był prosty i wygodny, a efekty jego działania satysfakcjonujące. Układ elektroniczny po zamontowaniu w budynku może być kontrolowany z wykorzystaniem aplikacji mobilnej. Użytkownik takiej aplikacji ma możliwość sterowania świa-



tłem, systemem alarmowym, zamkiem od drzwi wejściowych, a także analizy danych z czujników temperatury, natężenia światła słonecznego, czy wilgotności.

### **1.1. Nazewnictwo**

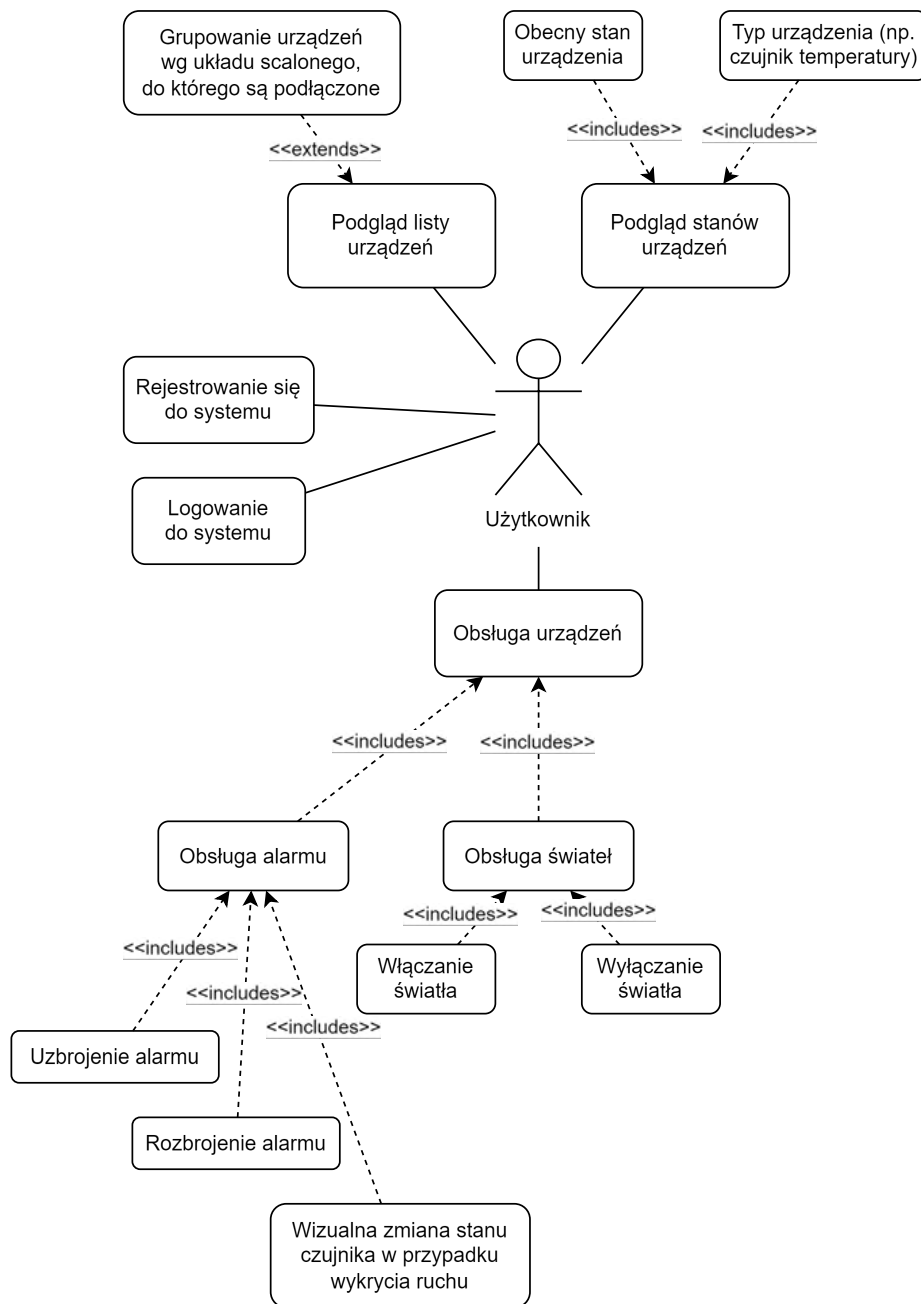
- Urządzenie, urządzenia - różnego rodzaju sensory, światła czy silniki sterujące zamkami do drzwi, podłączone do urządzeń Raspberry Pi
- Układ Raspberry Pi, układ RPi, płytka, układ scalony - platforma komputerowa Raspberry Pi 4B
- Chmura, serwer zewnętrzny, serwer w chmurze - serwer stanowiący jeden z głównych modułów systemu, pośredniczący w komunikacji pomiędzy Raspberry Pi 4B, a aplikacją mobilną

## 2. Projekt systemu

Przed rozpoczęciem prac konieczne było zaprojektowanie struktury całego systemu, a także jego aktywności i zachowań w określonych sytuacjach, tak aby wszyscy członkowie zespołu posiadali wspólną i zgodną wizję dotyczącą jego finalnego efektu. W niniejszym rozdziale przedstawiony został projekt konstruowanego systemu. Omówione zostały pojęcia takie jak: wymagania funkcjonalne i нефункционалне, analiza ryzyka, perspektywa biznesowa, czy też podział zadań pomiędzy członków zespołu. Na ich podstawie bazowała całość pracy wykonywanej nad budową systemu inteligentnego domu.

### 2.1. Wymagania funkcjonalne

Poniżej na rysunku 2.1 oraz w tabeli 2.1 przedstawione zostały sposoby na interakcję między użytkownikiem, a systemem. W skład tych interakcji wchodzi m.in. obsługa urządzeń elektronicznych zamontowanych w budynku, czy też rejestrowanie i logowanie się do systemu. Do obsługiwanych przez użytkownika urządzeń można zaliczyć system alarmowy, światła, czujniki oraz zamki do drzwi.



Rysunek 2.1: Przypadki użycia pokazujące możliwości interakcji użytkownika z systemem

## 2.1. WYMAGANIA FUNKCJONALNE

Tabela 2.1: Opis przypadków użycia dla użytkownika i systemu

Nazwa	Opis	Odpowiedź systemu
Podgląd listy urządzeń	Użytkownik ma dostęp do wyświetlania listy urządzeń z dodatkowymi opcjami grupowania.	Widok w aplikacji przedstawiający listę wszystkich dostępnych urządzeń.
Podgląd stanów urządzeń	W systemie wykorzystywane będą różnorodne urządzenia elektroniczne, dlatego cenną informacją dla użytkownika mogą być dokładne parametry każdego z nich, takie jak nazwa oraz typ urządzenia (np. czujnik temperatury, światło, itp.).	Widok w aplikacji przedstawiający nazwę, typ urządzenia oraz dodatkowe informacje zależne od typu, np. odczyt temperatury.
Obsługa urządzeń	Użytkownikowi zależy na tym, by z wykorzystaniem aplikacji mieć kontrolę nad urządzeniami, do których ma dostęp.	Reakcja na żądania użytkownika w postaci np. włączenia światła w jednym z pokoi czy uzbrojenia systemu alarmowego.
Rejestrowanie się do systemu	Każdy użytkownik systemu musi posiadać w nim utworzone dla siebie konto. Takie podejście ułatwi zapobiegnięciu nieautoryzowanym dostępom do urządzeń w budynku użytkownika	Utworzenie konta, z pomocą którego użytkownik jest w stanie się zalogować do systemu.
Logowanie się do systemu	Do korzystania z systemu wymagane jest od użytkownika zalogowanie się przy pomocy danych, które podał w trakcie rejestracji. Rozwiązanie to ma na celu zwiększenie bezpieczeństwa systemu, uniemożliwiając zdalny dostęp do płytki niepowołanym użytkownikom.	Weryfikacja istnienia konta, a następnie otwarcie dostępu do aplikacji.

## 2.2. Wymagania niefunkcjonalne

W tabeli 2.2 przedstawione zostały wymagania niefunkcjonalne systemu. Ważnym jest przede wszystkim, aby interfejs użytkownika był przejrzysty, natomiast system był możliwie jak najczęściej dostępny oraz jak najbardziej wydajny.

Tabela 2.2: Wymagania niefunkcjonalne

Obszar wymagań	Nr wymaga- nia	Opis
Użyteczność	1	Wszystkie funkcjonalności aplikacji dostępne dla użytkownika muszą mieścić się na ekranie dowolnego smartfona, natomiast czcionka tekstu obecnego w aplikacji nie powinna być mniejsza niż 12 pt.
Niezawodność	2	System powinien być dostępny przez co najmniej 99% czasu każdej doby
	3	System musi być odporny na usterki urządzeń do niego podłączonych i reagować na nie w sposób minimalizujący zarówno ich zauważenie przez użytkownika, jak i zagrożenia z nich wynikające

## 2.2. WYMAGANIA NIEFUNKCJONALNE

Wydajność	4	System musi realizować żądania użytkownika w czasie nie dłuższym, niż 3 sekundy od momentu ich złożenia oraz być w stanie podobnie obsługiwać do 100 żądań na minutę
Wsparcie	5	W przypadku fizycznej usterki układu elektronicznego wchodzącego w skład systemu, konieczne będzie poinformowanie o zdarzeniu administratora, który następnie pomoże w przywróceniu funkcjonalności układu

### 2.3. Analiza ryzyka

W tabeli 2.3 znajduje się analiza SWOT całego projektu. Przedstawione zostały zarówno ryzyka, jak i szanse stojące za jego realizacją.

Tabela 2.3: Analiza SWOT projektu - Zagrożenia i szanse wewnętrzne

SWOT	Wewnętrzne
Zagrożenia	<ol style="list-style-type: none"> <li>1. Bardzo prawdopodobne (Zagrożenie umiarkowane). Ograniczone możliwości obliczeniowe i pamięciowe mogą stwarzać problemy Rozwiązanie: przesył danych do zewnętrznej bazy danych w celu oszczędzania użytku pamięci.</li> <li>2. Pewne (Zagrożenie umiarkowane) Brak wiedzy programistycznej członków zespołu w związku z niektórymi obszarami. Rozwiązanie: Odpowiednie oddelegowanie zadań współpraca.</li> </ol>
Szanse	<ol style="list-style-type: none"> <li>1. Ekonomiczne rozwiązanie wykorzystujące Raspberry Pi 4B.</li> <li>2. Wysoka personalizacja i elastyczność integracji czujników.</li> <li>3. Możliwości nauki i rozwoju umiejętności w zakresie hardware'u i automatyzacji.</li> </ol>

Tabela 2.4: Analiza SWOT projektu - Zagrożenia i szanse zewnętrzne

SWOT	Zewnętrzne
Zagrożenia	<p>1. Prawdopodobieństwo średnie (Zagrożenie wysokie). Naruszenia bezpieczeństwa danych lub próby włamań do systemu Smart Home. Rozwiązanie: Zaimplementowanie systemów bezpieczeństwa m.in. autoryzacji użytkowników (na potrzebę projektu inżynierskiego implementacja systemów zabezpieczeń zostanie pominięta).</p> <p>2. Prawdopodobieństwo wysokie (Zagrożenie średnie). Problemy spowodowane brakiem/złymi wersjami sterowników do wykorzystywanych urządzeń. Rozwiązanie: Zaplanowanie na etap programowania dodatkowego czasu, aby rozwiązać nieprzewidziane trudności związane ze sterownikami.</p>
Szanse	<p>1. Rosnące zainteresowanie technologią Smart Home stwarza szansę na przekształcenie projektu w produkt komercyjny.</p> <p>2. Dofinansowania za oszczędzanie prądu - projekt można rozwinąć by spełnił wymagania do otrzymania dotacji.</p> <p>3. Współpraca z organizacjami badawczymi, uniwersytetami lub firmami poszukującymi wglądu w trendy i wzorce z pomocą zbieranych danych.</p>

## 2.4. Perspektywa biznesowa

Dalszy rozwój niniejszego systemu może przyczynić się do utworzenia konkurencyjnej alternatywy dla popularnych obecnie systemów inteligentnych domów. Zainteresowanie rozwiązaniami



tego typu wciąż rośnie i nowe spojrzenie na wytwarzanie podobnych systemów może przyczynić się do szybszego rozwoju tego rynku oraz zwiększenia świadomości na jego temat wśród osób niepowiązanych ze światem technologii. Zespół opracowywujący system z niniejszej pracy wierzy, że nie istnieje obszar technologiczny, w którym nie ma już miejsca na rozwój. Innowacje technologiczne są tym, co najbardziej przyciąga konsumentów i rozwija świat, dlatego warto próbować wdrażać swoje własne rozwiązania i rzucać wyzwanie obecnym liderom rynku.

W skład głównych elementów projektu, które mogą wzbudzić zainteresowanie rynku wchodzi:

- Nieszablonowe podejście do montażu urządzeń w budynku
- Wynajem sprawdzonych i przeszkolonych specjalistów od montażu
- Spójność działania całego systemu i wszystkich tworzących go urządzeń

## 2.5. Podział zadań

Aby postępy w pracy zespołu dokonywane były w sposób efektywny, dokonano podziału zadań pomiędzy jego członków zgodnie z poniższą tabelą 2.5:

Tabela 2.5: Podział zadań

Imię i nazwisko	Zadania
Adrian Cieśla	Przygotowanie makiety domu
	Montaż elektroniki w makiecie domu
	Dokumentacja wymagań i struktury systemu
	Dokumentacja diagramów komunikacji między modułami
	Integracja systemów
	Techniczne utworzenie bazy danych
	Stworzenie API serwera zewnętrznego
	Wdrożenie serwera zewnętrznego i bazy danych do chmury
	Przygotowanie testów serwera zewnętrznego

## 2.5. PODZIAŁ ZADAŃ

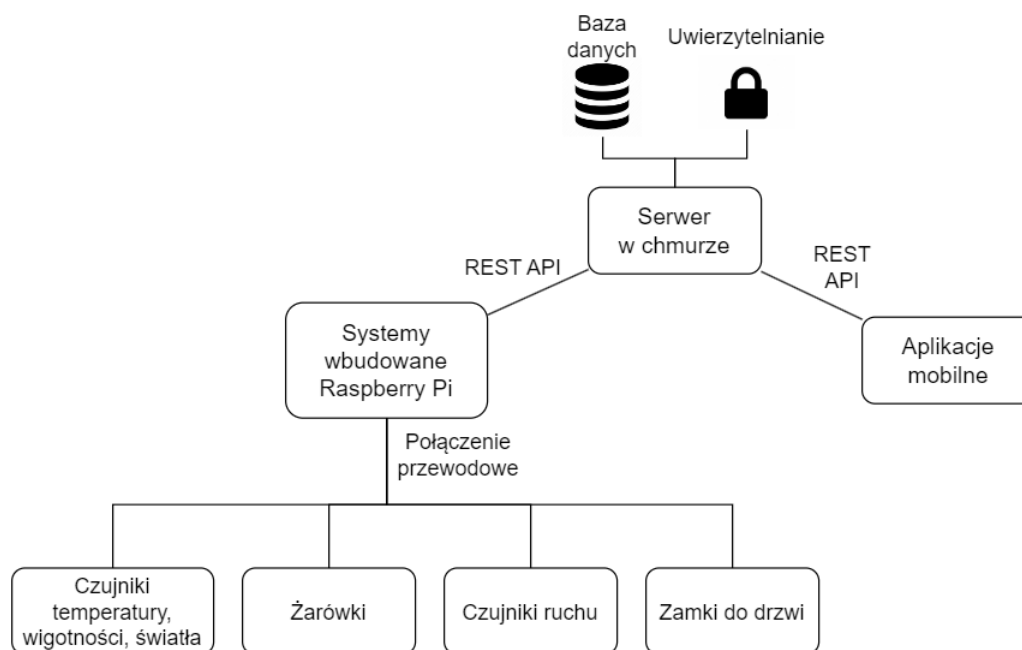
Michał Czapnik	Przygotowanie przykładowych połączeń urządzeń do RPI
	Stworzenie programu obsługującego urządzenia fizyczne na RPI
	Odpowiedzialność za komunikację API do płytki RPI
	Integracja systemów
Jan Palmer	Przygotowanie prototypu interfejsu użytkownika
	Wytworzenie aplikacji mobilnej wraz z testami
	Przygotowanie pliku .apk
	Instrukcja użytkowania aplikacji
	Pomoc przy implementacji autoryzacji na serwerze
	Integracja systemów

### 3. Opis rozwiązania

System podzielony jest na 3 główne komponenty:

- System wbudowany
- Serwer zewnętrzny
- Aplikację mobilną

Aby wszystkie funkcjonalności systemu działały zgodnie z jego pierwotną wizją, wspomniane wyżej komponenty muszą ściśle ze sobą współpracować. Forma tej współpracy została dokładniej zilustrowana na poniższym rysunku nr 3.1:



Rysunek 3.1: Schemat architektury systemu

Zgodnie z ideą, aby rozwiązanie skupiało się na zapewnieniu użytkownikowi jak największego komfortu, zdecydowano, aby komunikacja pomiędzy modułami odbywała się za pomocą łącza internetowego oraz korzystała z protokołu HTTP i rozwiązania architektonicznego REST API. Taka decyzja umożliwia sterowanie całym systemem bez względu na odległość użytkownika od budynku, w którym zamontowany został układ elektroniczny, co może być przydatne

w sytuacji, gdy zapomnieliśmy wyłączyć jedno z energochłonnych urządzeń w naszym domu. Serwer w chmurze tworzy „most” pomiędzy użytkownikiem, a układami elektronicznymi sterującymi urządzeniami systemu. Odgrywa ważną rolę przy pozyskiwaniu danych z bazy danych oraz uwierzytelnianiu użytkowników dla celów cyberbezpieczeństwa. Aplikacja mobilna odpowiada za pozyskiwanie danych wejściowych od użytkownika, natomiast układy Raspberry Pi sprawiają, by jego żądania wywoływały oczekiwany efekt w rzeczywistości.

Nie są to jednak jedyne zadania modułów tworzących nasz system. W skrócie przedstawione zostały one w poniższej tabeli 3.1, a dokładniej wyjaśnione zostaną w kolejnym podrozdziale.

