

WOJSKOWA AKADEMIA
TECHNICZNA
im. Jarosława Dąbrowskiego

Wydział Cybernetyki



PRACA DYPLOMOWA
Studia stacjonarne II^o

**Temat: MODUŁ PREZENTACJI INFORMACJI Z URZĄDZEŃ
INTERNETU RZECZY Z WYKORZYSTANIEM ROZSZERZONEJ
RZECZYWISTOŚCI**

Autor
sierż. pchor. inż. Jakub Zając

Promotor
dr inż. Dariusz Pierzchała

Warszawa 2019

OŚWIADCZENIE

„Wyrażam zgodę na udostępnianie mojej pracy przez Archiwum WAT”.

Dnia

.....

(podpis)

Pracę przyjąłem

promotor pracy
dr inż. Dariusz Pierzchała

Spis treści

1 Wstęp	7
2 Cel i zakres pracy	9
2.1 Cel	9
2.2 Zakres pracy	9
3 Technologia Rozszerzonej Rzeczywistości	10
3.1 Rys historyczny	10
3.2 Podział ze względu na wykorzystywanaą technologię	13
3.3 Podział ze względu na wyświetlanie treści	16
4 Rozszerzona Rzeczywistość w zastosowaniu IoT	20
4.1 Koncepcja Internetu Rzeczy	20
4.2 Przykłady wykorzystania Internetu Rzeczy	22
4.3 Rozszerzona Rzeczywistość, a Internet Rzeczy	24
5 Projekt techniczny	25
5.1 Wybór wykorzystywanej technologii	25
5.2 Wymagania funkcjonalne i poza funkcjonalne	27
5.3 Diagram - aktywności	46
5.4 Architektura systemu	50
5.5 Diagram - klas	54
5.6 Diagram - sekwencji	63
5.7 Diagram - wdrożenia	70
6 Implementacja oraz środowisko wytwarzcze	71
6.1 Wykorzystane technologie oraz narzędzia	71
6.2 Kluczowe aspekty programistyczne	73
6.3 Implementacja oprogramowania - urządzenie mobilne	77

7 Testy	84
7.1 Autoryzacja -logowanie	84
7.2 Autoryzacja - rejestracja nowego użytkownika	89
7.3 Komponent mapowy	92
7.4 Wyszukanie urządzeń IoT Bluetooth	95
7.5 Widok główny	97
7.6 Widok AR	99
8 Zakończenie	103
Spis rysunków	105
Spis tabel	107
Spis listingów kodu	108
Bibliografia	109

Rozdział 1

Wstęp

Liczba urządzeń podłączonych do Internetu każdego dnia rośnie, według raportu Global Digital z 2018 roku ponad połowa populacji świata używa Internetu [1]. Ponad 3 miliardy ludzi korzysta z mediów społecznościowych każdego dnia [1]. Informatyzacja coraz większej części naszego życia sprawia, że pojawiają się coraz to nowe gałęzie informatyki i ich pochodnych dziedzin. Część z nich ma ułatwić codzienne czynności, skrócić czas ich wykonywania, zwiększyć dokładność i bezpieczeństwo. Jedną z takich dziedzin jest Internet Rzeczy (Internet of Thing, IoT), czyli koncepcja wg której urządzenia z różnych dziedzin naszego życia (medycyna, przemysł, militaria, itp.) podłączone są do sieci i wymieniają ze sobą informacje, usprawniając tym samym nasze życie [2; 3]. Liczba urządzeń IoT podłączonych do sieci stale się zwiększa, z prognoz wynika iż do 2020 roku liczba ta urośnie z obecnych 23,14 miliardów do 30,73 miliardów urządzeń [4]. Kolejną dziedziną informatyki, która usprawnia nasze codzienne funkcjonowanie, ułatwia wykonywanie zawodu, czy też pozwala na naukę teoretycznych rzeczy w sposób praktyczny, jest Rozszerzona Rzeczywistość (Augmented Reality) [5]. Rozszerzona Rzeczywistość jest to ulepszona wersja fizycznej rzeczywistości, na którą nadkładane są elementy wygenerowane przez komputer, w postaci dźwięku lub grafiki [6, str. 2]. W swojej pracy zaprezentuję koncepcję reprezentacji danych pochodzących z urządzeń IoT oraz interakcję z nimi przy wykorzystaniu Rozszerzonej Rzeczywistości. Połączenie tych dwóch technologii sprawi, iż dane będą reprezentowane w sposób bardziej czytelny i intuicyjny dla użytkowania.

Jako pierwszy punkt pracy przedstawiony został cel i zakres pracy. Następnie zaprezentowano wykorzystanie Rozszerzonej Rzeczywistości w zastosowaniach koncepcji Internetu Rzeczy. Jako kolejny punkt dokonano przeglądu i wyboru dostępnych technologii, zarówno w kontekście Rozszerzonej Rzeczywistości, jak i komunikacji z urządzeniami Internetu Rzeczy. Główną częścią pracy jest projekt i implementacja systemu informatycznego, którego zadaniem jest

prezentacja informacji gromadzonych z urządzeń IoT z wykorzystaniem Rozszerzonej Rzeczywistości. W końcowej części zaprezentowane zostały testy systemu oraz napotkane trudności.

Rozdział 2

Cel i zakres pracy

2.1 Cel

Zasadniczym celem pracy jest zaprojektowanie, implementacja oraz przetestowanie systemu informatycznego pozwalającego na prezentowanie danych z urządzeń IoT przy wykorzystaniu koncepcji Rozszerzonej Rzeczywistości. W tym celu dokonałem przeglądu i wyboru technologii pozwalającej na użycie Rozszerzonej Rzeczywistości oraz przedstawiłem jej rodzaje ze względu na sposób prezentowania, wyświetlania informacji. Dodatkowo zaprezentowane zostały rodzaje komunikacji między urządzeniami IoT, a urządzeniami pozwalającymi na wizualizację otrzymanych danych.

2.2 Zakres pracy

Zakres pracy obejmuje realizację poniższych zadań:

1. Przedstawienie zagadnienia Internetu Rzeczy.
2. Przedstawienie technologii Rozszerzonej Rzeczywistości.
3. Opracowanie metody gromadzenia i przetrzymywania danych z urządzeń IoT na potrzeby prezentacji w Rozszerzonej Rzeczywistości.
4. Projekt aplikacji mobilnej prezentującej infomacje z wykorzystaniem Rozszerzonej Rzeczywistości w oparciu o odczyty z urządzeń IoT.
5. Implementacja aplikacji oraz testy w ramach wybranego Case Study.