Tabela 3.1: Zadania modułów systemu inteligentnego domu

Moduł	Zadania
System wbudowany	Przekazywanie sygnałów do urządzeń elektronicznych zgodnie z żądaniami serwera zewnętrznego
	Odbiór pomiarów z czujników oraz ich wysył do serwera zewnętrznego
Serwer zewnętrzny	Przetwarzanie danych z systemu wbudowanego i umieszczanie ich w bazie danych
	Przesyłanie danych z bazy danych do aplikacji mobilnej na żądanie użytkownika
	Autentykacja użytkowników
	Weryfikacja poprawności żądań do układów Raspberry Pi
	Umożliwienie komunikacji z systemem inteligentnego domu na odległość
	Pośredniczenie między użytkownikiem, a skomplikowanym układem elektronicznym
Aplikacja mobilna	Proste zwizualizowanie użytkownikowi możliwości całego systemu
	Komunikowanie akcji użytkownika do serwera zewnętrznego
	Bezpieczny dostęp do zasobów systemu

### 3.1. System Wbudowany

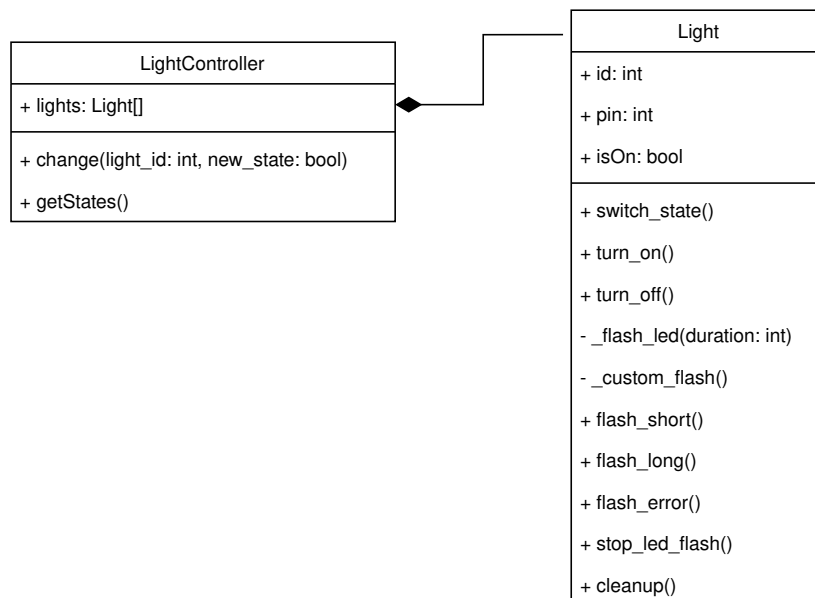
System wbudowany, zrealizowany na płycie Raspberry Pi (RPi), stanowi podstawową część projektu, będąc jego jednostką kontrolującą urządzenia. Jego głównym zadaniem jest obsługa różnorodnych urządzeń fizycznych, takich jak czujniki czy światła LED. Dodatkowo, system ten jest odpowiedzialny za lokalne przetwarzanie danych, w tym analizę sygnałów z czujników oraz przygotowywanie danych do przekazania na zewnętrzny serwer.

Na potrzebę realizacji systemu, na płycie Raspberry Pi zainstalowany został system operacyjny Raspberry Pi OS. Program obsługujący podłączone urządzenia został napisany przy pomocy języka Python3. Wybór tego języka jest uzasadniony faktem, że umożliwia on szybkie i proste pisanie kodu mikrokontrolera, obsługującego urządzenia fizyczne. Python3 oferuje również liczne biblioteki i moduły, które ułatwiają obsługę różnych urządzeń fizycznych, takich jak czujniki lub interfejsy komunikacyjne.

#### 3.1.1. Struktura systemu wbudowanego

Wedle modularnego paradygmatu programowania, kod został podzielony na moduły, które zostaną przybliżone w tej części pracy.

#### Moduł światła



Rysunek 3.2: Diagram klas kontrolera światła.

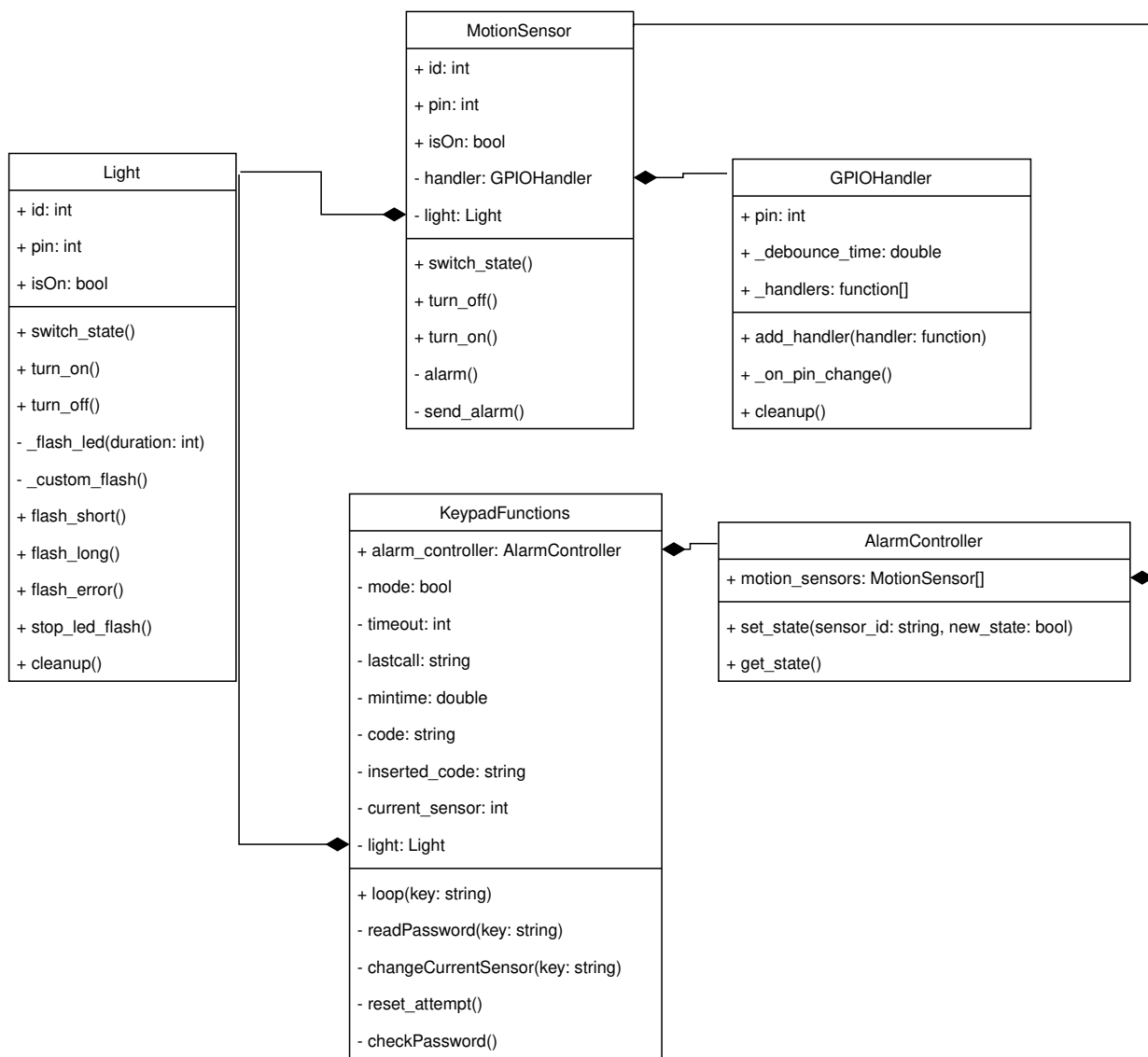
Powyższy rysunek przedstawia diagram klas modułu światła. Moduł ten przechowuje wszystkie dostępne światła, oraz udostępnia 2 metody. `change(...)` przyjmuje id światła oraz stan

### 3.1. SYSTEM WBUDOWANY

na jaki ma zostać ustawione; `getStates()` zwraca plik json zawierający wszystkie stany świateł.

Prostota tego modułu spowodowana jest tym, że w tej fazie projektu nie jest przewidywane fizyczne sterowanie światłami, a jedynie za pomocą aplikacji.

#### Moduł alarmu



Rysunek 3.3: Diagram klas kontrolera alarmu.

Na rysunku widoczny jest podział na klasy modułu alarmu. W module alarmowym zaimplementowano czujniki ruchu w dowolnej liczbie oraz klawiaturę fizyczną, której obsługa oparta jest na bibliotece `pad4pi`. Zarówno czujniki ruchu, jak i klawiatura posiadają dodatkowe diody LED, poprawiające doświadczenie użytkownika (UX) poprzez wizualizację stanu.

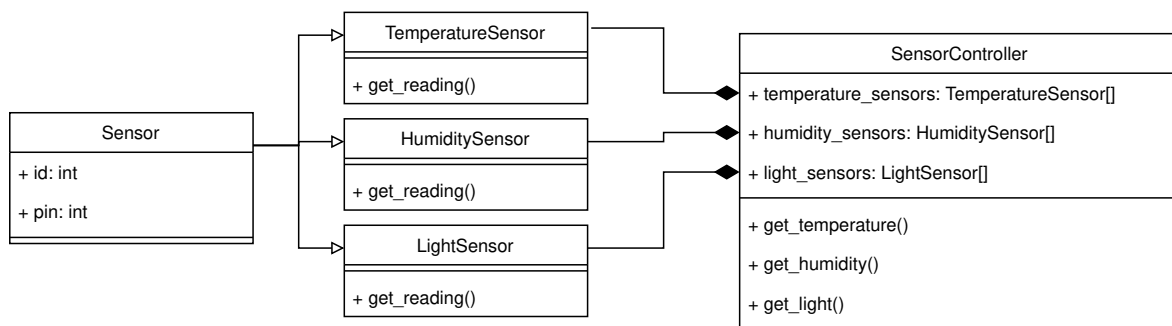
Czujniki ruchu są skonfigurowane do działania na zasadzie wykrywania ruchu, a każdy z nich jest powiązany z obiektem `Light`. Przy ich inicjalizacji tworzony jest również `Handler`, który

odpowiada za obsługę alarmu. W momencie detekcji ruchu, handler uruchamia metodę `alarm()` z klasy `MotionSensor`, co powoduje wysłanie informacji na zewnętrzny serwer, jeśli dany sensor jest aktywowany.

Klawiatura fizyczna umożliwia interakcję użytkownika poprzez wprowadzanie hasła. Po naciśnięciu klawisza, uruchamiana jest metoda `loop(key)` z klasy `KeypadFunctions`, gdzie „key” to napis zawierający pojedynczy znak reprezentujący przycisk. Klawiatura pozwala na aktywację i dezaktywację dowolnego czujnika ruchu poprzez podanie jego identyfikatora oraz wpisanie hasła.

Dodatkowo, klawiatura dostarcza informacji zwrotnej poprzez migotanie diody w odpowiedni sposób, na przykład po wpisaniu pojedynczego znaku, po poprawnym wpisaniu hasła oraz po błędnym wpisaniu hasła. To zapewnia użytkownikowi informacje zwrotne na temat aktualnego stanu systemu.

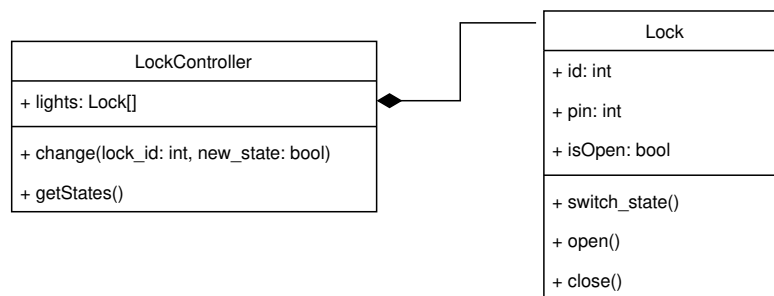
## Moduł Sensorów



Rysunek 3.4: Diagram klas kontrolera sensorów.

W tym module zawarta jest cała logika odczytów z sensorów wbudowanych w system. Jak wi-  
dać na powyższym rysunku klasy poszczególnych sensorów są do siebie podobne na tyle, aby mo-  
gły dziedziczyć po klasie `Sensor`. Różnią się jedynie przetwarzaniem wartości napięcia odczytanej  
z pina GPIO. Na przykładzie sensorów temperatury po wywołaniu metody `get_temperature()` z  
kontrolera zostanie pobrany wynik ze wszystkich czujników temperatury a następnie zwrócony  
zostanie json z danymi, takimi jak wartość temperatur, id czujnika i data pobrania pomiaru.  
System Raspberry Pi ograniczony jest brakiem analogowych pinów GPIO. Żeby możliwe było  
zdczytywanie danych z sensorów użyty został przetwornik sygnału analogowego na cyfrowy.

#### Moduł Zamka



Rysunek 3.5: Diagram klas kontrolera zamków.

Diagram klas i sposób działania modułu wygląda podobnie do modułu światła. Kontroler zamków udostępnia 2 metody: do odczytania stanów zamków oraz do zmiany stanu odpowiedniego zamka.

#### 3.1.2. Opis użytych urządzeń

##### Diody LED

W celu symulacji sterowania światłem wykorzystane zostały diody LED. Powinny one być zasilane napięciem 2-3 voltów. Minimalne napięcie wyjściowe z RPI wynosi 3,3 volta, dlatego w połączeniu występują również odpowiednie oporniki.

##### Czujnik temperatury - termistor NTC-MF52 z potencjometrem

Zastosowany czujnik wykrywa temperaturę w zakresie od -55°C do 125°C. Zasilany napięciem 5 voltów. Maksymalny błąd pomiaru wynosi 0.5 °C.

##### Czujnik do pomiaru wilgotności gleby - Waveshare 9527

Czujnik ten jest zasilany napięciem od 3,3 do 5 voltów. Na wyjściu zwraca napięcie proporcjonalne do wilgotności gleby. Dodatkowo sensor wyposażony jest w diodę led, która świeci się gdy sensor jest podłączony.

##### Czujnik światła otoczenia - ALS-PT19

Czujnik światła otoczenia działa na zasadzie fotorezystorów. Napięcie zasilania: od 2,5 do 5,5 voltów.



### **Przetwornik A/C MCP3008 10-bitowy 8-kanalowy SPI - DIP**

Przetwornik sygnału analogowego na cyfrowy umożliwia odczyt danych z sensorów analogowych na urządzeniu RPI. Zasilany jest napięciem od 2,7 do 5,5 voltów, oraz komunikuje się po interfejsie SPI. Posiada 8 kanałów co umożliwia podłączenie 8 niezależnych sensorów naraz, a czas konwersji sygnału wynosi 10 us.

### **Czujnik ruchu PIR HC-SR501**

Czujnik ruchu zasilany jest napięciem z zakresu 4,5 do 20 voltów, posiada zasięg do 7 metrów. Czujnik do działania wykorzystuje promieniowanie podczerwone. Po wykryciu obiektu wyjście czujnika przechodzi w stan wysoki. Jest również wyposażony w 2 potencjometry, dzięki którym można sterować długością utrzymywania się stanu wysokiego na wyjściu po wykryciu obiektu, oraz zasięgiem wykrywania ruchu.

### **Klawiatura membranowa 4x4**

W momencie nacisku na przycisk klawiatury membranowej, dochodzi do sprzężenia odpowiednich przewodów i umożliwienia przepływu prądu. Proces badania który przycisk został kliknięty zachodzi po stronie software'owej.

### **Siłownik - Serwo SG-90 - micro - 180**

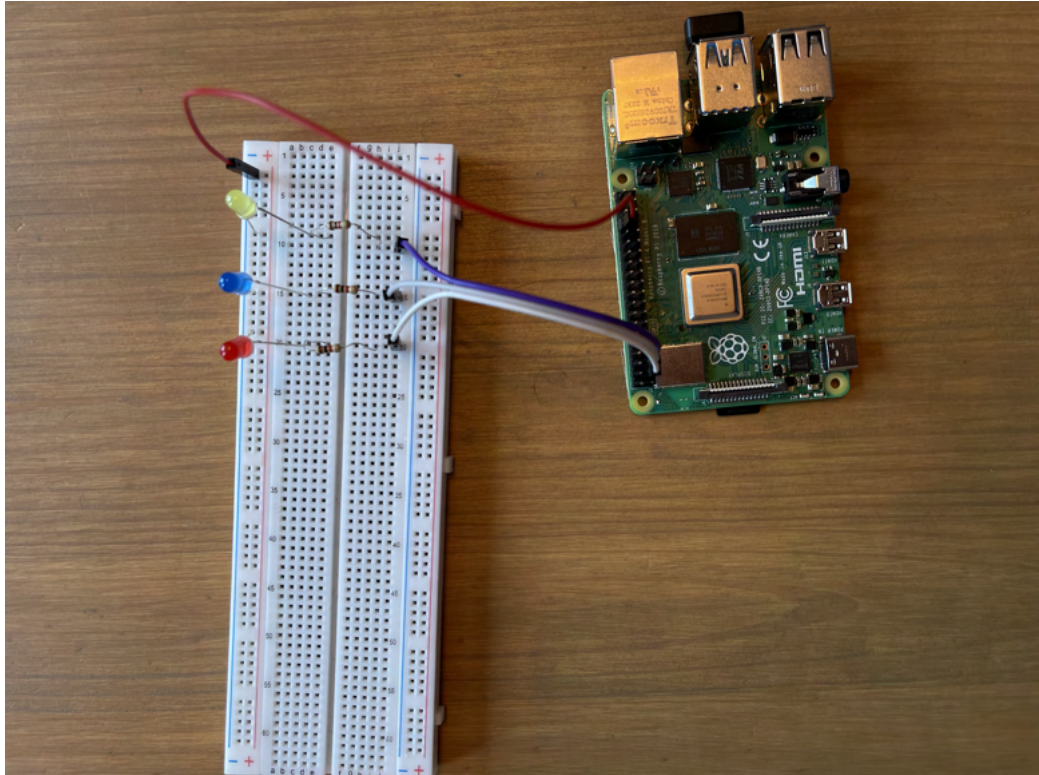
Siłownik zasilany jest napięciem 4,8 volta. Sterowanie odbywa się poprzez wysyłanie odpowiednich impulsów sygnału prostokątnego o zmiennej szerokości impulsów.

#### **3.1.3. Przykład podłączenia urządzeń**

##### **Światło**

Na poniższym zdjęciu widoczne jest przykładowe podłączenie 3-ech diod LED z urządzeniem Raspberry Pi. Do obsługi wykorzystywane są 3 piny GPIO oraz ground pin. W połączeniu wykorzystywane są również oporniki.