Rozdział 3

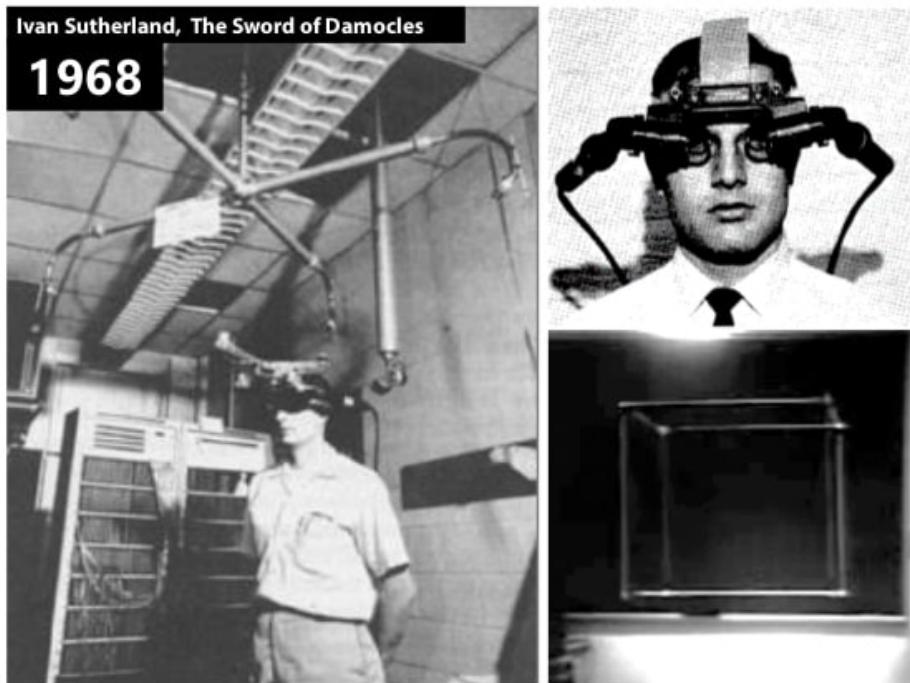
Technologia Rozszerzonej Rzeczywistości

Rozdział ten zawiera krótką charakterystykę oraz historię Rozszerzonej Rzeczywistości. W celu usystematyzowania szeroko pojętej technologii AR, dokonałem podziału ze względu rodzaj prezentacji danych oraz urządzeń, które ją obsługują.

3.1 Rys historyczny

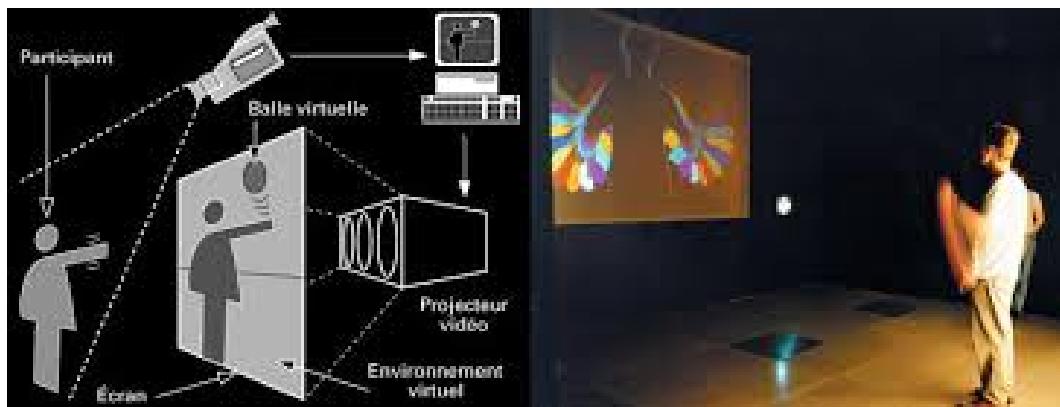
Pomimo ogólnego przekonania, iż technologia Rozszerzonej Rzeczywistości to wynalazek XXI wieku, to już w 1968 roku amerykański inżynier Ivan Sutherland opracował pierwsze urządzenie AR [7]. Dzięki nagłownemu wyświetlaczu, który z racji swojego ciężaru zwisał z sufitu, zobaczyć można było proste szkielety figur geometrycznych (Rys. 1). Kolejnym ważnym krokiem, który przyczynił się do rozwoju Rozszerzonej i Wirtualnej Rzeczywistości, był projekt Myrona Kreugera „Videoplace” [8]. Polegał on na wykorzystaniu projektorów, kamer video oraz oprogramowania, które śledziło ruch osób znajdujących się w specjalnie przystosowanym do tego pomieszczeniu, a następnie umieszczenie ich sylwetek w interaktywnym środowisku [9] (Rys. 2). Uczestnicy projektu mieli dodatkowo możliwość interakcji poprzez gesty z generowanym środowiskiem, którego odzwierciedleniem było wyświetlanie różnych, kolorowych kształtów [9].

Rysunek 1: Urządzenie AR opracowane przez Ivana Sutherlanda



<https://www.slideshare.net/YutaItoh1/introduction-to-optical-seethrough-hmds-in-ar>

Rysunek 2: Projekt "Videoplace"



<https://repository.library.teimes.gr>

Pomimo prowadzonych badań i projektów oficjalne nazewnictwo „Rozszerzona Rzeczywistość” zostało wprowadzone dopiero w 1990 roku przez Tom Caudell'a, amerykańskiego pracownika firmy Boeing [10]. Opisać chciał on w ten sposób działanie alternatywnego rozwiązania, ułatwiającego pracę robotników podczas montażu okablowania. Dotychczas dla każdego samolotu, przy pomocy schematów oraz specjalnych maszyn, znakowane były miejsca ułożenia kabli na płycie ze sklejki. Po wdrożeniu projektu, specjalne projektorzy zakładane na głowy pracowników wyświetlały

wcześniej zaprogramowane schematy okablowania na płytach wielokrotnego użytku. Takie rozwiązanie pozwalało ograniczyć ręczną konfigurację każdej z płyt oraz umożliwiło usprawnienie procesu nanoszenia poprawek (Rys. 3). Firma Boeing była pierwszą firmą, która wykorzystywała technologię rozszerzonej rzeczywistości [11].

Rysunek 3: Rozszerzona Rzeczywistość w firme Boeing



<https://www.codingdojo.com/blog/realitys-hyped-augmented-virtual-reality/>

Wraz z postępem technologicznym, również koncepcja Rozszerzonej Rzeczywistości rozwijała się. O ile do tej pory wszystkie wykorzystywane rozwiązania były komercyjne, to w 1999 roku Hirokazu Kato opublikował bibliotekę ARToolKit, z otwartym kodem, dostępną dla wszystkich developerów. Pozwalała ona połączenie świata rzeczywistego z elementami komputerowo wytworzony grafiki 3D przy wykorzystaniu kamery [7]. Wraz z wyparciem tradycyjnych telefonów przez smartphony, technologia Rozszerzonej Rzeczywistości zyskała popularność wśród developerów [12]. Wsparcie czołowych producentów oprogramowania (Google - ARCore oraz Apple – ARKit) umożliwiającego tworzenie aplikacji również implikuje rozwój technologii AR[13].

3.2 Podział i ogólna charakterystyka ze względu na wykorzystywana technologię

Urządzenia wykorzystujące technologię Rozszerzonej Rzeczywistości podzielić można ze względu na sposób działania i typ wykorzystywanego urządzenia [14]. Pierwszy podział określa, w jaki sposób urządzenie rozpoznaje scenerię i otoczenie, a następnie w odpowiednim miejscu wyświetla generowane obiekty. Wydzielić można [15]:

- Marker-based AR – czyli technologia oparta na znacznikach. Urządzenie wykorzystuje wbudowaną kamerę, która śledzi marker, a w jego miejscu wyświetla obiekt 3D. Również na podstawie znacznika określana jest orientacja oraz pozycja obiektu (Rys. 4).
- Markerless AR – technologia wykorzystująca szereg czujników, w jakie wyposażone są urządzenia AR. Są to między innymi: moduł GPS, akcelerometr, żyroskop czy też kompas. Na ich podstawie obliczane jest położenie urządzenia w przestrzeni oraz terenie. Wyświetlane obiekty 3D przypisane mają współrzędne, w jakich mają się znajdować. Jeśli znajdują się w polu widzenia kamery wyświetlane zostają na wyświetlaczu (Rys. 5).
- Projection-based AR – technologia polegająca na projekcji światła syntetycznego na powierzchnie fizyczne. Wykorzystując specjalne wskaźniki lub czujniki użytkownik może oddziaływać na projekcje, zmieniając np. jej kolor. Urządzenia wykorzystujące tą technologię to między innymi hologramy czy też montowane w samolotach lub samochodach HUD (Head-Up Displays) (Rys. 6).
- Superimposition-based AR – czyli wykorzystanie Rozszerzonej Rzeczywistości w celu całkowitego lub częściowego zastąpienia oryginalnego widoku. Rozpoznawanie obiektów, czy też scenerii jest, kluczowym aspektem tej technologii. Przykładowo system operacyjny Android, wykorzystujący bibliotekę ARCore, rozpoznaje scenerię na podstawie kontrastu oraz oświetlenia. Następnie na danej powierzchni umieszcza punkty (feature points), które śledzi i oblicza położenie względem nich, wykorzystując do tego kamerę oraz wbudowane czujniki (Rys. 7).

Poniżej przedstawiam obrazy, które zawierają przykładowe urządzenia i wykorzystywane technologii Rozszerzonej Rzeczywistości.

Rysunek 4: Marker AR



<https://www.realitytechnologies.com/augmented-reality/>

Rysunek 5: Markerless AR



<https://interaktivegestaltung.net/augmented-reality-for-exhibitions/>

Rysunek 6: Projection-based AR



<https://ljunlimited.com/products/car-hud-display>

Rysunek 7: Superimposition-based AR



<https://www.designnews.com/design-hardware-software/arcore-google-broadens-its-ar-development-scope/177795113657407>

3.3 Podział i ogólna charakterystyka ze względu na wyświetlanie treści

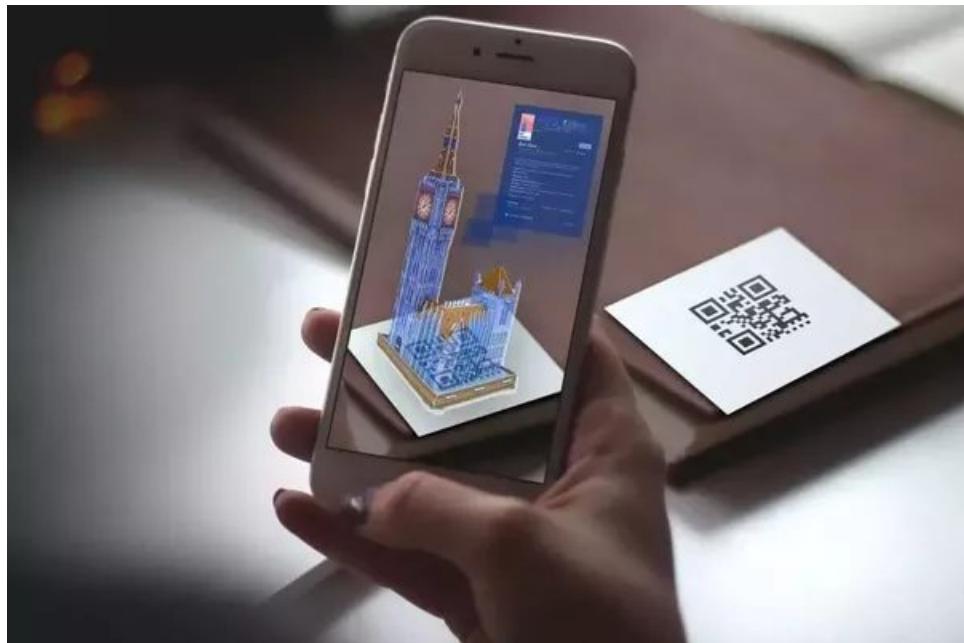
Drugi z podziałów jaki można wyróżnić analizując technologię Rozszerzonej Rzeczywistości, to podział ze względu na rodzaj urządzenia AR [17]:

- Urządzenia mobilne (smartphone i tablety) - Liczba użytkowników urządzeń mobilnych stale rośnie, według ekspertów w 2020 roku ich ilość osiągnie wartość 4,78 miliarda [18]. Dzięki temu z Rozszerzonej Rzeczywistości korzystać może prawie każdy, potrzebna jest jedynie do tego odpowiednia aplikacja. Niekwestionowanie od wykorzystanej technologii grafika nakładana jest na widok z wbudowanej kamery, a pozycja i orientacja w przestrzeni obliczana na podstawie danych z czujników. Czołowi producenci oprogramowania Google i Apple udostępniają open-sourcowe biblioteki, dzięki którym developerzy mogą projektować i budować aplikację, bez potrzeby zagłębiania się w sposób określania położenia urządzenia, czy rozpoznawania scenerii (Rys. 8).
 - Smart glasses (inteligentne okulary) – są to specjalne urządzenia zakładane na głowę, które wyglądają jak przypominają okulary [17]. Na szkle wyświetlane są obrazy, a dzięki zastosowaniu szeregu czujników i kamer, rozpoznawana jest sceneria. Są to specjalistyczne urządzenia, które wymagają specjalnego, nie zawsze darmowego, środowiska do ich programowania. Ponadto wymagania sprzętowe dla środowiska są bardzo wygórowane i podstawowy komputer stacjonarny lub laptop nie gwarantuje możliwość korzystania z niego [19; 20]. Najpopularniejszymi okularami typu smart glasses są: HoloLens, Meta 2 oraz Magic Leap Lightwear (Rys. 9,10).
 - Urządzeni specjalne - to takie, które służą tylko i wyłącznie do wyświetlania najważniejszych danych, tak aby odbiorca mógł skupić się na realizacji głównej czynności. Są one montowane na przykład w samochodach lub samolotach, jako wyświetlacze HUD. Dla przykładu pilot, dzięki zastosowaniu tego typu urządzeń może w szybki sposób, bez rozproszenia, zapoznać się z najważniejszymi odczytami czujników samolotu [21] (Rys. 11). Jest to bardzo droga technologia, która wymaga specjalnego urządzenia do wyświetlania i generowania obrazu, a środowisko służące do wytwarzania oprogramowania nie jest tak dostępne i rozwinięte jak w przypadku urządzeń mobilnych.
 - Ar contact lenses (specjalistyczne szkła kontaktowe) – jest to technologia polegająca na wyświetleniu generowanych obrazów na specjalnym szkle kontaktowym (Rys. 12). Takie rozwiązanie jest dopiero w fazie eksperymentów, ale potencjał w rozwoju Rozszerzonej Rzeczywistości jest ogromny [22]. Główną przewagą takich soczewek jest komfort użytkowania. Nie potrzeba zakładać dużych okularów, czy też używać telefonu aby korzystać z ułatwień Rozszerzonej Rzeczywistości. Nie mniej

jednak jest to technologia dopiero rozwijająca się, która może zrewolucjonizować codzienne życie, począwszy od zbierania informacji o stanie zdrowia użytkownika, poprzez wyświetlanie dodatkowych danych pochodzących na przykład z telefonu komórkowego, aż po całkowite zintegrowanie ich z Internetem.