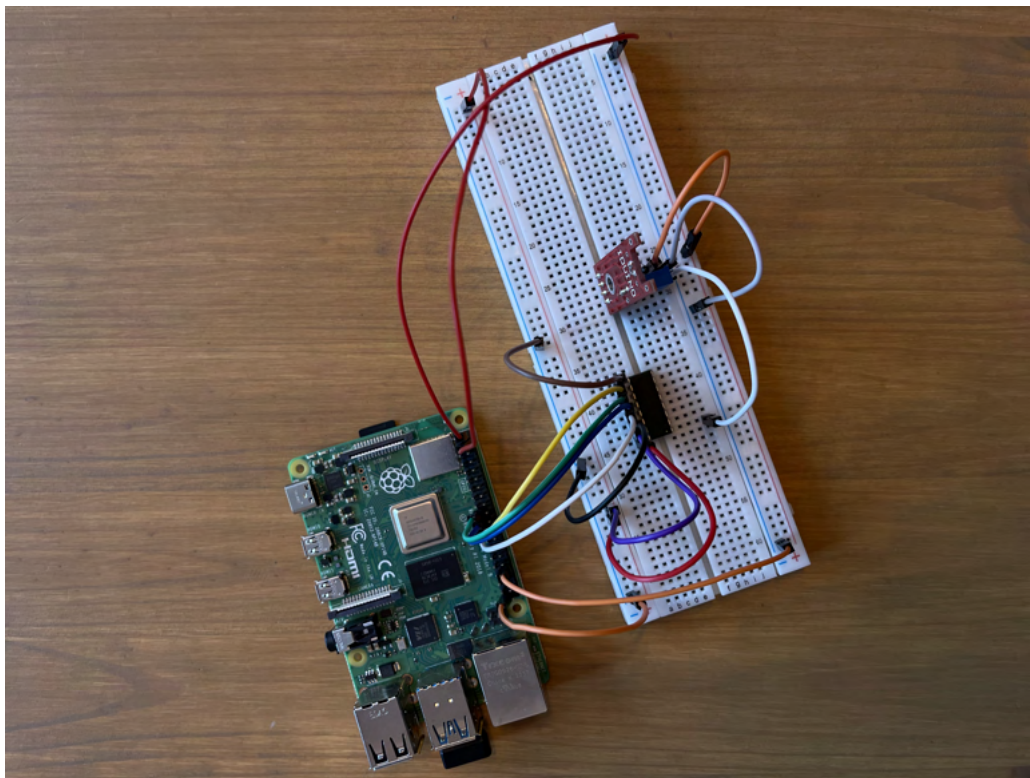
### 3.1. SYSTEM WBUDOWANY



Rysunek 3.6: Diody led połączone z RaspberryPi

### Sensory i przetwornik sygnału

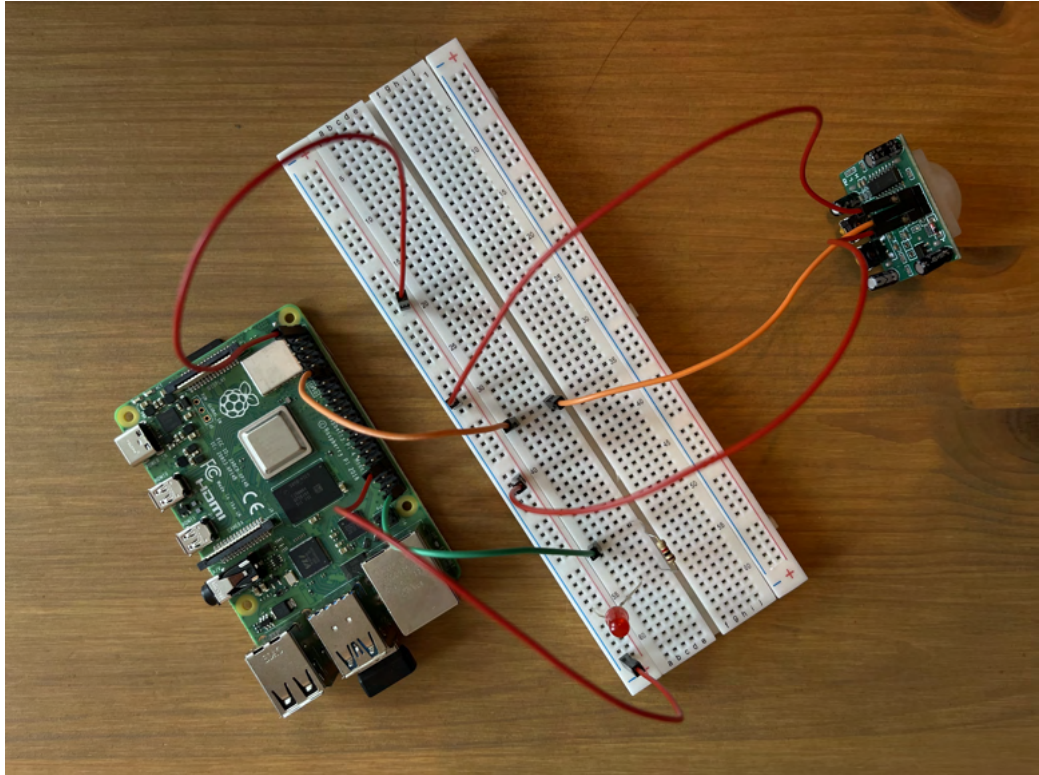
Czujnik temperatury na poniższym zdjęciu został podłączony do napięcia 5 voltów i do ground pin. Aby poprawnie odczytać wartość z czujnika sygnał został przetworzony za pomocą przetwornika MCP3008 również zasilanego napięciem 5 voltów.



Rysunek 3.7: Czujnik ruchu połączony z RaspberryPi

### Czujnik ruchu

Czujnik ruchu połączony został do napięcia 5-ciu voltów i do ground pin. Dodatkowo wykorzystywany jest jeszcze pin GPIO, w celu odczytu stanu sensora. LED pin połączony tak samo jak w poprzednim przykładzie.

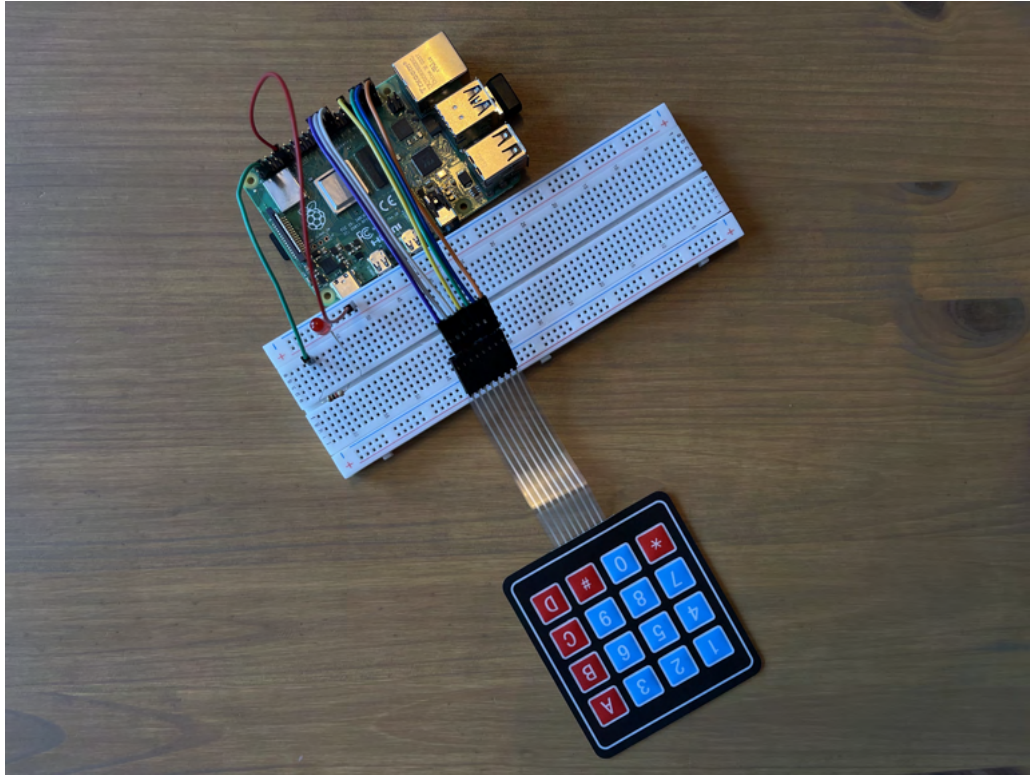


Rysunek 3.8: Czujnik ruchu połączony z RaspberryPi

## Klawiatura

Klawiatura membranowa do poprawnego działania używa tylko piny GPIO. na poniższym obrazku do podłączenia wykorzystano jedynie 7 pinów, ponieważ do funkcjonowania programu nie jest potrzebna ostatnia kolumna przycisków z klawiatury. Led pin podłączony tak samo jak w poprzednim przykładzie.





Rysunek 3.9: Klawiatura membranowa połączona z RaspberryPi

#### 3.1.4. Biblioteki

Do poprawnego działania systemu sterowniczego wykorzystywane są następujące biblioteki w języku Python3:

- Flask [4]
- RPi.GPIO [5]
- pad4pi [6]

Biblioteka Flask odpowiada za uruchomienie serwera na każdym z systemów wbudowanych, który z wykorzystaniem rozwiązania architektonicznego REST API umożliwi komunikację elektronicznej warstwy systemu z chmurą. RPi.GPIO to z kolei biblioteka umożliwiająca dostęp do wyprowadzeń GPIO na płycie Raspberry Pi. Jest on konieczny do przesyłania informacji w postaci napięcia prądu elektrycznego do poszczególnych elementów układu elektrycznego zamontowanego w budynku. pad4pi umożliwia korzystanie z klawiatury membranowej oraz odpowiednie obsługiwanie eventów spowodowanych przyciśnięciem przycisku.

### 3.2. Serwer w chmurze

Zdalne i efektywne sterowanie naszym systemem nie byłoby możliwe bez istnienia zewnętrznego modułu takiego jak serwer w chmurze. Bezpośrednia komunikacja płytki Raspberry Pi i aplikacji mobilnej byłaby możliwa, jednak w zaawansowanym systemie o wysokim potencjale do skalowalności, takim jak ten z niniejszej pracy, konieczne są również:

- zadbanie o autentykację i autoryzację użytkowników,
- możliwość nadawania nazw własnych konkretnym urządzeniom podłączonym do systemu
- kontrola uprawnień użytkowników do danych układów Raspberry Pi (na potrzebę projektu inżynierskiego implementacja systemów zabezpieczeń zostanie pominięta)
- gromadzenie i przekazywanie danych z układów Raspberry Pi dla wszystkich użytkowników posiadających do nich uprawnienia
- łatwe i szybkie aktualizowanie systemu, niewymagające interakcji ze strony jego użytkowników

Spełnienie powyższych wymagań możliwe jest tylko i wyłącznie z wykorzystaniem pojedynczego, centralnego i niezależnego modułu, z którym zarówno płytki Raspberry Pi, jak i użytkownicy mogą się bezpośrednio komunikować. Takim modulem jest właśnie serwer wystawiony w chmurze.

#### 3.2.1. Opis ogólny

Serwer w chmurze, tak jak było to wspomniane na początku niniejszej pracy inżynierskiej, pełni rolę „mostu” pomiędzy użytkownikami systemu, a płytkami Raspberry Pi i odpowiada za odpowiednie przekierowywanie żądań REST API oraz przetwarzanie danych. Ułatwia zadanie aplikacji mobilnej przy wyszukiwaniu odpowiedniej płytki Raspberry Pi, która może spełnić żądanie użytkownika, czy też przy weryfikacji uprawnień użytkownika do danej płytki. Dla płytek Raspberry Pi natomiast upraszcza przekazywanie danych z ich urządzeń do wszystkich użytkowników mających do nich dostęp. To wszystko sprawia, że każda z obu stron komunikacji może w pełni skupić się na swoich głównych przeznaczeniach, natomiast zmartwienie dotyczące wzajemnego porozumiewania się pozostawić serwerowi w chmurze.

#### 3.2.2. Wybór technologii

Moduł serwera zewnętrznego został stworzony i wdrożony z wykorzystaniem następujących technologii:

- ASP .NET Web API [7]
- Microsoft Entity Framework Core [8]
- PostgreSQL [9]
- AWS (Amazon Web Services) [10]

ASP .NET Web API to framework umożliwiający budowę bezpiecznych REST API. Dokładna struktura projektu serwera z jego użyciem zostanie opisana w następnym podrozdziale.

Microsoft Entity Framework Core to biblioteka znacznie upraszczająca proces rozwijania serwera w połączeniu z bazą danych. Pozwala na wykonywanie wszystkich niezbędnych operacji na bazie danych za pomocą jedynie komend napisanych w języku C#, a także przyspiesza proces modyfikacji bazy danych dzięki wykorzystaniu tzw. migracji.

PostgreSQL jest popularnym i otwartym systemem zarządzania relacyjnymi bazami danych. Został wybrany przez zespół ze względu na łatwość w obsłudze oraz na koszty, gdyż wybór właśnie tej technologii umożliwił darmowe wystawienie bazy danych w chmurze.

AWS (Amazon Web Services) to platforma usług w chmurze, udostępniająca zasoby, dzięki którym zarówno serwer zewnętrzny, jak i baza danych, mogły zostać wdrożone na publiczne domeny. Oferuje atrakcyjny plan darmowego dostępu, który zakłada bezpłatne korzystanie z konkretnych zasobów przez pierwsze 12 miesięcy od czasu założenia konta użytkownika. To w głównej mierze przyczyniło się do wyboru przez zespół właśnie tej platformy.

### 3.2.3. Struktura projektu serwera

Projekt serwera zewnętrznego oparty o framework ASP .NET Web API składa się z następujących modułów:

- *kontrolery*, których zadaniem jest przechwytywanie żądań kierowanych do serwera i następnie wykonywanie na ich podstawie odpowiednich działań.
- *serwisy*, które odpowiadają za wykonanie powtarzalnych czynności wchodzących w skład logiki działania serwera, i które odciażają *kontrolery* od koniecznej do wykonania pracy nad żadaniami REST API.
- *modele*, które reprezentują obiekty obecne w systemie, takie jak: użytkownicy, czujniki, światła, czy zamek od drzwi.
- *baza danych*, która jest połączona z serwerem i zapisuje stany obecnych w systemie obiektów.

#### Kontrolery

Poniżej znajduje się lista obecnych w systemie *kontrolerów*:

- *AuthController*, odpowiedzialny za przetwarzanie żądań związanych z autoryzacją i autentykacją użytkowników
- *AlarmController*, odpowiedzialny za przetwarzanie żądań związanych z czujnikami alarmu
- *DoorLockController*, odpowiedzialny za przetwarzanie żądań związanych z zamkami od drzwi
- *LightsController*, odpowiedzialny za przetwarzanie żądań związanych z przełączalnymi światłami
- *SensorsController*, odpowiedzialny za przetwarzanie żądań związanych z czujnikami natężenia światła, wilgotności oraz temperatury
- *UserController*, odpowiedzialny za przetwarzanie żądań związanych z danymi użytkowników w bazie danych

#### Serwisy

Poniżej znajduje się lista obecnych w systemie *serwisów*:

- *AuthService*, odpowiedzialny za wykonanie rzeczywistej pracy serwera w obszarze autoryzacji i autentykacji użytkowników
- *DatabaseRefreshService*, odpowiedzialny za okresowe wykonywanie żądań do odpowiednich płytek Raspberry Pi w celu odświeżenia danych w bazie danych
- *DeviceService*, odpowiedzialny za wykonywanie żądań do płytek Raspberry Pi, a także pozyskiwanie aktualnego stanu urządzeń z bazy danych
- *UserService*, odpowiedzialny za wykonanie rzeczywistej pracy serwera w obszarze pozyskiwania danych użytkowników z bazy danych

#### Modele

Poniżej znajduje się lista obecnych w systemie *modeli*:

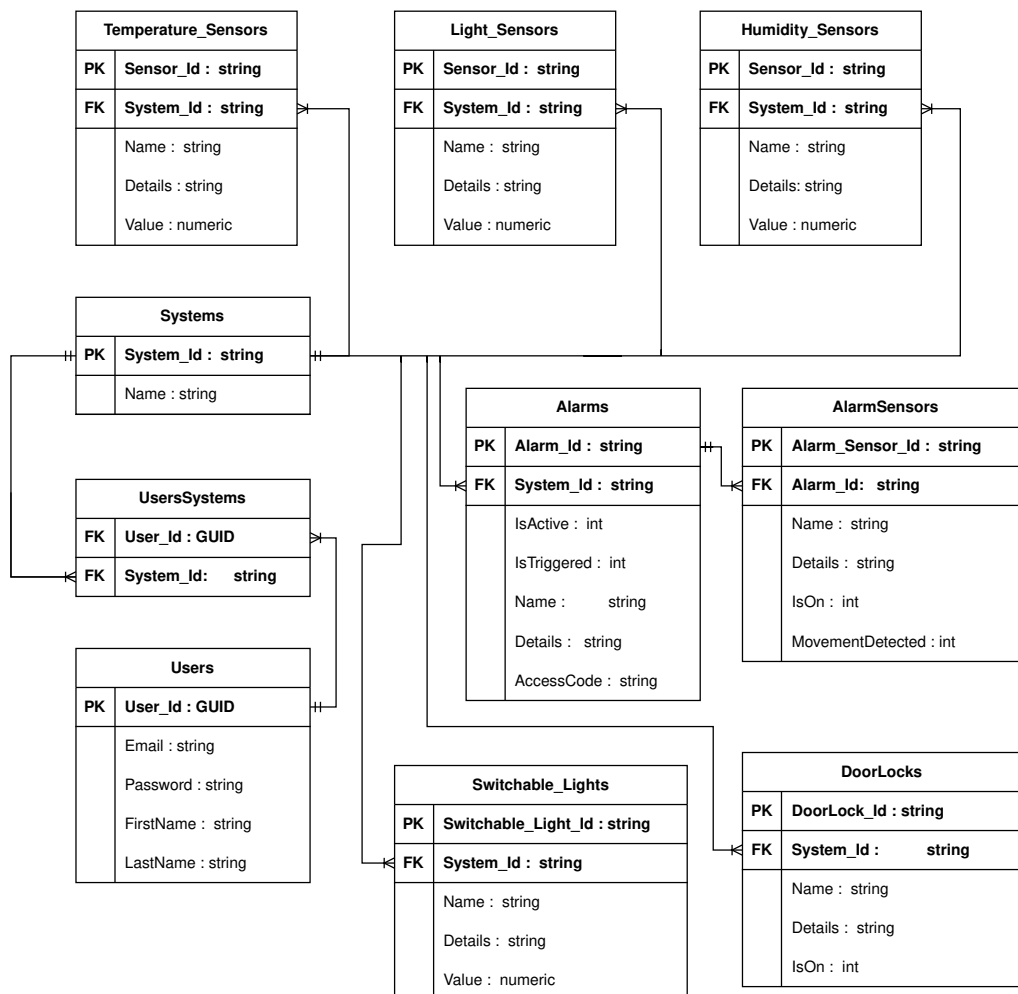
- *AlarmSensor*, skojarzony z czujnikiem alarmu
- *DoorLock*, skojarzony z zamkiem od drzwi



- *HumiditySensor*, skojarzony z czujnikiem wilgotności
- *SunlightSensor*, skojarzony z czujnikiem natężenia światła
- *TemperatureSensor*, skojarzony z czujnikiem temperatury
- *SwitchableLight*, skojarzony z przełączalnym światłem
- *User*, skojarzony z użytkownikiem
- *System*, skojarzony z systemem
- *Board*, skojarzony z płytą Raspberry Pi

## Baza danych

Baza danych została wygenerowana w sposób automatyczny na podstawie modeli i relacji między nimi z wykorzystaniem technologii Microsoft Entity Framework Core i migracji, które oferuje. Jej dokładny diagram przedstawiony został na poniższym rysunku:



Rysunek 3.10: Diagram bazy danych utworzonej przez zespół w trakcie prac

### 3.2. SERWER W CHMURZE

Z racji, iż do testowania opracowywanego systemu używano jedynie dwóch płytek Raspberry Pi, podjęto decyzję o pominięciu tabeli Boards w bazie danych i implementacji kierunku przekazywania żądań REST API bezpośrednio w kodzie serwera.