Poniżej zaprezentuję zdjęcia, które przedstawiają urządzenia wykorzystujące Rozszerzoną Rzeczywistość:

Rysunek 8: Urządzenie mobile wykorzystujące AR



<https://www.techleer.com/articles/331-must-use-augmented-reality-apps-for-iphone-and-android/>

Rysunek 9: Smart glasses - Meta 2



http://techshow.com.au/posts_detailed.php?postid=630

Rysunek 10: Smart glasses - hololens



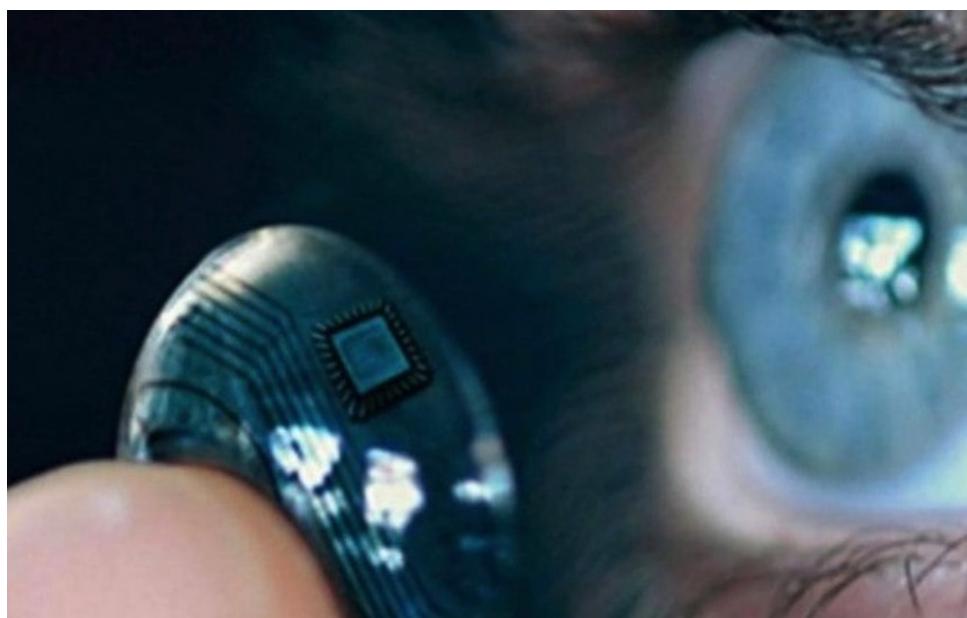
<https://www.microsoft.com/pl-pl/hololens>

Rysunek 11: Kokpit samolotu



<https://uploadvr.com/ar-aviation-safer-better/>

Rysunek 12: Soczewki AR



<https://appleinsider.com/articles/16/10/17/epgl-working-on-iphone-apps-compatible-with-smart-contact-lenses-for-ar>

Rozdział 4

Rozszerzona Rzeczywistość w zastosowaniu IoT

Rozdział ten zawiera koncepcję oraz definicję Internetu Rzeczy, przykłady wykorzystania w różnych aspektach życia człowieka. Następnie zaprezentuję sposoby transmisji danych oraz powiązania Internetu Rzeczy z technologią Rozszerzonej Rzeczywistości.

4.1 Koncepcja Internetu Rzeczy

Terminu „Internet Rzeczy” jako pierwszy użył Kevin Ashton w 1999r, w kontekście wykorzystania technologii RFID (ang. Radio-frequency identification) w zarządzaniu procesem dostaw [25, str. 3]. W opublikowanym artykule Ashton zdefiniował koncepcję Internet Rzeczy jako sieć komputerów, które dzięki zebranym informacjom z czujników, pozwalały śledzić i nadzorować wymianę towarów. Wykorzystanie Internetu Rzeczy ograniczyłoby również straty i koszty. Komputery informowałyby użytkowników o konieczności wymiany lub naprawy urządzeń. Według Ashtona technologia RFID i czujników umożliwia komputerom obserwowanie, identyfikowanie i rozumienie świata bez ograniczeń danych wrodzonych przez człowieka [26, str. 76]. Nie jest to jedyna definicja, którą można znaleźć w artykułach i publikacjach. Z racji ciągłego rozwoju technologicznego, koncepcja Internetu Rzeczy również się zmienia i obejmuje coraz to nowe dziedziny naszego życia. The International Telecommunication Unios definiuje IoT jako sieć obiektów fizycznych lub „rzeczy” wbudowanych w elektronikę, czujniki, oprogramowanie w celu wymiany danych między nimi, przetwarzania i udostępniania [26, str. 76]. Kolejną definicją Internetu Rzeczy, poruszającą obecne aspekty życia codziennego, jest ta zaprezentowana w artykule „*Understanding the Internet of Things: definitione, potentials and societal role of a fast evolving paradigm*”. Internet Rzeczy jest

nowym sposobem rozwiązywania problemów o znaczeniu społecznym. Zaliczają się do nich nowe sposoby kształcenia, postrzeganie domów i miast w skali ludzkiej oraz reagowanie na potrzeby mieszkańców (inteligentne domy i miasta). Internet Rzeczy oferuje również rozwiązania problemów związanych z zarządzaniem energią oraz wspomaganiem opieki zdrowotnej człowieka[27, str. 17]. Jak widać, definicji Internetu Rzeczy jest wiele i każda z nich obejmuje inny sektor, w którym może zostać wykorzystywany. Nie mniej jednak cechą wspólną wszystkich przytoczonych definicji jest wykorzystanie sensorów do gromadzenia i przetwarzania danych z nich w celu ułatwienia codziennego życia.