#### 3.2.4. API serwera

Dokładny opis API, które udostępnia serwer dla innych systemów znajduje się w załączniku do niniejszej pracy nazwanym „SmartHomeBackendAPI.yaml”.

#### 3.2.5. Przeprowadzone testy jednostkowe

Testy jednostkowe utworzone dla serwera dotyczyły funkcjonalności, za które odpowiedzialny jest wyłącznie on, tzn. autoryzacji i autentykacji oraz przetwarzania danych użytkowników. Weryfikacji poddane zostały:

- Dodawanie nowego użytkownika do bazy danych po rejestracji
- Usuwanie użytkownika z bazy danych przez administratora systemu
- Logowanie użytkownika
- Brak zezwolenia na rejestrację użytkownika przy podaniu adresu mailowego użytkownika już istniejącego w systemie
- Wyszukiwanie użytkowników w bazie danych

System udało się doprowadzić do stanu, w którym wszystkie powyższe testy przebiegały pomyślnie.

#### 3.2.6. Przeprowadzone testy integracyjne

Testy integracyjne dotyczyły komunikacji serwera z płytką Raspberry Pi. Weryfikacji poddane zostały:

- Włączanie i wyłączanie przełączalnych świateł
- Odczytywanie pomiarów z czujników

Również w tym przypadku system udało się doprowadzić do stanu, w którym oba powyższe testy przebiegały pomyślnie.

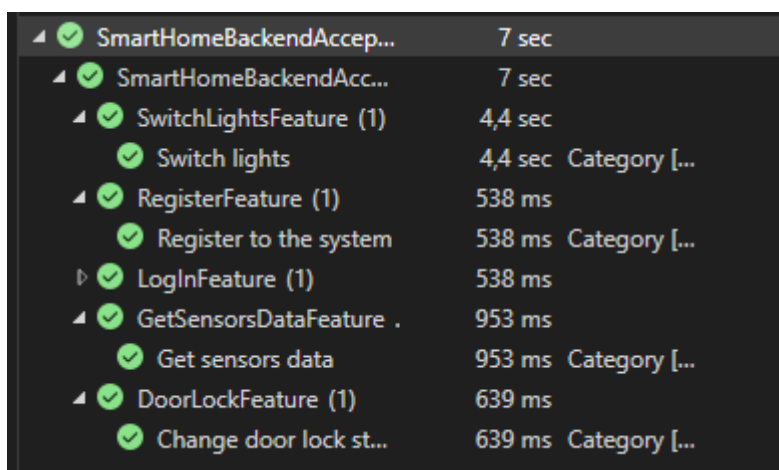
### 3.2.7. Przeprowadzone testy akceptacyjne

W celu finalnej weryfikacji funkcjonalności serwera zewnętrznego zespół wykonał testy akceptacyjne pełniące rolę zupełnie niezależnych od systemu klientów wykonujących żądania i oczekujących adekwatnych do nich odpowiedzi. Testy akceptacyjne dotyczące serwera w chmurze wykonane zostały z użyciem biblioteki SpecFlow for Visual Studio 2022 [11]. Biblioteka ta umożliwia tworzenie czytelnych w języku naturalnym scenariuszy oraz automatyczne generowanie na ich podstawie kodu, który służy implementacji testów.

Przetestowane zostały najważniejsze dla użytkownika funkcjonalności związane z:

- Przełączalnymi światłami
- Zamkami od drzwi
- Czujnikami temperatury, wilgotności, natężenia światła
- Logowaniem
- Rejestracją

Środowisko testowe nie było w żaden sposób powiązane z kodem projektu. Całość komunikacji odbywała się z użyciem API na wystawionym w chmurze AWS serwerze zewnętrznym, oraz serwera Flask symulującego Raspberry Pi i wystawiającego swój lokalny serwer (localhost) dzięki narzędziu ngrok [12]. Testy odbywały się zatem na środowisku mocno zbliżonym do środowiska docelowego systemu. Na poniższym rysunku przedstawiono pomyślny rezultat przebiegu wszystkich testów.



▲ ✓ SmartHomeBackendAccep...	7 sec	
▲ ✓ SmartHomeBackendAcc...	7 sec	
▲ ✓ SwitchLightsFeature (1)	4,4 sec	
✓ Switch lights	4,4 sec	Category [...]
▲ ✓ RegisterFeature (1)	538 ms	
✓ Register to the system	538 ms	Category [...]
▶ ✓ LoginFeature (1)	538 ms	
▲ ✓ GetSensorsDataFeature .	953 ms	
✓ Get sensors data	953 ms	Category [...]
▲ ✓ DoorLockFeature (1)	639 ms	
✓ Change door lock st...	639 ms	Category [...]

Rysunek 3.11: Pomyślny przebieg wszystkich testów akceptacyjnych serwera zewnętrznego widziany w Visual Studio 2022

### 3.2. SERWER W CHMURZE

Warto spojrzeć również na sposób generowania kodu do testów. Wykorzystywane są do tego scenariusze z biblioteki SpecFlow, które pozwalają na opis przebiegu testów w przejrzysty sposób, i których przykłady znajdują się na rysunkach poniżej.

```
1 Feature: LogIn
2
3 @logintag
4 Scenario: Log in to system
5     When I put the following credentials to log in
6         | Email | Password |
7         | adrian@test.com | 123 |
8     Then I log in to system successfully
```

Rysunek 3.12: Scenariusz testów dla logowania

```
1 Feature: DoorLock
2
3 @doorlocktag
4 Scenario: Change door lock state
5     Given Certain door locks in system
6     Then Successfully turn on and off the door locks
```

Rysunek 3.13: Scenariusz testów dla zamków od drzwi

### 3.3. Aplikacja mobilna

Pomysł na sterowanie systemem przy pomocy aplikacji mobilnej powstał z analizy podobnych rozwiązań z dziedziny Smart Home dostępnych na rynku. Telefon możemy mieć praktycznie zawsze przy sobie, a dzięki bezprzewodowemu połączeniu z internetem, możliwe jest zarządzanie urządzeniami z każdego miejsca w domu niczym przy pomocy pilota.

#### 3.3.1. Opis ogólny

Celem aplikacji mobilnej jest udostępnienie użytkownikowi interfejsu graficznego, przy którego pomocy mógłby łatwo zarządzać systemem z urządzenia mobilnego. Program umożliwia przeglądanie obecnych odczytów z dostępnych termometrów, mierników wilgotności i światła, włączanie i wyłączanie żarówek, sterowanie alarmem oraz zamkami do drzwi. W ramach bezpieczeństwa wymagane jest zalogowanie się przed korzystaniem z aplikacji, aby uniemożliwić sterowanie urządzeniami użytkownika przez osoby trzecie.

#### 3.3.2. Wybór technologii

Do napisania projektu został wykorzystany framework *Xamarin.Forms* [18], wspierany biblioteką *FreshMVVM* [19]. Oprogramowanie to zostało wybrane między innymi ze względu na nabyte wcześniej doświadczenie w jego użytkowaniu, ale także wykorzystanie języka C# do definiowania logiki biznesowej. Nauka C# była częścią programu naszych studiów, dzięki czemu członkowie zespołu niezaznajomieni z tworzeniem aplikacji mobilnych mogli bez znacznych problemów zrozumieć operacje na danych zawarte w kodzie. Było to szczególnie przydatne podczas pracy nad integracją modułów, kiedy możliwe było wspólne sprawdzanie poprawności działania komunikacji między aplikacją, a serwerem.

#### 3.3.3. Architektura aplikacji

Zgodnie z ideą *Xamarin.Forms* oraz *FreshMVVM*, projekt korzysta z architektury Model-View-Viewmodel (MVVM), w projekcie określanymi odpowiednio sufiksami Service, Page oraz PageModel w nazwach plików.

- Serwisy (C#) - Zawierają wszystko potrzebne do uzyskiwania danych. Istnieją jako pojedyncze instancje, których funkcje powodują wysyłanie zapytań do serwera, po czym zwracają otrzymane dane. Na potrzeby testowania utworzono także dla każdego serwisu odpowiedniki zawierające lokalnie zdefiniowane, sztuczne dane.

- Modele stron (C#) - Pośrednicy między serwisami a stronami. Obsługują logikę odpowiadającym im stron, przechowując dane uzyskane z serwisów oraz zmienne wpływające na wyświetlane elementy.
- Strony (XAML) - Definiują interfejs użytkownika przy pomocy specjalnego języka opracowanego przez Microsoft. Przypisują wybrane zmienne lub funkcje opisane w odpowiadającym im modelach pod elementy UI, same definiując logikę związaną jedynie z wyświetlaniem danych.

#### 3.3.4. Struktura projektu aplikacji

Folder „Fakes” zawiera implementacje serwisów do testowania bez wykorzystania serwera. Aby serwisów można było używać w aplikacji, należy je zarejestrować w kontenerze FreshIOC w głównym konstruktorze w pliku *App.xaml.cs*. Dla ułatwienia pracy z kodem, napisane zostały dwa różne bloki kodu definiujące serwisy testowe oraz korzystające z REST API, a to, który będzie użyty w trakcie działania aplikacji, zależy od zdefiniowanej na szczycie pliku zmiennej „#define FAKES”. Linie tę można odkomentować, by nie musieć polegać na serwerze, na przykład w przypadku oczekiwania na niedostępne jeszcze funkcje, lub testach interfejsu.

Folder „Models” zawiera definicje klas obiektów związanych z logiką biznesową aplikacji, np. urządzeń czy użytkownika. W folderze „BackendModels” zawarte są dodatkowo klasy odpowiadające obiektom dostarczonym przez odpowiedzi serwera. Ponieważ klasy obiektów wykorzystywanych w logice aplikacji zwykle zawierają dodatkowe właściwości, lub dziedziczą po specjalnych klasach (np. *ObservableObject*), zdecydowano się wyodrębnić te klasy od klas, do których automatycznie serializowane są odpowiedzi z serwera. Innym rozwiązaniem mogłoby być tworzenie po stronie serwera paczek NuGet, zawierających definicje klas odpowiadających przesyłanym przez serwer obiektom, tak by nie definiować tych klas oddzielnie po stronie aplikacji.

Folder „Infrastructure” zawiera definicje obiektów wykorzystywanych lokalnie przez aplikację, niezwiązanych z logiką biznesową. Jest tam na przykład klasa „AppState”, w której przechowywane są dane użytkownika i konfiguracja aplikacji w trakcie jej działania.

Ekranów przedstawiających urządzenia jest wiele, zatem by nie duplikować definicji sposobów wyświetlania stanów urządzeń, zdecydowano się utworzyć uniwersalne szablony dla każdego typu urządzenia. Są one umieszczone w folderze „ViewCells”.

Foldery „Converters” i „Behaviors” zawierają dodatkową logikę związaną z prezentowaniem danych użytkownikowi, powiązaną ze stronami. Odpowiada ona na przykład za zmianę wyglądu komórki światła po wyłączeniu lub włączeniu.

Plik „appsettings.json” został dodany jako plik zawierający konfigurację projektu. Został on

stworzony w celu ułatwienia pracy z serwerem, aby istniało jedno miejsce, gdzie zdefiniowane będą adresy URL wykorzystywanego API.

Projekty „SmartHome.Android” oraz „SmartHome.iOS” zawierają kod natywny wykorzystywany przez urządzenia mobilne. Dużą ilość logiki i wyglądu aplikacji można zawrzeć we wspólnym projekcie „SmartHome”, ale aby wykorzystywać funkcje zależne od platformy, jak np. powiadomienia, należy je zaimplementować właśnie w tych projektach.

W oddzielnym folderze „test” zawarte są projekty „SmartHome.Tests”, opisujące testy jednostkowe logiki aplikacji, oraz „SmartHome.UITests”, zawierający zautomatyzowane testy interfejsu aplikacji. Testy interfejsu są bardziej dokładnie opisane w dalszej sekcji „Testy akceptacyjne”.

### 3.3.5. Prototyp interfejsu użytkownika

Przed rozpoczęciem pracy nad aplikacją sporządzono następujące schematy wyglądu programu. Prototypy te stały się później wzorami widoków w fazie implementacji projektu.

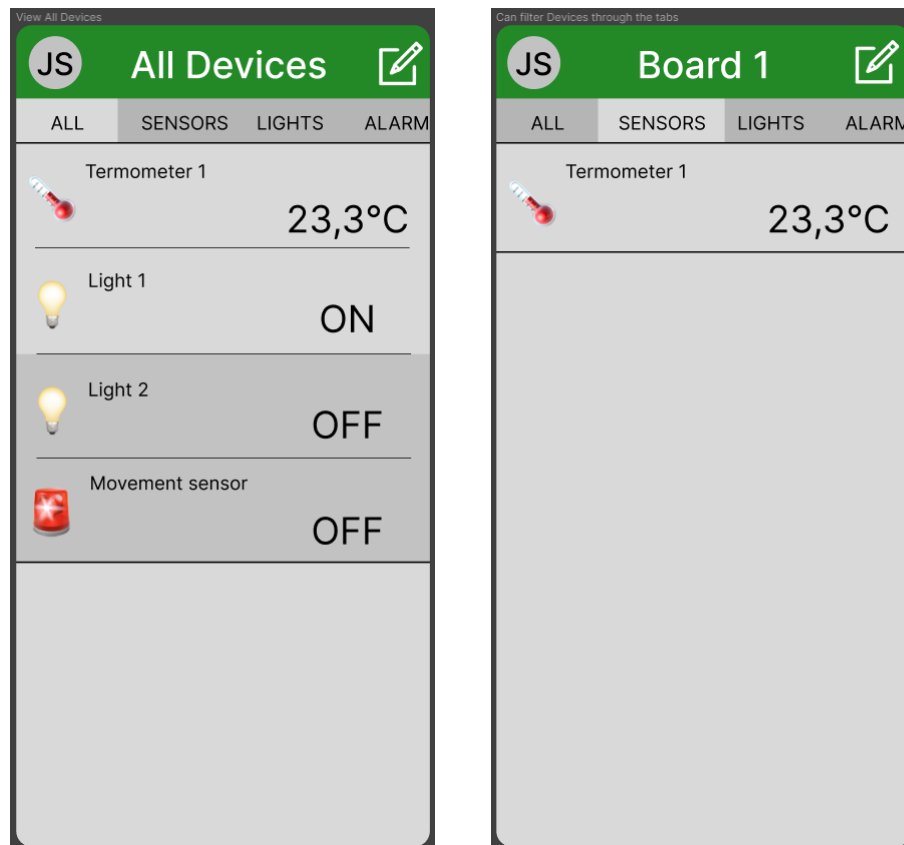
Widoki aplikacji były tworzone w programie *Figma* [20]. Poniżej znajdują się schematy wraz z opisami napisanymi przed fazą implementacji aplikacji. Zapis schematów eksportowany z programu znajduje się w załączniku „Smart Home App.fig”.

## Ekran główny

Po zalogowaniu się, użytkownik zostanie powitany ekranem przedstawiającym wszystkie dostępne mu urządzenia. We wszystkich ekranach pokazujących urządzenia z różnych typów możliwe jest filtrowanie po rodzaju urządzenia.

Naciskając ikonę edycji widoczną w prawym górnym rogu możemy wejść w tryb edycji nazw widocznych urządzeń. Wszystkie teksty nazw zmieniają się w pola tekstowe, pozwalając na ich edycję. Aby zaakceptować zmiany, należy następnie kliknąć w ikonę potwierdzenia, która zastąpi ikonę edycji. Przykład stanu edycji strony znajduje się w sekcji opisującej obsługę grup w aplikacji.

### 3.3. APLIKACJA MOBILNA

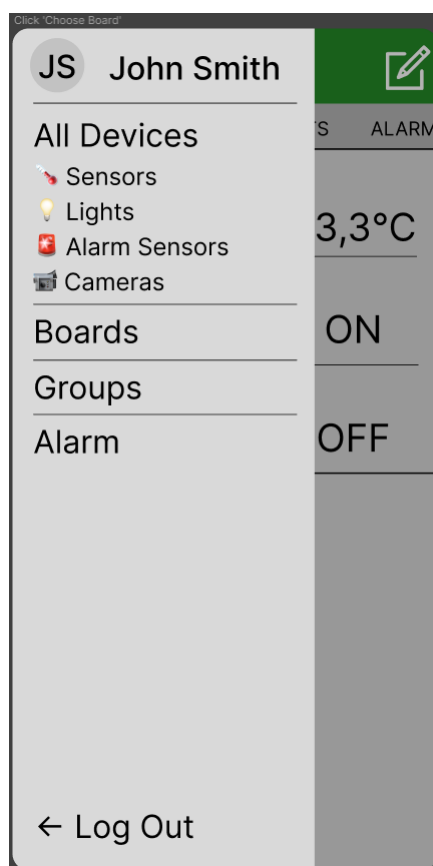


Rysunek 3.14: Ekran pokazujący wszystkie urządzenia użytkownika. Urządzenia można filtrować po ich typie, np. pokazując jedynie czujniki lub przełączniki światła.

#### Panel boczny

Głównym sposobem nawigacji między widokami jest panel boczny. Można go wysunąć przesuwając po ekranie w prawą stronę, powinien on wtedy wyskoczyć z lewego boku ekranu. Pozwala on na przechodzenie do ekranów wszystkich urządzeń, filtrowanie po przynależności do płytki oraz sterowanie czujnikami alarmu. Zawiera on także informację jako kto jesteśmy obecnie zalogowani oraz opcję wylogowania z aplikacji.

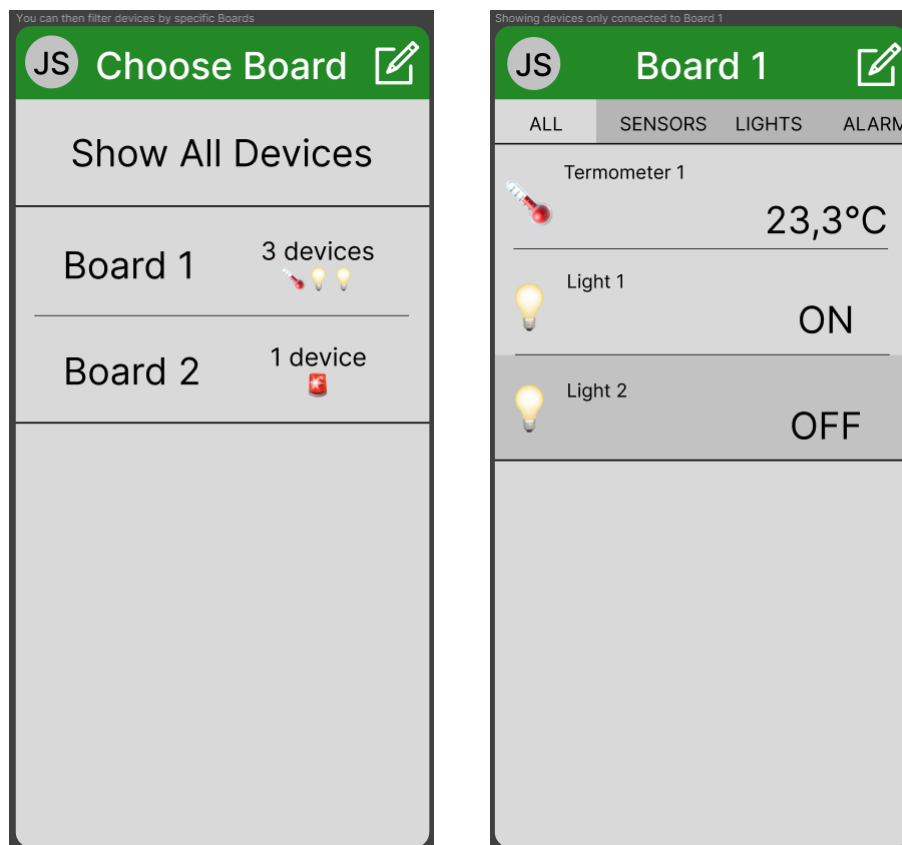




Rysunek 3.15: Panel boczny aplikacji. Używany do nawigacji.

### Sekcja filtrowania po przynależności do Raspberry Pi - *Boards*

Przejdźcie do ekranu układów Raspberry Pi (*Boards*) pozwala na zobaczenie urządzeń należących do konkretnych płytek. Aby wejść do tego ekranu, należy nacisnąć opcję *Boards* na panelu bocznym, a następnie wybrać układ, którego urządzenia chcemy zobaczyć. W ekranie tym możemy dodatkowo nacisnąć ikonę edycji widoczną w prawym górnym rogu by móc edytować nazwy płytek.



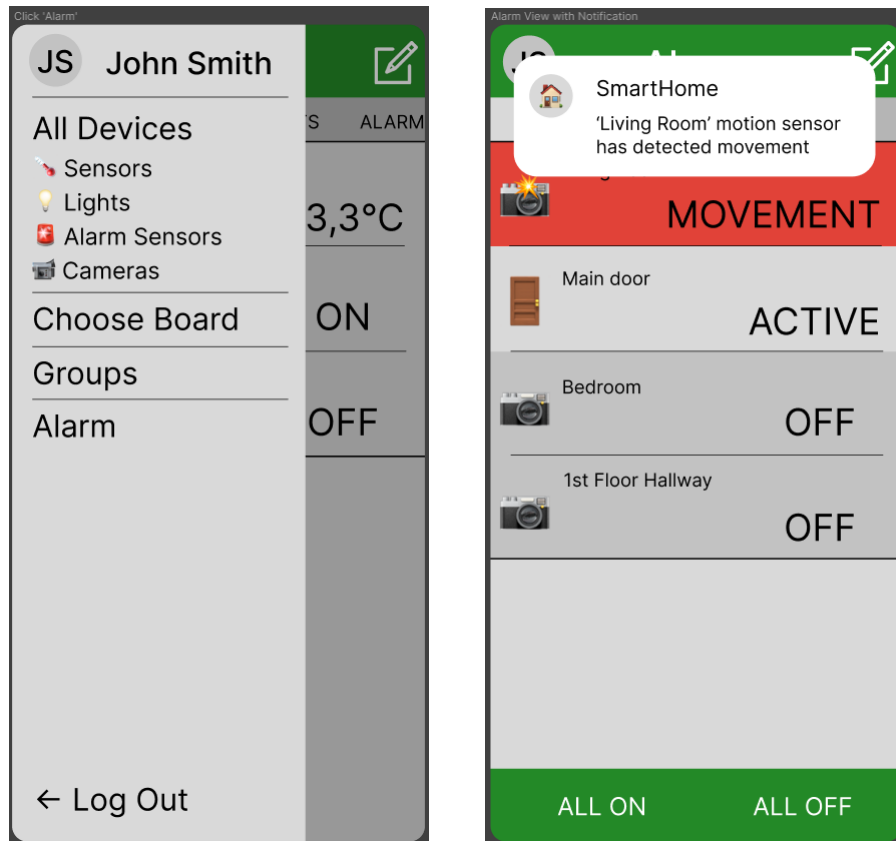
Rysunek 3.16: Ekran y filtrowania według układów RPi. Pozwalają podejrzeć urządzenia podłączone do konkretnej płytki.

## Alarm

Alarm posiada swój własny, dodatkowy ekran. Aby do niego wejść, należy kliknąć przycisk *Alarm* na dole panelu bocznego.

Na tym ekranie możemy na bieżąco podglądać stany czujników ruchu oraz otwarcia drzwi. Aby włączyć lub wyłączyć czujnik należy na niego kliknąć. Możliwe jest także włączenie i wyłączenie wszystkich czujników na raz przy pomocy przycisków na dole ekranu.

W przypadku wykrycia ruchu przez aktywny czujnik, aplikacja wyśle użytkownikowi powiadomienie zawierające informację, który czujnik wykrył ruch. W aplikacji czujnik taki będzie miał czerwone podświetlenie, aby dodatkowo zwrócić na niego uwagę użytkownika.



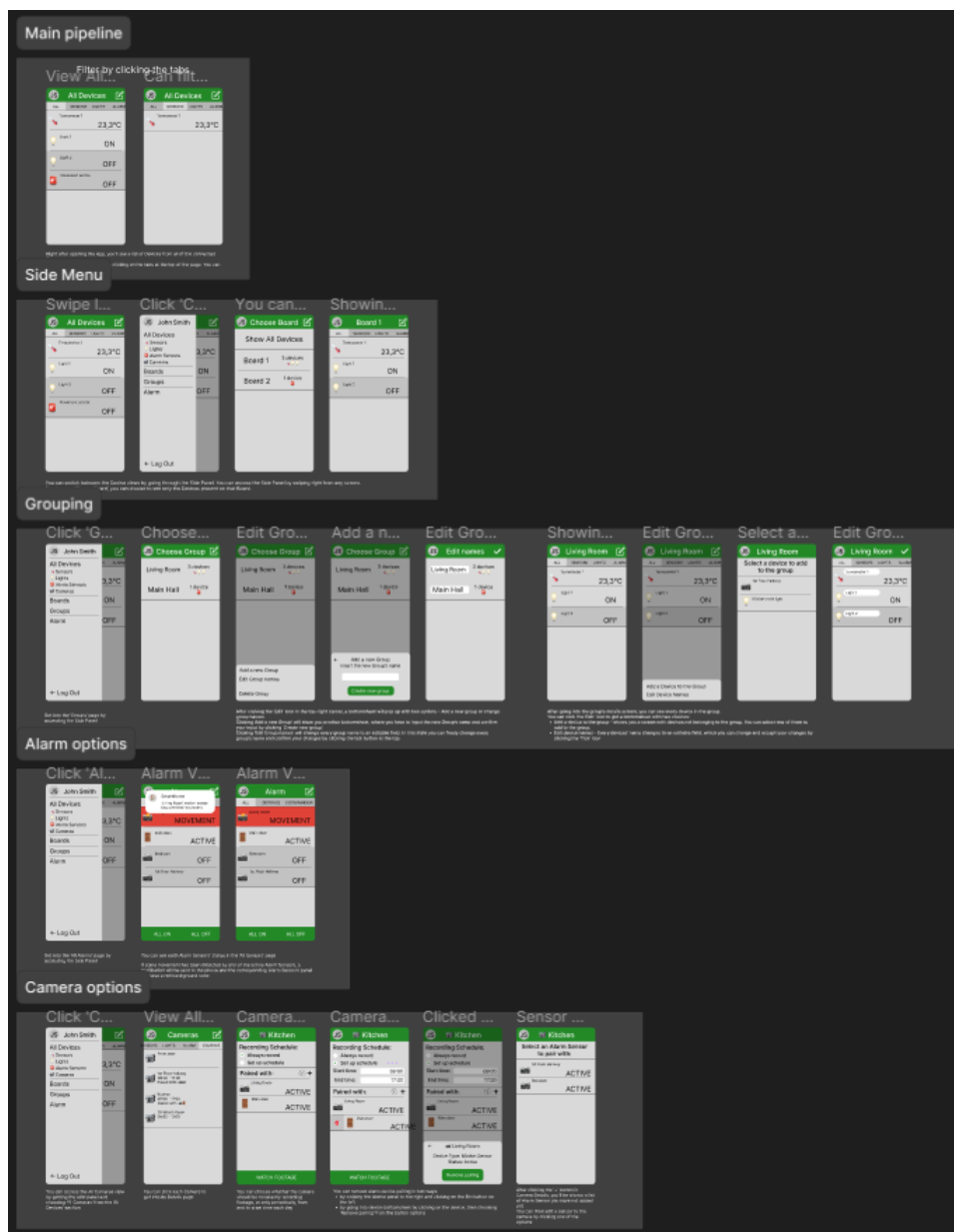
Rysunek 3.17: Ekran alarmu pozwala użytkownikowi na podglądanie stanów czujników ruchu.

### 3.3.6. Przebieg implementacji

Przed rozpoczęciem pracy ustalone zostały wymagania funkcjonalne, które aplikacja miała spełnić. Poprzez utworzenie przykładowych ekranów przedstawiających sterowanie programem, powstał pewien punkt zaczepienia, który stanowił swego rodzaju definicję ukończenia (*ang. Definition of Done*) kolejnych segmentów modułu.

Ekran w programie Figma pogrupowane zostały w sekcje przedstawiające wygląd aplikacji w trakcie wykonywania podstawowych czynności, takich jak przeglądanie list urządzeń czy nawigacja między ekranami. Każda sekcja zawierała opis wykonywanych akcji oraz szczegóły zachowania aplikacji.

### 3.3. APLIKACJA MOBILNA



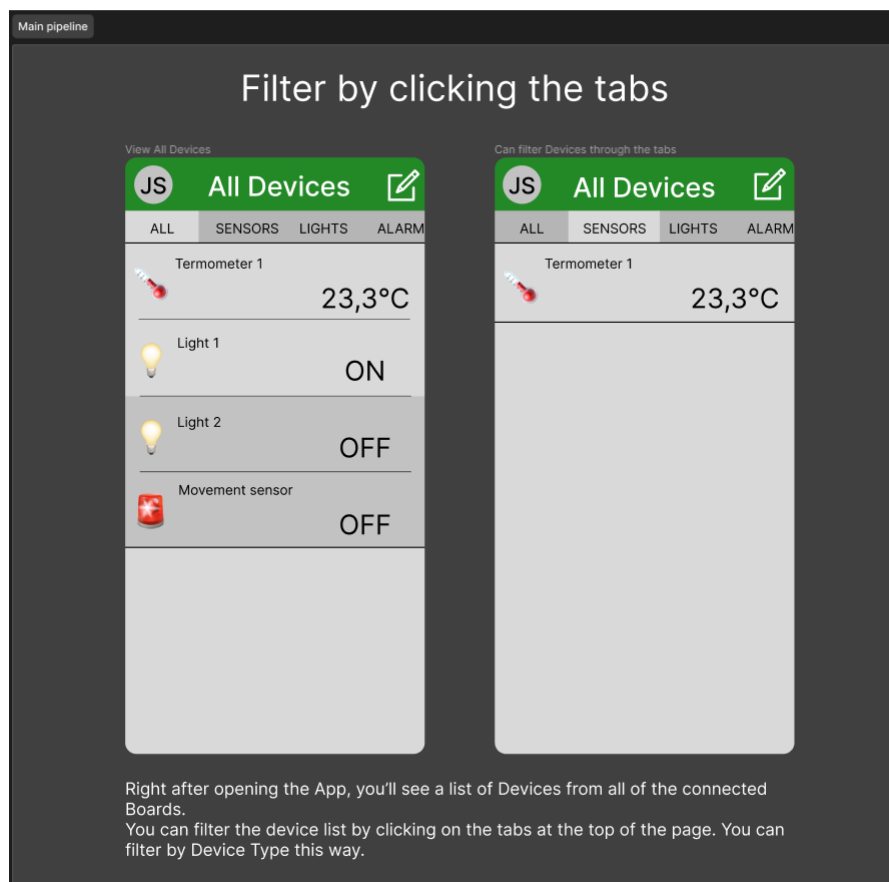
Rysunek 3.18: Wszystkie utworzone przykłady widoków

W trakcie implementacji często zdarzało się, że pomysły z fazy projektowania nie mogły być odwzorowane 1 : 1 wewnątrz aplikacji, czy to przez ograniczenia frameworku, czy przez nieoczekiwane zachowania niektórych elementów. Idee trzeba było modyfikować, próbując zaradzić problemom.

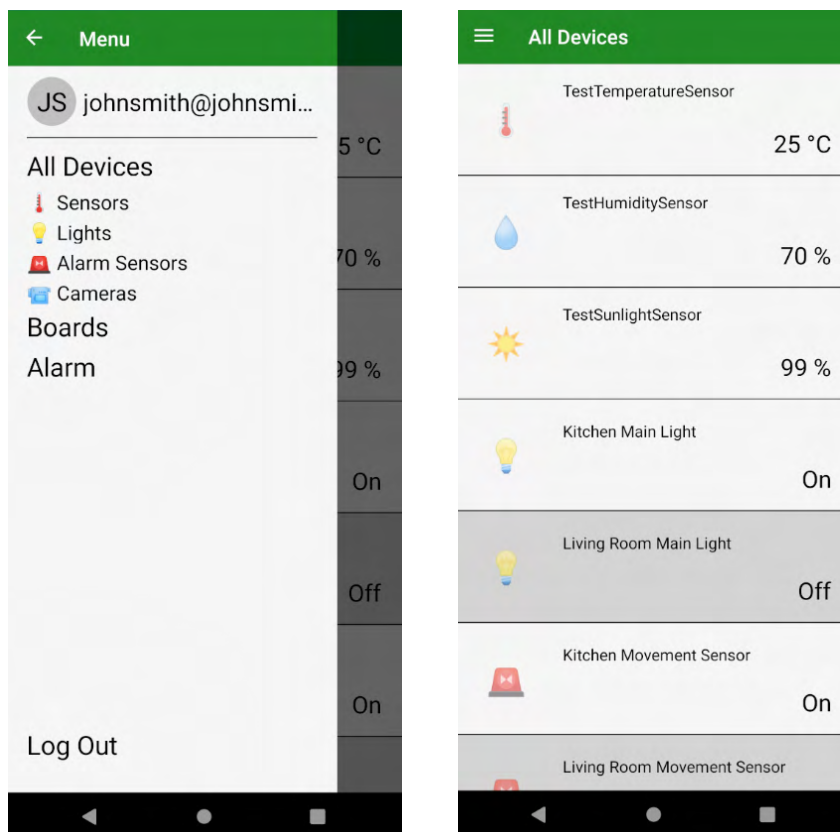
Przykładem może być widok list urządzeń użytkownika. Początkowy pomysł (rys. 3.16) zakładał podzielenie tego ekranu na zakładki, widoczne na górze, pozwalające na szybkie filtrowanie listy według typu urządzenia. W trakcie implementacji okazało się, że element frameworku opisujący taki układ umożliwia przechodzenie między ekranami nie tylko poprzez naciskanie zakładek na górze, ale także przez przesuwanie po ekranie w lewo czy prawo. Prowadziło to do problemów w używaniu aplikacji, ponieważ w połączeniu z panelem nawigacyjnym, który można wysunąć

przesuwając od lewego brzegu ekranu w prawo, przesuwanie po ekranie czasami powodowało przejście do kolejnej zakładki, a czasami do panelu nawigacji.

Problem ten rozwiązano poprzez zrezygnowanie z zakładek na górze, zamiast tego oddelegowując przechodzenie między przefiltrowanymi listami do panelu nawigacyjnego (rys. 3.17).



Rysunek 3.19: Sekcja dokumentacji przedstawiająca widok listy wszystkich urządzeń dostępnych dla użytkownika.



Rysunek 3.20: Finalny wygląd głównej listy urządzeń oraz panelu bocznego. Filtrowanie po typach urządzeń zostało przeniesione do oddzielnych widoków, między którymi nawigację umożliwia wysuwany z boku panel.

Niektórych pomysłów widocznych na schematach nie udało się finalnie zaimplementować, skupiając się na pełnym ukończeniu bardziej podstawowych funkcjonalności. Część z nich została opisana w dalszej sekcji „Możliwości dalszego rozwoju projektu”.

W ogólnej ocenie, udało się osiągnąć wygląd oraz funkcjonalność aplikacji odpowiadające zamierzonym na początku celom.

#### 3.3.7. Testy akceptacyjne

Testy akceptacyjne aplikacji mobilnej zaimplementowano jako zautomatyzowane testy interfejsu użytkownika (*ang. UI tests*). Znajdują się one w projekcie „SmartHome.UITests” w folderze „test”. Korzysta on z frameworku *Xamarin.UITest* [21], pozwalającego na pisanie testów interfejsu uniwersalnie dla platform Android i iOS. Udostępnia on funkcje związane z interakcją z aplikacją poprzez znajdowanie i naciskanie wyświetlanych elementów czy przesuwanie po ekranie według wyznaczonych współrzędnych.

Zostały napisane trzy testy sprawdzające działanie podstawowych funkcjonalności programu. Korzystają one z lokalnych wersji serwisów, nie łączących się z serwerem, by dodatkowy moduł nie wpływał na ich przebieg. Każdy z nich rozpoczyna się logowaniem do aplikacji, następnie

przechodząc różne scenariusze, które obejmują:

- Wysuwanie panelu bocznego
- Wyłączenie obecnego na ekranie światła
- Wyłączenie, a następnie włączenie wybranego czujnika ruchu

W ten sposób sprawdzona została poprawność aktualizacji widoków aplikacji w trakcie przeprowadzania podstawowych operacji na urządzeniach oraz logowania.

Testy były przeprowadzane wykorzystując emulator urządzenia Pixel 5 z zainstalowanym Androidem 10.0 (API 29), procesorem x86\_64, 1 GB pamięci wewnętrznej oraz rozdzielczością ekranu 1080 na 2340, 440 dpi.