4.2 Przykłady wykorzystania Internetu Rzeczy

Internet Rzeczy jest obecny praktycznie w każdej dziedzinie, dlatego można przytaczać wiele przykładów jego wykorzystania. W podrozdziale tym przytoczę główne aspekty wykorzystujące IoT:

- Energetyka – przemysł energetyczny wykorzystuje szereg czujników lub dronów do monitorowania stanu technicznego rurociągów i linii energetycznych, dzięki czemu mogą zapobiegać awarią i przerwą w dostawie zasobów. Zbieranie danych, z zamontowanych inteligentnych liczników wody, elektryczności oraz gazu w domach odbiorców, pozwala na analizę zużycia, przewidzenie zapotrzebowanie oraz ewentualną konserwację zapobiegawczą [28].
- Logistyka – wykorzystanie technologii RFID do tagowania zasobów, towarów oraz zastosowanie szeregu czujników pozawała na monitorowanie stanu, lokalizacji i ilości towarów w magazynach. Ponadto firmy transportowe, które obsługują klientów na całym świecie mogą mieć wgląd w czasie rzeczywistym, gdzie dokładnie znajduje się dana przesyłka i jaki jest status. Jedną z firm, która sukcesywnie wprowadza koncepcję Internetu Rzeczy w aspekcie logistyki jest DHL, przedsiębiorstwo specjalizujące się w międzynarodowej dostawie, transporcie i usługach kurierskich [29].
- Inteligentny dom, inteligentne miasto – jest to koncepcja, według której w domach i miastach zainstalowane zostaną czujniki, które tworzyć będą globalna sieć. Sieć ta wymieniać będzie dane miedzy sobą, następnie przetwarzając je i na ich podstawie możliwe będzie między innymi analizowanie natężenie ruchu w mieście, oszczędzanie energii, czy też monitorowanie stanu jakości powietrza. Wszystkie te dane będą agregowane i prezentowane mieszkańcom miasta przy wykorzystaniu na przykład Rozszerzonej Rzeczywistości. Inteligentny dom pozwala na zarządzaniem nim bez konieczności przebywania w takim domu (ustawianie temperatury, włączenie pralki lub oświetlenia) [30, str. 9].
- Medycyna – wykorzystanie licznych sensorów medycznych oraz urządzenia IoT pozwala na wykrycie zagrożenia życia człowieka we wczesnym etapie. Dostępne rozwiązania pozwalają cukrzykom na monitorowanie poziomu cukru we krwi na bieżąco oraz alarmowanie w chwili, gdy jego poziom będzie za niski [31]. Internet Rzeczy pomaga również w codziennym życiu dzięki opaską, które na podstawie tętna i ruchu dloni analizują jakość snu, poziom aktywności oraz mogą przypominać o terminowym dawkowaniu leków.
- Wojsko – Internet Rzeczy jest obecnie wdrażany w wielu wojskowych sektorach: logistyka i transport, dowodzenie (C4ISR), sterowanie kontroli ogniem, trening i symulacja. Armia USA wdrożyła system kontroli przesyłek i dostaw, wykorzystując technologię RFID między głównymi węzłami. Kolejnym przykładem jest symulowanie

pola walki w celach treningowych, które korzysta z potencjału Internetu Rzeczy. Szereg czujników zamontowanych na ciele ćwiczącego oraz w pomieszczeniu pozwala na przeprowadzenie symulacji misji w warunkach kontrolowanych. Ponadto, po odbytym ćwiczeniu, dzięki nagraniom i zgromadzonym danym, możliwa jest jego pełna analiza oraz opracowanie statystyk żołnierzy i ocena wykonania zadania [30, str. 15-16].

4.3 Rozszerzona Rzeczywistość, a Internet Rzeczy

Połączenie technologii Rozszerzonej Rzeczywistości z koncepcją Internetu Rzeczy pozwoliłoby na zobrazowanie danych pochodzących z sieci sensorów jak również interakcję z nimi. Takie rozwiązanie jest wizją przyszłości, nie mniej jednak powstają już produkty integrujące te dwie dziedziny. Przykładem są inteligentne soczewki (smart lens), które dzięki wbudowanym czujnikom mierzą poziom stężenia cukru we krwi pacjenta chorego na cukrzycę. W przypadku obniżenia się poziomu do stanu zagrażającego życiu pacjenta na soczewkach wyświetlane zostaje ostrzeżenie [32]. Dzięki takiemu rozwiązaniu chorujący na cukrzycę nie potrzebuje monitorować poziomu cukru we krwi przy pomocy specjalnych urządzeń, do których potrzebna jest próbka krwi np. z palca. Kolejną dziedziną, w której połączenie AR z IoT ma wielki potencjał, jest inteligentny dom i miasto. Wykorzystanie inteligentnych okularów (smart glass) lub urządzeń mobilnych, które są podłączone do sieci, pozwoli na prezentację danych użytkownikowi, których aktualnie potrzebuje. Wzbogacenie nawigacji samochodowej o dane dotyczące warunków pogodowych, natężenia ruchu czy informacji pochodzących z innych pojazdów, również podłączonych do sieci, ułatwi poruszanie się po mieście oraz poprawi bezpieczeństwo na ulicach [33]. Możliwość filtrowania interesujących użytkownika danych, pozwoli na wyszukiwanie informacji na temat atrakcji turystycznych, sklepów, miejsc kultury i umieszczenie, np. lokalizacji lub godzin otwarta, na urządzeniach AR [33]. Kolejną innowacją, która ma ułatwić komfort pasażerów linii lotniczych, jest aplikacja londyńskiego lotniska Gatwick. Sensory IoT rozmieszczone na terenie całego portu lotniczego, pozwalają lokalizować pasażera, a następnie na urządzeniu mobilnym wyświetlane zostają wskazówki dotyczące odpraw lotniczych, punktów gastronomicznych, parkingów czy też środków transportu publicznego. Aplikacja wyświetla drogę do odpowiedniej bramki, nałożoną na widok z kamery, dzięki czemu odszukanie najkrótszej drogi będzie znacznie łatwiejsze [34]. Jak widać potencjał tych dwóch technologii jest ogromny, a ich połączenie pozwala użytkownikom w intuicyjny i prosty sposób ingerować w świat wirtualny.

Rozdział 5

Projekt techniczny

Celem tego rozdziału jest identyfikacja wymagań systemu. W tym celu określiłem wymagania oraz projekt techniczny w postaci diagramów UML. Ponadto zaprezentowany zostanie stos technologiczny wraz z uzasadnieniem wybranych technologii, które pozwolą zrealizować wymagania.

5.1 Wybór wykorzystywanej technologii

Rozszerzona Rzeczywistość oraz Internet Rzeczy są to technologie, których synergia przede wszystkim ułatwia dostęp do danych i prezentuje je w sposób bliski naturalnemu. W swojej pracy zdecydowałem się na wykorzystanie technologii Rozszerzonej Rzeczywistości do reprezentacji danych pochodzących z urządzeń Internetu Rzeczy. W tym celu zaprojektowałem, zaimplementowałem oraz przetestowałem aplikację na urządzeniu mobilnym, działającym w oparciu o system Android (w wersji od 4.3). Agregować ona będzie dane pochodzące z urządzeń IoT, a następnie wyświetlać na urządzeniu mobilnym. Zdecydowałem się na taki rodzaj urządzenia AR ze względu na fakt iż liczba urządzeń mobilnych na rynku stale rośnie [35], a co za tym idzie praktycznie każdy użytkownik będzie mógł skorzystać z aplikacji bez potrzeby kupowania specjalistycznych urządzeń. Ponadto istnieje możliwość pobrania darmowych, dedykowanych środowisk programistycznych służących do pisania aplikacji mobilnych, takich jak Eclipse, Android Studio czy też NetBeans IDE [36]. Dodatkowym czynnikiem, który przyczynił się do wyboru urządzeń mobilnych posiadających system Android, był fakt, iż posiadają one moduł Bluetooth Low Energy, który pozwala na transmisję danych wykorzystując do tego minimalną ilość energii [37].

Spośród dostępnych technologii AR (opisanych w rozdziale 3) w pracy wykorzystane zostały urządzenia markerless, z którymi użytkownik będzie się łączył za pomocą wyżej wspomnianego modułu Bluetooth Low Energy. Następnie urządzenia IoT będą

wysyłały odczyty ze swoich czujników oraz informację o swoim położeniu w przestrzeni. Urządzenie mobilne na podstawie otrzymanych danych oraz wbudowanych czujników obrazować będzie sensory w postaci widoku AR.

Rolę urządzenia IoT pełnić będą moduły Arduino Genuino 101 [39], do których można podłączyć różne rodzaje sensorów. Mogą do być na przykład czujniki temperatury, dymu, ruchu czy też natężenia światła [38]. Ponadto urządzenia te posiadają moduły komunikacji Bluetooth Low Energy, dzięki któremu będą mogły nawiązywać nisko energetyczne połączenie z urządzeniem użytkownika [39]. Do wybranego modułu Arduino dostępne jest darmowe, dedykowane środowisko programistyczne Arduino IDE [40]. Pozwala ono między innymi na pisanie kodu, przeprowadzanie testów, dołączanie dodatkowych bibliotek oraz wgrywanie programu do pamięci urządzenia.

Ponadto urządzenie mobilne będzie zasilane danymi pochodząymi z serwera, który agreguje dane z urządzeń IoT. Dzięki takiemu rozwiązaniu nawet odległe odczyty z sensorów będą mogły być wizualizowane na urządzeniu mobilnym. Rolę serwera pełnić będzie aplikacja oparta na framework'u Spring Boot, który zapewnia podstawową konfigurację oraz dostarcza serwer aplikacji z rodziny Apache Software Foundation - Tomcat 7 [41]. Ponadto serwer pełnić będzie dodatkową funkcję, a mianowicie dostarczać będzie mechanizm autoryzacji użytkowników. W tym celu posłużę się mechanizmem JWT (Json Web Token). Token zapisany w formacie Json zostaje zaszyfrowany po stronie serwera, który jako jedyny posiada klucz, pozwalający na zweryfikowanie jego autentyczności. Każde zapytanie kierowane do serwera zawiera Token i tylko te, które zawierają poprawny Token, są wykonywane.

5.2 Wymagania funkcjonalne i poza funkcjonalne

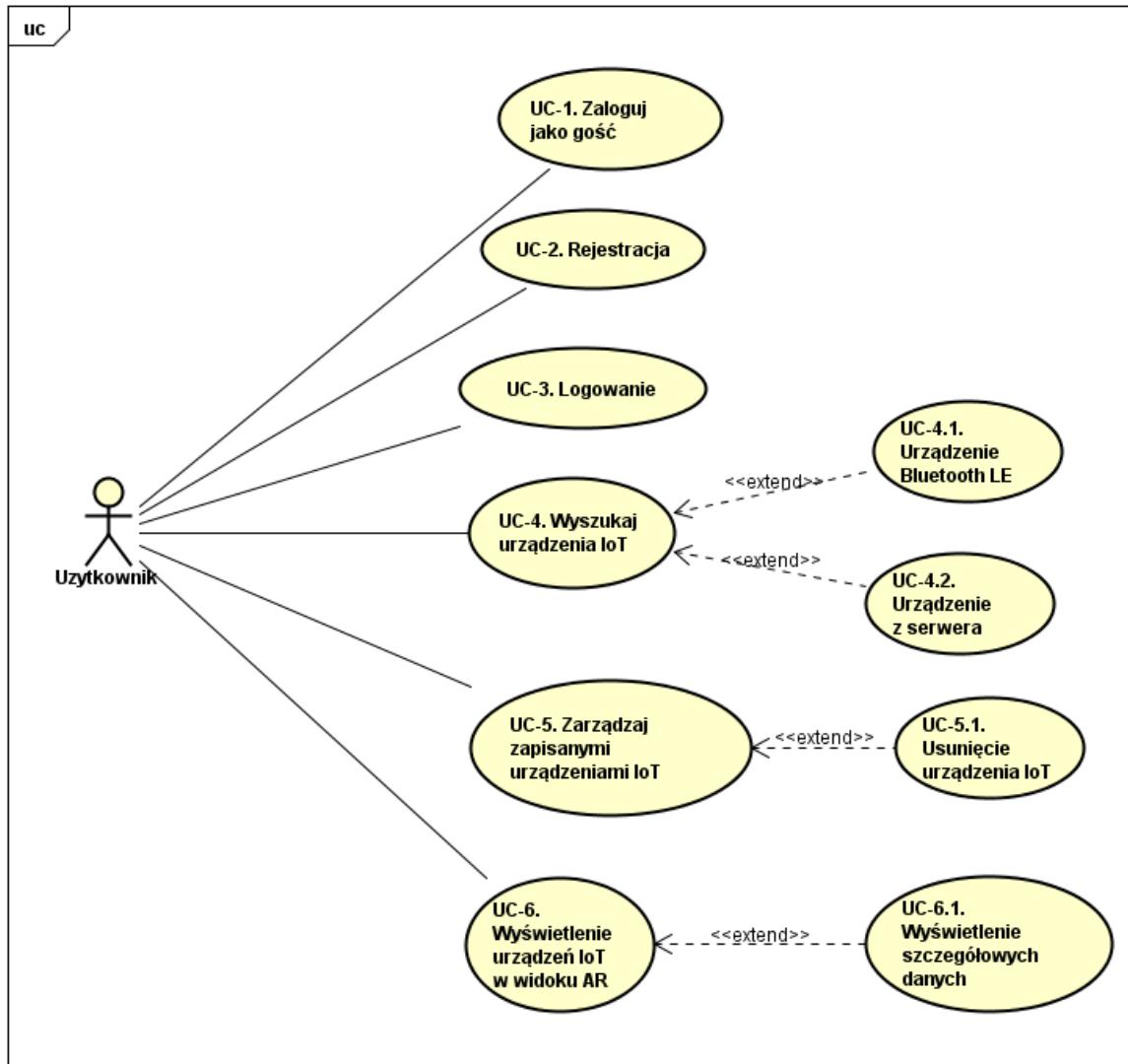
Głównym celem, a zarazem zadaniem projektowanego systemu, jest umożliwienie użytkownikowi wizualizacji odczytów z czujników urządzeń Internetu Rzeczy wykorzystując do tego Rozszerzoną Rzeczywistość. Z racji tego, iż jest to złożona czynność, wymagająca przygotowania urządzenia, postanowiłem podzielić ją na podzadania. Zidentyfikowane wymagania zostały zebrane w postaci przypadków użycia wraz ze scenariuszami ich realizacji.