### 3.4. Komunikacja pomiędzy modułami

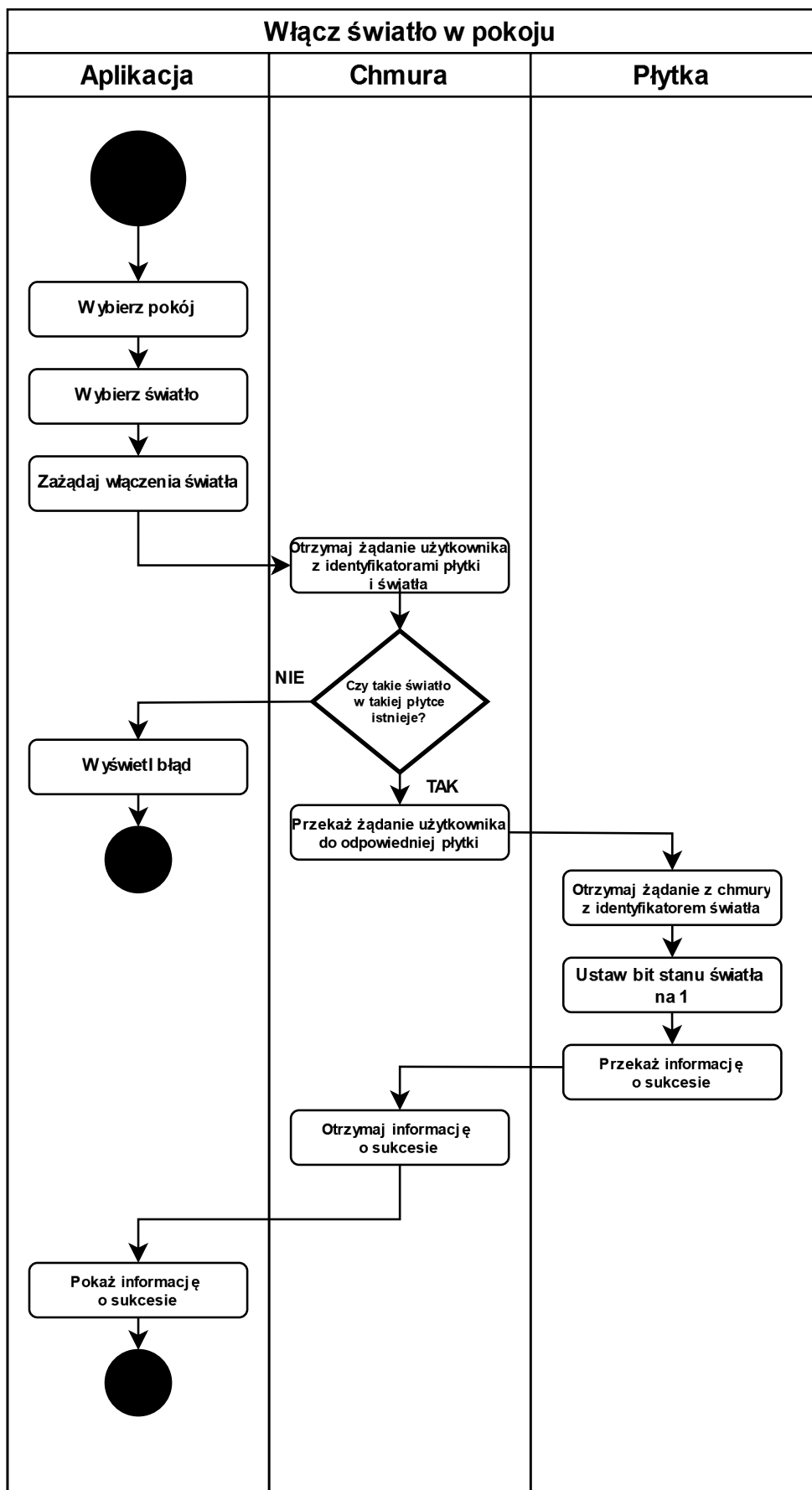
Aby system działał prawidłowo, muszą istnieć w nim odpowiednie schematy komunikacji między aplikacją mobilną i serwerem w chmurze oraz między serwerem w chmurze i systemami wbudowanymi. Zarówno przy pierwszej, jak i drugiej relacji zapewnione są osobne API umożliwiające wymianę informacji i pozyskiwanie danych między modułami. Różnią się one od siebie pod względem potrzeb obu relacji i ich celów. API wystawione dla komunikacji *aplikacja mobilna - serwer w chmurze* ma na celu zapewniać bezpieczny dostęp do systemu oraz pozyskiwanie danych z systemów wbudowanych w sposób możliwie najbardziej przejrzysty i klarowny. Z kolei API wystawione dla komunikacji *serwer w chmurze - systemy wbudowane* ma na celu zapewniać prawidłową funkcjonalność układów elektronicznych na podstawie żądań użytkownika. Warto zagłębić się w techniczne aspekty przedstawionych wyżej mechanizmów.

Na podstawie przykładów najbardziej typowych zastosowań, do których wszystkie pozostałe mają analogiczny schemat wymiany informacji, przedstawiony zostanie dokładniejszy opis komunikacji w naszym systemie. Przejdziemy przez 2 różne scenariusze wykorzystania systemu przez użytkownika:

- Włączanie światła w pokoju
- Rejestracja użytkownika w systemie

Poniżej, na rysunku 3.21, znajduje się diagram aktywności prezentujący przebieg działań podejmowanych przez system w celu włączenia światła przez użytkownika. Diagram ten w dobry sposób ilustruje relacje między modułami systemu w tym konkretnym przypadku użycia.

### 3.4. KOMUNIKACJA POMIĘDZY MODUŁAMI

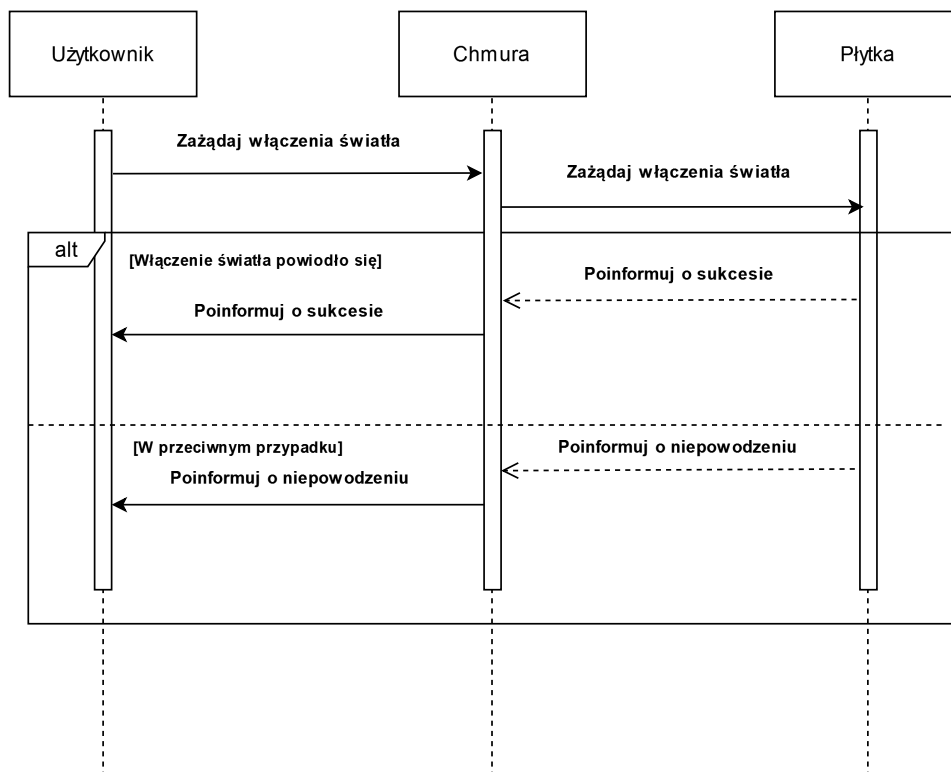


Rysunek 3.21: Diagram aktywności dla włączania światła z poziomu aplikacji



Użytkownik na początku wybiera światło z konkretnego pokoju, a następnie wysyła żądanie jego włączenia do serwera. Serwer bada, czy żądanie jest właściwe i jeżeli tak, to przekazuje je do odpowiedniej płytki Raspberry Pi. Płytkę po otrzymaniu żądania od serwera przesyła wysokie napięcie na odpowiednie wyjście GPIO do światła w pokoju, powodując jego włączenie. Warto zauważyć, że jedynym działaniem, które wykonuje aplikacja mobilna w całym procesie to przekazanie żądania użytkownika do chmury, a następnie oczekiwanie na ostateczny rezultat. Rzeczywista praca nad spełnieniem żądania użytkownika odbywa się we współpracy chmury z systemem wbudowanym, do którego dane światło jest bezpośrednio podłączone. Zadaniem chmury jest zweryfikować żądanie użytkownika pod kątem prawidłowości i autoryzacji. Kiedy żądanie trafia do systemu wbudowanego, ze względów bezpieczeństwa jest ono ponownie analizowane pod kątem autoryzacji, a następnie do danego światła z odpowiedniego wyjścia kierowane jest niezerowe napięcie, które zapala światło. Taki podział systemu umożliwia optymalną komunikację użytkownika z układami elektronicznymi oraz przekazywanie użytkownikowi tylko i wyłącznie niezbędnych informacji na temat uruchamianych procesów. W jeszcze prostszy sposób można przedstawić całe zjawisko za pomocą diagramu komunikacji zgodnie z rysunkiem 3.22.

### Zapalenie światła



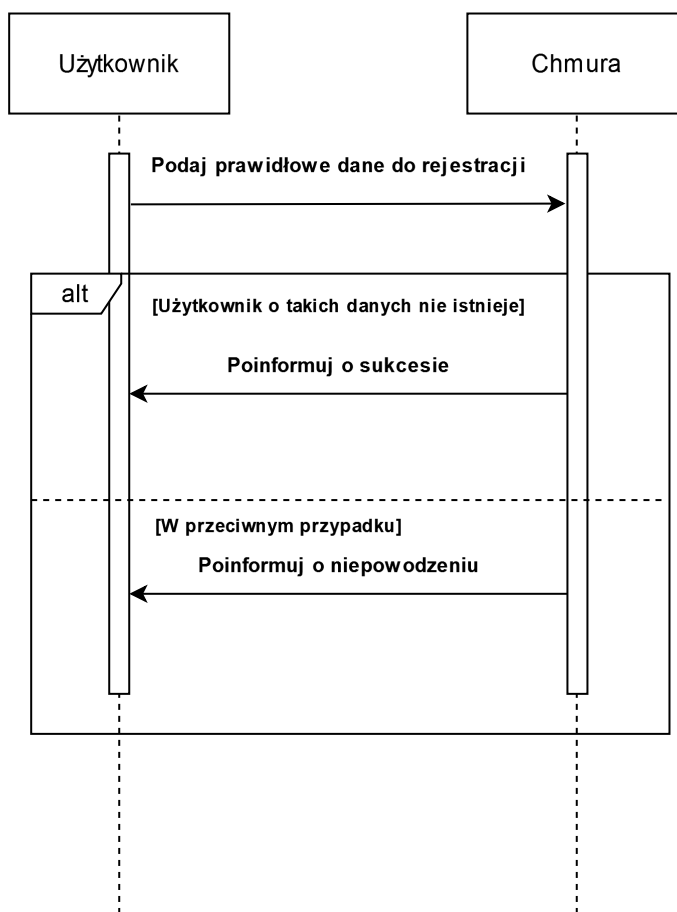
Rysunek 3.22: Diagram komunikacji dla włączania światła z poziomu aplikacji

### 3.4. KOMUNIKACJA POMIĘDZY MODUŁAMI

Opis tego diagramu można wykonać w analogiczny sposób do poprzedniego diagramu. Użytkownik przekazuje żądanie włączenia światła chmurze, która po odpowiednim jego przetworzeniu przekazuje je odpowiedniej płytce Raspberry Pi i w wyniku użytkownik informowany jest albo o sukcesie, albo o porażce całej operacji.

Mimo, że możliwe jest utworzenie systemu bez modułu serwera zewnętrznego poprzez bezpośrednią komunikację systemu wbudowanego i aplikacji, został on jednak utworzony, aby ułatwić proces weryfikacji użytkownika. Ponadto, karty SD, z których korzysta system wbudowany, nie są przystosowane do długoterminowego utrzymania często zmieniającej się bazy danych.

#### Rejestracja do systemu



Rysunek 3.23: Diagram komunikacji dla rejestracji użytkownika do systemu

Rejestracja do systemu jest przykładowym procesem, w którym komunikacja przebiega tylko i wyłącznie pomiędzy aplikacją, a chmurą, co udowadnia konieczność istnienia obu tych modułów w całym systemie. Zgodnie z powyższym diagramem użytkownik po podaniu prawidłowych danych do rejestracji otrzymuje odpowiedź od serwera informującą o sukcesie lub o porażce całej operacji.

Przedstawiony został teoretyczny model komunikacji pomiędzy najważniejszymi komponentami systemu. Przejdziemy teraz do technicznego aspektu tej funkcjonalności i odpowiemy na pytanie co sprawia, że wymyślony przez nas model rzeczywiście działa. Każda aktywność systemu rozpoczyna się od użytkownika, a więc od aplikacji mobilnej. Żądania użytkownika przekształcane są w żądania protokołu REST API i przekazywane do serwera w chmurze wdrożonego z pomocą platformy AWS na konkretny adres URL w sieci (urządzenie mobilne musi mieć połączenie z internetem). Serwer odbiera żądanie, wykonuje odpowiednie dla niego działania i przekazuje w przekształconej formie przez tunel ngrok do serwera Flask odpowiedniej płytki Raspberry Pi 4B w odpowiednim budynku. Serwer Flask po odebraniu żądania z chmury, wykonuje na płycie odpowiednie działanie, np. zapala światło, aktywuje zamek od drzwi.

#### 3.4.1. Protokół komunikacji

Podstawowym protokołem komunikacji między modułami systemu jest REST API. Zdecydowano się na jego użycie z powodu prostoty jego działania, dużej popularności wśród społeczności programistów, a także możliwości rozdzielenia oprogramowania na serwerowe i klienckie, a tym samym większej modułowości systemu. Wspomniana modułowość okazała się być kluczowa w podziale obowiązków pomiędzy członków zespołu, a co za tym idzie, wydajności całej pracy.

#### 3.4.2. Konfiguracja sieci

Każdy z modułów systemu musi być objęty pewnymi ograniczeniami związanymi z konfiguracją sieci.

- Urządzenie, na którym uruchamiana jest aplikacja mobilna powinno mieć połączenie z internetem.
- Serwer odbierający żądania użytkownika powinien być wystawiony w chmurze na publicznej domenie z wykorzystaniem platformy AWS. To umożliwi użytkownikom dostęp do systemu elektronicznego niezależnie od ich odległości względem urządzeń Raspberry Pi, co w wielu sytuacjach może być przydatne.
- Systemy wbudowane powinny mieć dostęp do internetu. Dodatkowo konieczny jest eksport ich lokalnych serwerów (localhostów) na odpowiednio nazwane domeny z wykorzystaniem narzędzia ngrok. To umożliwi nawiązywanie połączenia z chmurą w przypadku nadchodzących żądań użytkowników.

Jedną z zalet takiego rozwiązania jest chociażby możliwość kontroli nad układami elektronicznymi budynku będąc daleko poza nim. Problemem natomiast może się okazać uzależnienie funk-

### 3.4. KOMUNIKACJA POMIĘDZY MODUŁAMI

cjonalności całego systemu od połączenia z internetem. Jeżeli którykolwiek z modułów (Płytki RaspberryPi 4B, Serwer w chmurze, Aplikacja mobilna) napotka problem z nawiązaniem połączenia internetowego, użytkownik będzie mógł doświadczyć trudności z użytkowaniem systemu inteligentego domu.

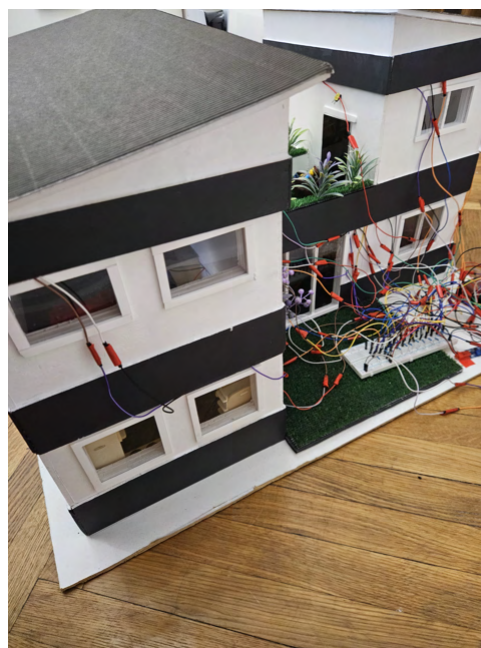
#### 3.4.3. Specyfikacja API

W załączniku do tego dokumentu zostały dodane kody API na serwerze zewnętrznym i na systemie wbudowanym. Znajdują się odpowiednio w plikach „SmartHomeBoardAPI.yaml” oraz „SmartHomeBackendAPI.yaml”.

## 4. Makieta domu

### 4.1. Opis makiety

W celu przetestowania działania systemu postanowiono wykonać makietę domu symulującą rzeczywisty budynek, w którym opracowywany system mógłby zostać zainstalowany. Poniżej znajdują się zdjęcia jej zewnętrznego wyglądu.



Rysunek 4.1: Po lewo: przód wykonanej makiety domu Po prawo: tył makiety

Ściany makiety wykonane zostały z płyt styropianowych, okna z folii transparentnej, natomiast podłoga okryta została wykładziną. Elewacja makiety pomalowana została farbami, starając się przy tym wytworzyć w niej chropowatość, która jest często spotykana w rzeczywistych rozwiązaniach. Również w trakcie projektowania całej konstrukcji starano się oddać w niej jak największy realizm, dbając o obecność pokoi takich jak kuchnia, łazienka, salon, sypialnia, balkon, czy też o odpowiednie ustawienie mebli i wystarczająco dużą przestrzeń. Zadbano również o możliwość odsłonięcia wnętrza makiety poprzez wysuwalne kondygnacje.

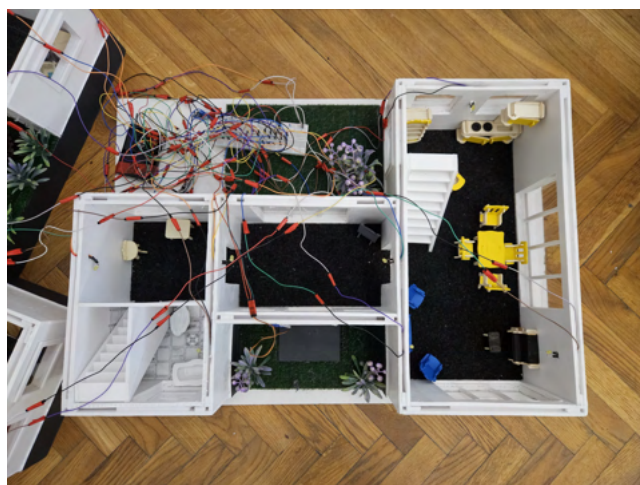
#### 4.1. OPIS MAKIETY



Rysunek 4.2: Rzut z góry na dach makiety domu



Rysunek 4.3: Rzut z góry na 1. piętro makiety domu



Rysunek 4.4: Rzut z góry na parter makiety domu

W celu zamontowania elektroniki niezbędnej do przetestowania funkcjonalności systemu, konieczne było zaprojektowanie odpowiedniego układu elektronicznego możliwie najlepiej symulującego tego typu układy w rzeczywistych budynkach.