Zidentyfikowani aktorzy, to:

- Urządzenie mobilne,
- Serwer,
- Urządzenie IoT Bluetooth,
- Urządzenie IoT Internet.

Pierwszy diagram przypadków użycia został zaprojektowany dla Urządzenia Mobilnego.

Rysunek 13: Diagram przypadków użycia - Urządzenie Mobilne



Źródło własne

Opis przypadków użycia dla powyższego diagramu zawarłem w tabelach poniżej.

Tabela 5.1: Scenariusz UC-1.

Nazwa	Zaloguj jako gość.
Identyfikator	UC-1.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Włączenie aplikacji.
Warunki początkowe	Użytkownik zarejestrowany lub nie.
Wynik pozytywnego zakończenia	Wyświetlenie widoku głównego jako gość. Komponent mapowy niedostępny.
Wynik negatywnego zakończenia	Brak.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję działania aplikacji bez autoryzacji - logowanie jako gość. 2. Wyświetlenie widoku głównego.
Przebieg alternatywny przypadku użycia	Brak.

Źródło własne

Tabela 5.2: Scenariusz UC-2.

Nazwa	Rejestracja.
Identyfikator	UC-2.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Pierwsze włączenie aplikacji lub wylogowanie się użytkownika.
Warunki początkowe	Użytkownik niezarejestrowany
Wynik pozytywnego zakończenia	Prawidłowa rejestracja użytkownika, uzyskanie Tokena.

Wynik negatywnego zakończenia	Brak autoryzacji użytkownika, wyświetlenie komunikatu o braku połączenia z Internetem, nieprawidłowego formatu danych lub istnieniu użytkownika o takiej nazwie.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik podaje nazwę użytkownika, hasło oraz kod PIN, którym będzie się logował. 2. Podane dane są wstępnie weryfikowane na urządzeniu mobilnym - długość hasła (min. 5 znaków) oraz kodu PIN (min. 5 liczb). 3. Poprawne dane wysyłane są na serwer w celu ich weryfikacji. Po stwierdzeniu ich poprawności użytkownik w ramach odpowiedzi otrzymuje Token. 4. Zapisanie zaszyfrowanych danych użytkownika. 5. Aplikacja przechodzi o widoku głównego.
Przebieg alternatywny przypadku użycia - 1	<ol style="list-style-type: none"> 1. Użytkownik podaje nazwę użytkownika, hasło oraz PIN, którym będzie się logował. 2. Długość hasła lub kodu PIN jest niepoprawna, wyświetlony zostaje komunikat o błędzie "Błędne dane". Wynik walidacji danych negatywny.
Przebieg alternatywny przypadku użycia - 2	<ol style="list-style-type: none"> 1. Użytkownik podaje nazwę użytkownika, hasło oraz PIN, którym będzie się logował. 2. Brak dostępu do Internetu, wyświetlony zostaje komunikat o błędzie "Sprawdź połączenie z Internetem". Wynik rejestracji negatywny.
Przebieg alternatywny przypadku użycia - 3	<ol style="list-style-type: none"> 3. Użytkownik podaje nazwę użytkownika, hasło oraz PIN, którym będzie się logował. 2. Podane dane są wstępnie weryfikowane na urządzeniu mobilnym - długość hasła (min. 5 znaków) oraz kodu PIN (min. 5 liczb). 3. Poprawne dane wysyłane są na serwer w celu ich weryfikacji. Użytkownik o takiej nazwie już jest zarejestrowany, wyświetlony zostaje komunikat "Nazwa użytkownika jest już zajęta".

Źródło własne

Tabela 5.3: Scenariusz UC-3.

Nazwa	Logowanie.
Identyfikator	UC-3.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Włączenie aplikacji po uprzednim zarejestrowaniu się.
Warunki początkowe	Użytkownik zarejestrowany.
Wynik pozytywnego zakończenia	Prawidłowe zalogowanie użytkownika, uzyskanie Tokena.
Wynik negatywnego zakończenia	Brak autoryzacji użytkownika, wyświetlenie komunikatu o braku połączenia z Internetem lub błędного kodu PIN.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik wpisuje kod PIN. 2. Kod PIN jest porównywany z zaszyfrowanym kodem PIN. 3. Zgodność kodów PIN powoduje zalogowanie się do serwera oraz otrzymanie Tokena. 4. Aplikacja przechodzi o widoku głównego.
Przebieg alternatywny przypadku użycia - 1	<ol style="list-style-type: none"> 1. Użytkownik wpisuje kod PIN. 2. Kod PIN nie jest zgodny z zaszyfrowanym kodem PIN, wyświetlony zostaje komunikat "Błędny PIN".
Przebieg alternatywny przypadku użycia - 2	<ol style="list-style-type: none"> 1. Użytkownik wpisuje kod PIN. 2. Brak dostępu do Internetu, wyświetlony zostaje komunikat o błędzie "Sprawdź połączenie z Internetem".

Źródło własne

Tabela 5.4: Scenariusz UC-4.

Nazwa	Wyszukanie urządzeń IoT.
Identyfikator	UC-4.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Włączenie skanera bluetooth.

Warunki początkowe	Użytkownik zalogowany, moduł bluetooth włączony.
Wynik pozytywnego zakończenia	Dodanie do listy urządzeń znalezionych urządzeń bluetooth.
Wynik negatywnego zakończenia	Wyświetlenie komunikatu z prośbą o włączenie modułu bluetooth.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik włącza skaner urządzeń bluetooth. 2. Znalezione urządzenia zostają dodane do listy. 3. Użytkownik wybiera urządzenia bluetooth, z którymi chce nawiązać połączenie. 4. Zapisanie wybranych urządzeń.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik włącza skaner urządzeń bluetooth. 2. Wyświetlony zostaje komunikat z prośbą o włączenie modułu bluetooth.

Źródło własne

Tabela 5.5: Scenariusz UC-4.1.

Nazwa	Wyszukanie urządzeń IoT - Urządzenie Bluetooth LE.
Identyfikator	UC-4.1.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Włączenie skanera bluetooth.
Warunki początkowe	Użytkownik zalogowany, moduł bluetooth włączony.
Wynik pozytywnego zakończenia	Dodanie do listy urządzeń znalezionych urządzeń bluetooth.
Wynik negatywnego zakończenia	Wyświetlenie komunikatu z prośbą o włączenie modułu bluetooth.

Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik włącza skaner urządzeń bluetooth. 2. Znalezione urządzenia zostają dodane do listy. 3. Użytkownik wybiera urządzenia bluetooth, z którymi chce nawiązać połaczenie. 4. Zapisanie wybranych urządzeń.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik włącza skaner urządzeń bluetooth. 2. Wyświetlony zostaje komunikat z prośbą o włączenie modułu bluetooth.

Źródło własne

Tabela 5.6: Scenariusz UC-4.2.