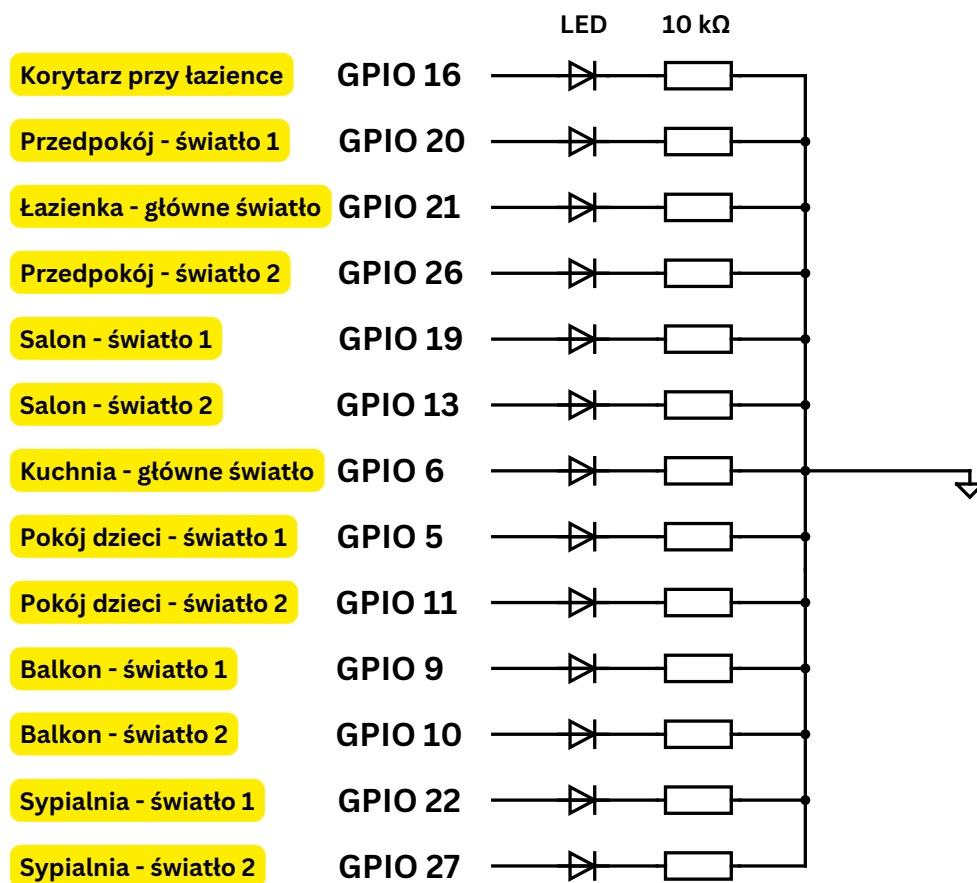
## 4.2. Układ elektroniczny wewnątrz makiety domu

W niniejszym rozdziale dokładniej opisany zostanie układ elektroniczny zamontowany w makiecie domu i służący za symulację układów elektronicznych obecnych w rzeczywistych budynkach. Utworzony przez nas układ jest w pełni kontrolowalny przez użytkownika. W jego skład wchodzi urządzenia takie jak: diody LED symulujące rzeczywiste światła, serwomechanizm pełniący funkcję zamka od drzwi, czujniki temperatury, wilgotności, natężenia światła służące do pomiaru tych wielkości fizycznych, czy też czujnik ruchu służący uaktywnieniu alarmu obecnego w systemie. Wszystkie wymienione urządzenia połączone są ze sobą w układzie w sposób równoległy.

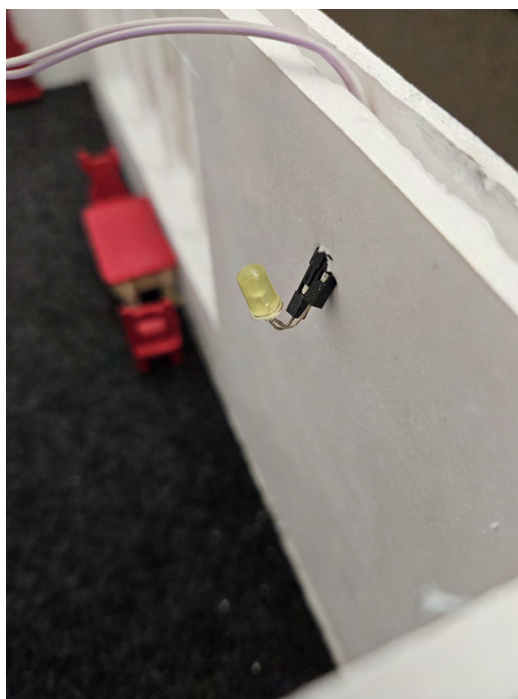
Na rysunku 4.5 przedstawiony został schemat układu elektronicznego odpowiadającego za działanie wszystkich światel obecnych w makiecie domu.

Diody LED połączone są ze sobą w obwodzie w sposób równoległy, co zapobiega sytuacjom, w których uszkodzenie jednej diody uniemożliwiłoby działanie wszystkim pozostałym. Ponadto przy każdej diodzie obecny jest rezystor, co chroni diody przed zgromadzeniem zbyt dużej ilości prądu i ostatecznie spalaniem. Do każdego wyjścia GPIO w płytce Raspberry Pi przyporządkowane jest jedno światło. Takie rozwiązanie umożliwia kontrolę nad każdym światłem obecnym w makiecie domu oddzielnie. Pojedynczą diodę LED zamontowaną w makiecie można zobaczyć na poniższym zdjęciu (rysunek 4.6).

#### 4.2. UKŁAD ELEKTRONICZNY WEWNĄTRZ MAKIETY DOMU



Rysunek 4.5: Układ elektroniczny świateł w makiecie domu.



Rysunek 4.6: Dioda LED zamontowana w makiecie domu



Kolejne urządzenie podłączone do układu elektronicznego, które zostanie omówione to serwomechanizm SG-90 - micro - 180°. Jego dokładny wygląd i lokalizacja przedstawione zostały na poniższym rysunku.



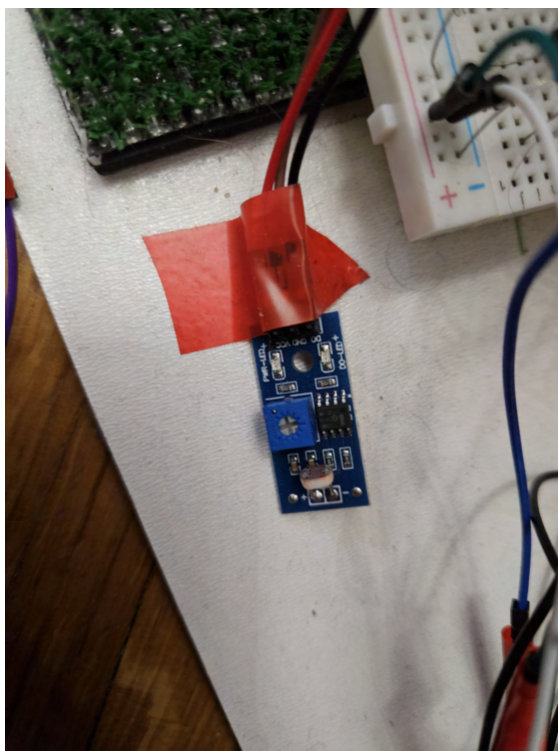
Rysunek 4.7: Serwomechanizm pełniący rolę zamka od drzwi wejściowych makiety domu

Użytkownik ma możliwość aktywować lub dezaktywować zamek za pomocą aplikacji mobilnej. W chwili aktywacji serwomechanizm przyjmuje pozycję 90°, w której blokuje drzwi wejściowe, natomiast po dezaktywacji urządzenie powraca do pozycji 0°.

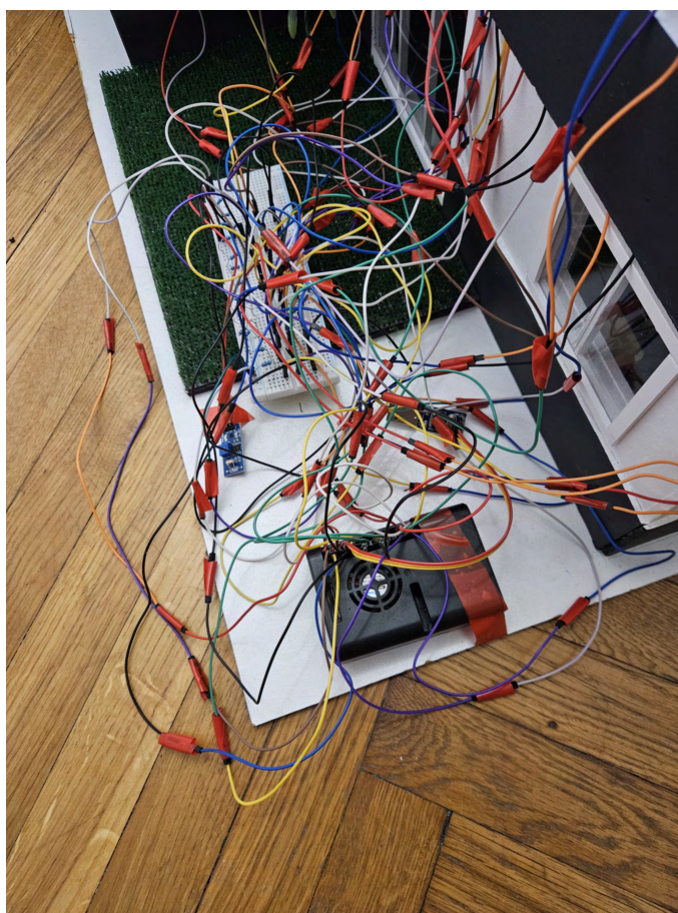
Dodatkowo bezpośrednio podłączony do układu elektronicznego makiety domu jest również czujnik natężenia światła przedstawiony na rysunku 4.8.

Jest to cyfrowy czujnik, a więc jako zwrotną informację przekazuje jedynie wysokie napięcie w sytuacji, gdy nie pada na niego światło o dużym natężeniu lub niskie napięcie w przeciwnym przypadku. Końcowa wartość natężenia zwracana użytkownikowi jest równa stosunkowi ilości czasu, w którym na czujnik padało światło o dużym natężeniu do całkowitego czasu działania serwera na płycie Raspberry Pi.

#### 4.2. UKŁAD ELEKTRONICZNY WEWNĄTRZ MAKIETY DOMU



Rysunek 4.8: Czujnik natężenia światła zamontowany w makiecie



Rysunek 4.9: Płytką Raspberry Pi wraz z płytą stykową

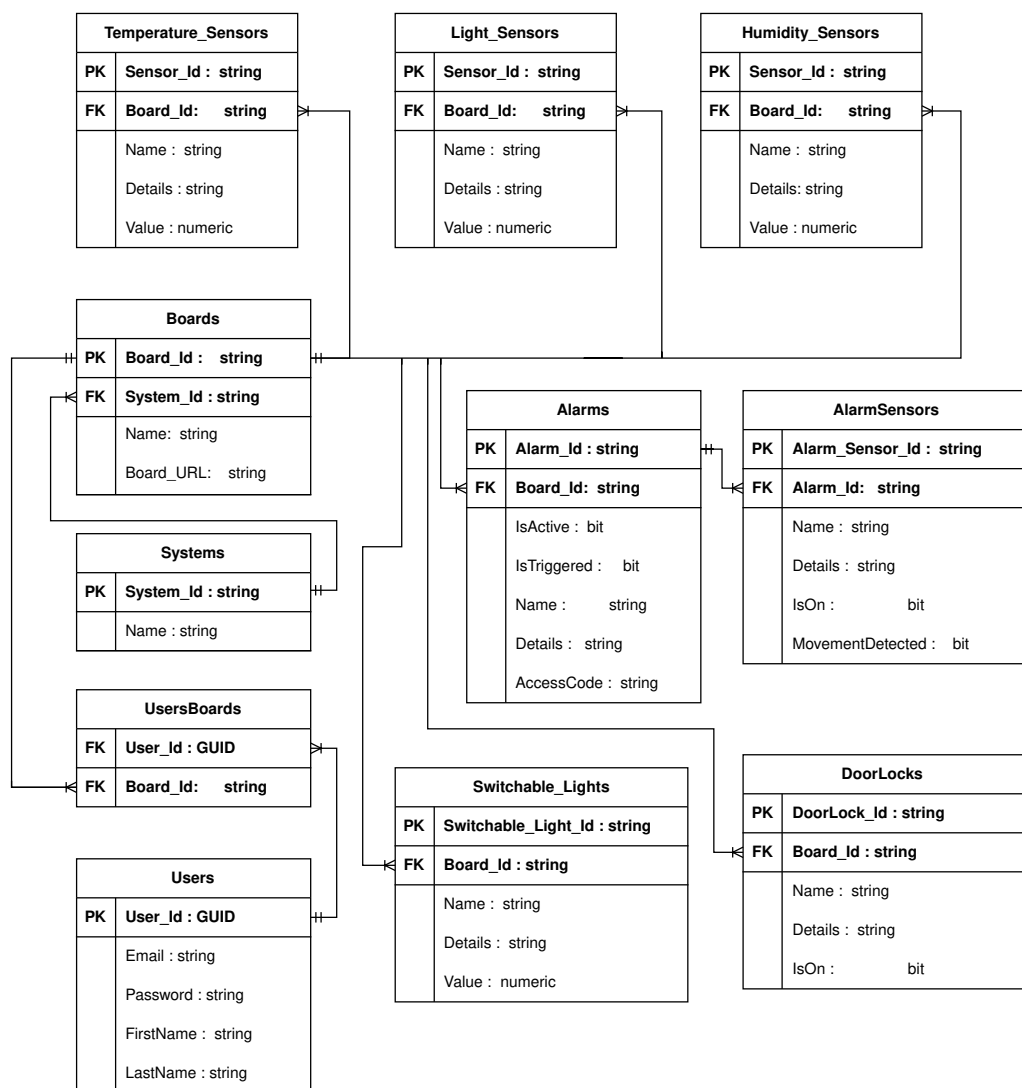
Na zdjęciu 4.9 przedstawiony został rdzeń całego układu elektronicznego, a mianowicie płytki Raspberry Pi, która steruje wszystkimi wchodzącymi w jego skład urządzeniami, a także płytki stykowa ułatwiająca konstrukcję całego układu.

To właśnie przez komponenty widoczne na tym zdjęciu przekazywane są wszystkie żądania użytkowników do obecnych w układzie elektronicznym urządzeń. Na zdjęciu widoczne jest również oklejenie połączeń kabli taśmą izolacyjną, dzięki której układ jest bardziej odporny na ewentualne rozłączenia kabli pod wpływem działania sił fizycznych.

## 5. Możliwości dalszego rozwoju projektu

### 5.0.1. Wsparcie dla działania wielu układów Raspberry Pi jednocześnie

Serwer w obecnym stanie posiada zakodowane na stałe adresy URL, poprzez które odwołuje się do poszczególnych układów RPi. Możliwe jest jedynie ustawienie różnych adresów dla różnych typów urządzeń, przez co w najlepszym wypadku możnaby oddelegować po jednej płytce dla wszystkich sensorów, świateł i czujników alarmu.



Rysunek 5.1: Diagram bazy danych wspierający działanie wielu urządzeń Raspberry Pi.

Propozycją rozwiązania byłoby dodanie tabeli *Boards* w bazie danych posiadającą kolumnę *Board\_URL*, odpowiadającą adresowi URL serwera uruchomionego na systemie wbudowanym, po którym możliwe byłoby połączenie się z danym układem Raspberry Pi. Aby możliwe było określenie przynależności układu do użytkownika, można wtedy dodać także tabelę *UsersBoards* opisującą relację *many to many*, zawierającą w kolejnych wpisach identyfikatory użytkowników i płytek, które do nich należą. Rola tabeli *Systems* ogranicza się jedynie do przetrzymywania informacji o płytkach Raspberry Pi wchodzących w skład systemu.

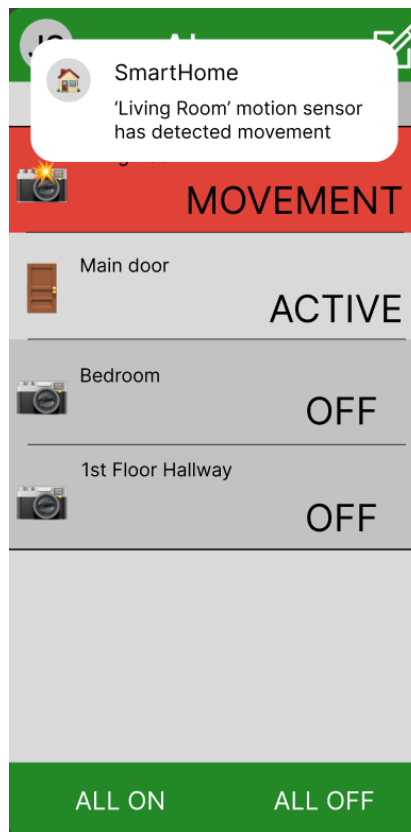
### 5.0.2. Aktualizowanie stanów urządzeń na bieżąco

W obecnej wersji aplikacji, stany urządzeń są pobierane z serwera w momencie wejścia na dowolny z ekranów listujących urządzenia oraz przez manualne odświeżenie listy poprzez pociągnięcie od szczytu ekranu w dół. To rozwiązanie daje użytkownikowi dobre pojęcie o stanach urządzeń w przypadku, gdy jest jedyną korzystającą z nich osobą. W przypadku, gdy dwóch użytkowników korzysta z tych samych urządzeń na raz, zmiany w stanach zobaczy jedynie osoba, która zainicjowała zmianę. Sytuacja ta wynika z ograniczenia komunikacji *serwer - aplikacja* do komunikacji jednokierunkowej, gdzie jedynie aplikacja wysyła zapytania do serwera, nie w drugą stronę.

Proponowanym rozwiązaniem w tym wypadku byłoby użycie biblioteki SignalR [22] w aplikacji oraz na serwerze. Umożliwia ono komunikację dwukierunkową *serwer-aplikacja*, dzięki czemu serwer mógłby powiadamiać aplikację o zaistniałych zmianach w stanach urządzeń, pobudzając ją do odświeżenia. Pakiet SignalR jest proponowany ze względu na działanie w środowisku .NET Core, przez co wymagane do jego działania zmiany mogłyby być bez problemu wprowadzone i na serwer, i do aplikacji. Jest szybko działającym rozwiązaniem rozwijanym przez Microsoft, udostępniającym funkcje wysokiego poziomu sterujące komunikacją *aplikacja-serwer* w czasie rzeczywistym.

### 5.0.3. Powiadomienia o wykryciu ruchu w aplikacji mobilnej

W przypadku wykrycia ruchu przez uzbrojony czujnik alarmu, użytkownik otrzymałby powiadomienie na swoim urządzeniu mobilnym informujące go o zaistniałym zdarzeniu. Dzięki temu byłby w stanie szybko zareagować na przykład na próby wtargnięcia na swoją posesję.



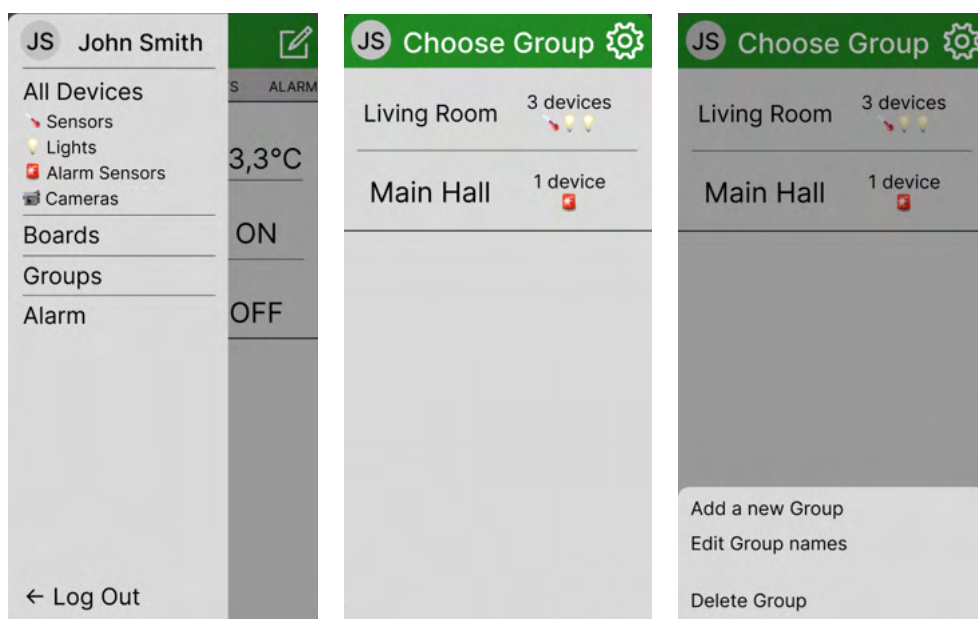
Rysunek 5.2: Schemat wyglądu powiadomienia o wykryciu ruchu.

#### 5.0.4. Zmiana nazw urządzeń

Jednym z pomysłów na zwiększenie komfortu korzystania z systemu jest udostępnienie użytkownikom możliwości zmiany nazw poszczególnych urządzeń. Obecnie nazwy te są ustalane przez administratorów, jednak zaimplementowanie tej funkcji w aplikacji znacznie ułatwiłoby korzystanie z urządzeń, na przykład poprzez możliwość ustawiania intuicyjnych nazw lub ich wygodnej zmiany.

#### 5.0.5. Niestandardowe grupowanie urządzeń

Kolejnym pomysłem na zwiększenie komfortu jest pozwolenie użytkownikom na tworzenie własnych grup urządzeń, by mogli je łatwiej i szybciej znajdować w aplikacji. Użytkownik miałby wtedy dostęp do dodatkowego ekranu z wyborem grup, wyglądającego podobnie do zaimplementowanego ekranu wyboru układu Raspberry Pi. Z poziomu tego ekranu mógłby tworzyć nowe lub usuwać istniejące grupy urządzeń. Po naciśnięciu wybranej grupy, w opcjach ekranu byłoby możliwe dodawanie i usuwanie urządzeń z grupy.



Rysunek 5.3: Propozycja wyglądu nawigacji, wyboru i opcji tworzenia i usuwania grup.

#### 5.0.6. Rozpoznawanie głosu

Możliwe byłoby dodanie sterowania urządzeniami przy pomocy komend głosowych, wykorzystując gotowe biblioteki języka C# do rozpoznawania mowy.

#### 5.0.7. Łączenie działania urządzeń

Wygodną dla użytkownika funkcjonalnością mogłoby być ustawianie przez niego obsługiwać pewnych zdarzeń wykrytych przez urządzenia na systemie wbudowanym. Na przykład ustawienie automatycznego włączenia światła w momencie, gdy sensor światła przestanie je wykrywać, albo otrzymanie powiadomienia, gdy temperatura wzrośnie na pewien poziom. Część tych funkcjonalności jest możliwe do wprowadzenia, a niektóre wymagają rozszerzenia całego systemu.

#### 5.0.8. Cyberbezpieczeństwo

W obecnym stanie systemu, bezpieczeństwo jest ograniczone jedynie do weryfikacji adresu e-mail oraz hasła użytkownika, by sprawdzić czy podane ciągi znaków znajdują się w bazie danych. W przypadku komercjalizacji projektu, takie zabezpieczenie nie byłoby wystarczające by uchronić system przed włamaniami. Pierwszymi krokami do podjęcia w tą stronę było by wykorzystanie funkcji haszujących do szyfrowania hasła oraz użycie tokenów JWT do autoryzacji zapytań płynących z aplikacji.

## 6. Podsumowanie i wnioski

Implementacja systemu z niniejszej pracy pozwoliła członkom zespołu pogłębić wiedzę w obszarze wytwarzania systemów inteligentnych domów, zapoznać się z trudnościami, jakie za tym stoją oraz uzyskać satysfakcjonujący rezultat. Osiągnięcie takiego efektu wymagało przewyciężenia wielu problemów, które również zostaną dokładnie opisane w tym rozdziale.

### 6.1. Trudności stojące za wytwarzaniem systemów inteligentnego domu

Zagadnienia, na które producenci systemów inteligentnego domu powinni zwrócić szczególną uwagę można wypunktować następująco:

- Szybkość komunikacji pomiędzy modułami tworzącymi system - ważne jest, aby algorytmy wykorzystywane w programach modułów systemu skupiały się na wydajności i zapewniały możliwie najszybsze odpowiedzi na żądania użytkowników.
- Dokładność dokumentacji API - niedokładna dokumentacja API serwera zewnętrznego i serwerów uruchomionych na systemach wbudowanych może przyczynić się do powstawania nieporozumień pomiędzy członkami zespołu wytwarzającymi system, generując opóźnienia.
- Jakość wykorzystywanych urządzeń elektronicznych - urządzenia elektroniczne o niskiej jakości najpewniej w bardzo krótkim czasie przestaną być użyteczne, dlatego w celu zapewnienia satysfakcji z wykorzystania systemu należy zadbać o ich wysoką jakość.
- Cyberbezpieczeństwo - nie można dopuścić do sytuacji, w której ktoś niepowołany przejmie kontrolę nad urządzeniami elektronicznymi w domu użytkownika systemu. Należy zadbać o dostarczenie wysokiej jakości zabezpieczeń przed możliwymi atakami hakerskimi.
- Łatwość wdrożenia systemu - należy zaprojektować możliwie najbardziej przyjazny dla potencjalnych klientów sposób wdrożenia systemu inteligentnego domu w ich własnych domostwach. Skomplikowana instrukcja i brak wsparcia przy montażu podobnych systemów jest najczęściej odstraszającym czynnikiem, który zmienia zdanie niejednej zastanawiającej się nad kupnem podobnego produktu osoby.



## 6.2. Doświadczenia z projektem

W czasie implementacji projektu doświadczono zarówno mniej jak i bardziej poważnych problemów. Dokładniej opisane zostaną one w poniższych podrozdziałach.

### 6.2.1. System wbudowany

Największym problemem w tej części projektu okazał się system operacyjny i instalowanie na nim części bibliotek. Spowodowało to opuszczenie zbudowania małego systemu Unix z projektu i instalację systemu operacyjnego RaspberryPi OS. Pewne trudności spowodowane były również brakiem doświadczenia w korzystaniu z języka Python3, nie stanowiło to jednak poważnych przeszkód.

### 6.2.2. Serwer w chmurze

Główne wyzwanie z jakim musiał się mierzyć nasz zespół podczas pracy nad serwerem w chmurze dotyczyło utworzenia prostego interfejsu API i odpowiednich schematów zwracanych danych, a także technicznego wdrożenia serwera do chmury. Poniżej w punktach dokładniej opisane zostaną konkretne przypadki problemów, z którymi nasz zespół mierzył się przy podołaniu wspomnianemu wcześniej wyzwaniu:

#### 1. Pierwszy test komunikacji *urządzenie mobilne - serwer - Raspberry Pi*

Przed rozpoczęciem zaawansowanej implementacji serwera i pozostałych modułów pragnęliśmy najpierw dojść do momentu, w którym komunikacja na linii *urządzenie mobilne - serwer - Raspberry Pi* działa i powoduje zapalenie się diody. Okazało się to nie być zbyt proste, ponieważ laptop, na którym lokalnie uruchomiony był serwer i z którego korzystał zespół, z powodu zabezpieczeń czynnie odrzucał żądania REST API pochodzące z urządzenia mobilnego, natomiast Raspberry Pi pomimo połączenia kablem Ethernet z wcześniej wspomnianym laptopem nie mógł nawiązać połączenia. W celu umożliwienia komunikacji *serwer - urządzenie mobilne* konieczne okazało się użycie narzędzia o nazwie „ngrok”. Pozwala ono w bezpieczny sposób udostępnić na publicznej domenie localhost danego urządzenia, dzięki czemu dowolne urządzenie może przysyłać do niego swoje żądania bez problemów z zabezpieczeniami. W przypadku umożliwienia komunikacji *Raspberry Pi - serwer* konieczne okazało się dokonanie odpowiednich ustawień w systemie operacyjnym laptopa.

### 2. Wdrożenie serwera do chmury

Pierwszym pomysłem zespołu na wdrożenie serwera do chmury było wykorzystanie platformy Azure. Nie posiadała ona jednak ofert umożliwiających darmowe wykorzystanie. Ponadto przez brak doświadczenia jeden z członków zespołu po niewyłączeniu jednej z jednostek obliczeniowych zmuszony został do zapłacenia dużej kwoty pieniężnej. Postanowiliśmy więc poszukać innych rozwiązań. Tym najbardziej nam odpowiadającym okazała się platforma AWS. Umożliwia ona darmowe korzystanie przez 12 miesięcy od daty utworzenia konta z niektórych swoich zasobów, co pozwoliło nam testować nasz serwer w docelowej postaci bez ponoszenia żadnych kosztów.

### 3. API i schematy danych

Wyobrażenie API i schematu zwracanych przez serwer danych często różniło się pomiędzy członkami zespołu. Niejednokrotnie musieliśmy dokonywać poprawek, które umożliwiały transmisję danych w coraz prostszy i optymalniejszy sposób. Uświadomiło nam to jednak, jak kluczową rolę stanowi wcześnie ustalenie schematu komunikacji między modułami. Dla przykładu w fazie integracji nie było łatwo dojść do porozumienia w sprawie dokładnej formy komunikacji aplikacji mobilnej z serwerem. W rezultacie często trzeba było dokonywać poprawek po kończeniu pracy nad dalszymi etapami, czy też schematy API ulegały zmianom w kodzie, ale nie w dokumentacji.

## Bibliografia

- [1] Google Home, <https://home.google.com/what-is-google-home/> (korzystano 15 stycznia 2024)
- [2] Home Assistant, <https://www.home-assistant.io/docs/> (korzystano 15 stycznia 2024)
- [3] openHAB, <https://www.openhab.org/docs/> (korzystano 15 stycznia 2024)
- [4] Flask, [https://pl.wikipedia.org/wiki/Flask\\_\(framework\)](https://pl.wikipedia.org/wiki/Flask_(framework)) (korzystano 22 października 2023)
- [5] RPi.GPIO, <https://pypi.org/project/RPi.GPIO/> (korzystano 22 października 2023)
- [6] pad4pi, <https://pypi.org/project/pad4pi/> (korzystano 19 grudnia 2023)
- [7] ASP .NET Web API, <https://learn.microsoft.com/pl-pl/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api> (korzystano 20 października 2023)
- [8] Microsoft Entity Framework Core, <https://learn.microsoft.com/en-us/ef/core/> (korzystano 15 grudnia 2023)
- [9] PostgreSQL, <https://pl.wikipedia.org/wiki/PostgreSQL> (korzystano 19 grudnia 2023)
- [10] AWS (Amazon Web Services), [https://pl.wikipedia.org/wiki/Amazon\\_Web\\_Services](https://pl.wikipedia.org/wiki/Amazon_Web_Services) (korzystano 2 grudnia 2023)
- [11] Specflow for Visual Studio 2022, <https://marketplace.visualstudio.com/items?itemName=TechTalkSpecFlowTeam.SpecFlowForVisualStudio2022> (korzystano 10 stycznia 2024)
- [12] ngrok, <https://ngrok.com/docs> (korzystano 5 listopada 2023)
- [13] Testowanie za pomocą pozorowania platformy, <https://learn.microsoft.com/pl-pl/ef/ef6/fundamentals/testing/mocking> (korzystano 29 grudnia 2023)
- [14] Zadania w tle z hostowanymi usługami w ASP.NET Core, <https://learn.microsoft.com/pl-pl/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-8.0&tabs=visual-studio> (korzystano 17 grudnia 2023)

- [15] Bell, Charles, „Beginning Sensor Networks with Arduino and Raspberry Pi”, 2013
- [16] Marcin Bis, „Linux w systemach embedded”, Legionowo, Wydawnictwo BTC, 2011
- [17] Karim Yaghmour et al., Building Embedded Linux Systems, 2nd Ed., O'Reilly Media, 2008
- [18] Framework Xamarin.Forms, <https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms> (korzystano 24 stycznia 2024)
- [19] Michael Ridland, FreshMvvm for Xamarin.Forms, <https://github.com/rid00z/FreshMvvm> (korzystano 8 stycznia 2024)
- [20] Strona programu Figma, <https://www.figma.com/> (korzystano 24 stycznia 2024)
- [21] Framework testów akceptacyjnych aplikacji mobilnej, Xamarin UITest, <https://learn.microsoft.com/en-us/appcenter/test-cloud/frameworks/uitest/xamarin-forms?tabs=windows> (korzystano 24 stycznia 2024)
- [22] Pakiet SignalR, <https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-8.0> (korzystano 14 stycznia 2024)

## Spis rysunków

2.1	Przypadki użycia pokazujące możliwości interakcji użytkownika z systemem . . .	8
3.1	Schemat architektury systemu . . . . .	16
3.2	Diagram klas kontrolera światła. . . . .	18
3.3	Diagram klas kontrolera alarmu. . . . .	19
3.4	Diagram klas kontrolera sensorów. . . . .	20
3.5	Diagram klas kontrolera zamków. . . . .	21
3.6	Diody led połączone z RaspberryPi . . . . .	23
3.7	Czujnik ruchu połączony z RaspberryPi . . . . .	24
3.8	Czujnik ruchu połączony z RaspberryPi . . . . .	25
3.9	Klawiatura membranowa połączona z RaspberryPi . . . . .	26
3.10	Diagram bazy danych utworzonej przez zespół w trakcie prac . . . . .	30
3.11	Pomyślny przebieg wszystkich testów akceptacyjnych serwera zewnętrznego widziany w Visual Studio 2022 . . . . .	32
3.12	Scenariusz testów dla logowania . . . . .	33
3.13	Scenariusz testów dla zamków od drzwi . . . . .	33
3.14	Ekran pokazujący wszystkie urządzenia użytkownika. Urządzenia można filtrować po ich typie, np. pokazując jedynie czujniki lub przełączniki światła. . . . .	37
3.15	Panel boczny aplikacji. Używany do nawigacji. . . . .	38
3.16	Ekran filtrowania według układów RPi. Pozwalają podejrzeć urządzenia podłączone do konkretnej płytki. . . . .	39
3.17	Ekran alarmu pozwala użytkownikowi na podglądanie stanów czujników ruchu. .	40
3.18	Wszystkie utworzone przykłady widoków . . . . .	41
3.19	Sekcja dokumentacji przedstawiająca widok listy wszystkich urządzeń dostępnych dla użytkownika. . . . .	42
3.20	Finalny wygląd głównej listy urządzeń oraz panelu bocznego. Filtrowanie po typach urządzeń zostało przeniesione do oddzielnych widoków, między którymi nawigację umożliwia wysuwany z boku panel. . . . .	43

3.21	Diagram aktywności dla włączania światła z poziomu aplikacji . . . . .	45
3.22	Diagram komunikacji dla włączania światła z poziomu aplikacji . . . . .	46
3.23	Diagram komunikacji dla rejestracji użytkownika do systemu . . . . .	47
4.1	Po lewo: przód wykonanej makiety domu Po prawo: tył makiety . . . . .	50
4.2	Rzut z góry na dach makiety domu . . . . .	51
4.3	Rzut z góry na 1. piętro makiety domu . . . . .	51
4.4	Rzut z góry na parter makiety domu . . . . .	51
4.5	Układ elektroniczny świateł w makiecie domu. . . . .	53
4.6	Dioda LED zamontowana w makiecie domu . . . . .	53
4.7	Serwomechanizm pełniący rolę zamka od drzwi wejściowych makiety domu . . . .	54
4.8	Czujnik natężenia światła zamontowany w makiecie . . . . .	55
4.9	Płytki Raspberry Pi wraz z płytką stykową . . . . .	55
5.1	Diagram bazy danych wspierający działanie wielu urządzeń Raspberry Pi. . . . .	57
5.2	Schemat wyglądu powiadomienia o wykryciu ruchu. . . . .	59
5.3	Propozycja wyglądu nawigacji, wyboru i opcji tworzenia i usuwania grup. . . . .	60

## Spis tabel

2.1	Opis przypadków użycia dla użytkownika i systemu . . . . .	9
2.2	Wymagania нефункционалне . . . . .	10
2.3	Analiza SWOT projektu - Zagrozenia i szanse wewnętrzne . . . . .	12
2.4	Analiza SWOT projektu - Zagrozenia i szanse zewnętrzne . . . . .	13
2.5	Podział zadań . . . . .	14
3.1	Zadania modułów systemu inteligentnego domu . . . . .	17

## Spis załączników

- „SmartHome-dokumentacja-wdrozeniowa-instrukcja.pdf” - Dokumentacja wdrożeniowa oraz instrukcja użytkowania systemu
- „SmartHomeBoardAPI.yaml” - Dokumentacja API serwera na układach Raspberry Pi
- „SmartHomeBackendAPI.yaml” - Dokumentacja API serwera w chmurze
- „Smart\_Home\_App.fig” - Zapis schematów widoków aplikacji z programu Figma
- „SmartHomeApp-main.zip” - Kopia repozytorium kodu aplikacji mobilnej
- „SmartHomeBackend-main.zip” - Kopia repozytorium kodu serwera chmurowego
- „RPi-main.zip” - Kopia repozytorium kodu serwera na systemie wbudowanym