Nazwa	Wyszukanie urządzeń IoT - Urządzenie z serwera.
Identyfikator	UC-4.2.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Włączenie widoku mapy.
Warunki początkowe	Użytkownik zalogowany.
Wynik pozytywnego zakończenia	Pobranie danych o sensorach z serwera.
Wynik negatywnego zakończenia	Wyświetlenie komunikatu o brak dostępu do Internetu lub błędu serwera.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik włącza widok mapy. 2. Użytkownik definiuje obszar, z którego będą pobierane dane z urządzeń IoT. 3. Dane o urządzeniach IoT zostają pobrane z serwera. 4. Zapisanie urządzeń znajdujących się w obszarze.
Przebieg alternatywny przypadku użycia - 1.	<ol style="list-style-type: none"> 1. Użytkownik włącza widok mapy. 2. Użytkownik definiuje obszar, z którego będą pobierane dane z urządzeń IoT. 3. Wyświetlony zostaje komunikat o błędzie serwera.

Przebieg alternatywny przypadku użycia - 2.	<ol style="list-style-type: none"> 1. Użytkownik włącza widok mapy. 2. Użytkownik definiuje obszar, z którego będą pobierane dane z urządzeń IoT. 3. Wyświetlony zostaje komunikat o braku dostępu do Internetu.
---	---

Źródło własne

Tabela 5.7: Scenariusz UC-5.1.

Nazwa	Zarządzanie zapisanymi urządzeniami IoT - usunięcie urządzenia IoT.
Identyfikator	UC-5.1.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Dłuższe naciśnięcie urządzenia IoT z listy.
Warunki początkowe	Urządzenia IoT znajdują się na liście.
Wynik pozytywnego zakończenia	Usunięcie urządzenia IoT z listy.
Wynik negatywnego zakończenia	Urządzenie IoT pozostaje na liście zapisanych urządzeń.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik naciska przez dłuższy czas urządzenie IoT z listy. 2. Wyświetlenie okna potwierdzenia usunięcia urządzenia IoT. 3. Użytkownik potwierdza chęć usunięcia urządzenia IoT. 4. Urządzenie IoT zostaje usunięte.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik naciska przez dłuższy czas urządzenie IoT z listy. 2. Wyświetlenie okna potwierdzenia usunięcia urządzenia IoT. 3. Użytkownik anuluje proces usunięcia urządzenia IoT. 4. Urządzenie IoT pozostaje na liście.

Źródło własne

Tabela 5.8: Scenariusz UC-6.

Nazwa	Wyświetlenie urządzeń IoT w widoku AR.
Identyfikator	UC-6.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Włączenie widoku AR.
Warunki początkowe	Urządzenie skalibrowane.
Wynik pozytywnego zakończenia	Wyświetlenie urządzeń IoT.
Wynik negatywnego zakończenia	Brak.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Użytkownik włącza widok AR, przy skalibrowanym urządzeniu. 2. Dane z sensorów określają lokalizację urządzenia mobilnego w przestrzeni i na bieżąco aktualizują widok AR. 3. Zapisane urządzenia IoT zostają dodane do widoku AR. 4. Zmiany statusu połączenia urządzeń IoT reprezentowane są graficznie w widoku. 5. Odczyty z sensorów IoT są cyklicznie aktualizowane.
Przebieg alternatywny przypadku użycia	Brak.

Źródło własne

Tabela 5.9: Scenariusz UC-6.1.

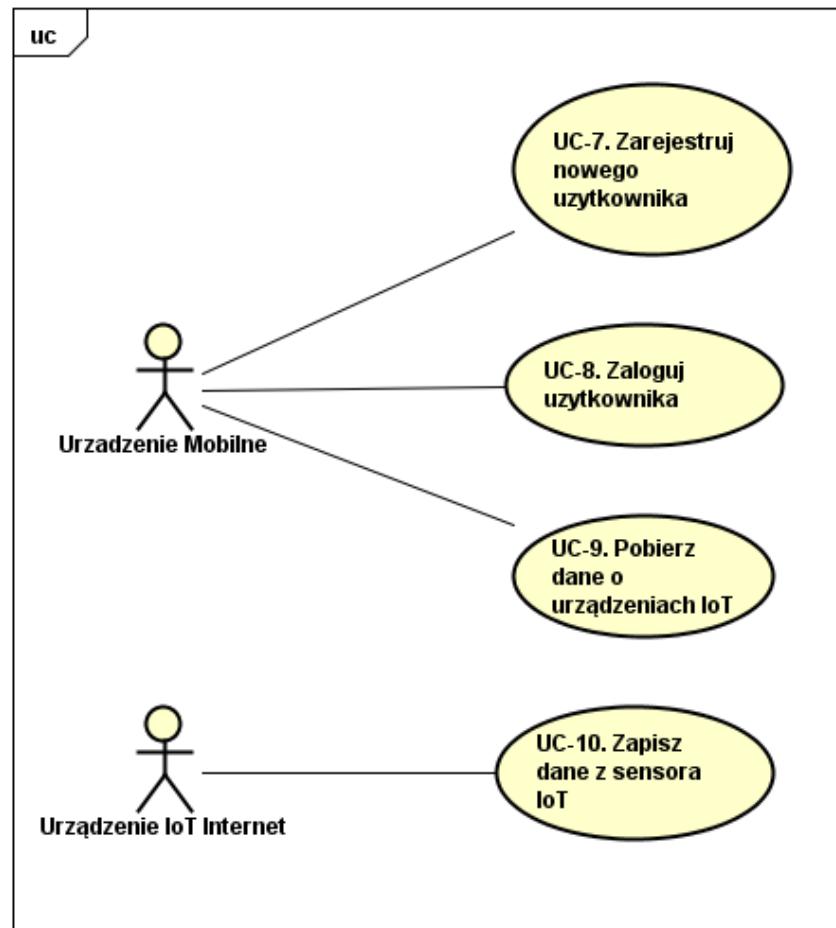
Nazwa	Wyświetlenie szczegółowych danych.
Identyfikator	UC-6.1.
Aktor	Użytkownik aplikacji.
Zdarzenie inicjujące	Dotknięcie wyświetlanego urządzenia IoT w widoku AR.
Warunki początkowe	Urządzenie IoT znajduje się w aktualnym widoku AR, urządzenie mobilne skalibrowane.

Wynik pozytywnego zakończenia	Wyświetlenie danych z urządzenia IoT.
Wynik negatywnego zakończenia	Brak.
Przebieg przypadku użycia	1. Użytkownik dotyka wybranego urządzenia IoT. 2. Szczegółowe dane urządzenia IoT zostają wyświetlane w widoku AR.
Przebieg alternatywny przypadku użycia	Brak.

Źródło własne

Kolejny diagram przypadków użycia został zaprojektowany dla Serwera.

Rysunek 14: Diagram przypadków użycia - Serwer



Źródło własne

Opis przypadków użycia dla powyższego diagramu zawarłem w tabelach poniżej.

Tabela 5.10: Scenariusz UC-7.

Nazwa	Zarejestruj nowego użytkownika.
Identyfikator	UC-7.
Aktor	Aplikacja Mobilna.
Zdarzenie inicjujące	Rozpoczęcie procesu rejestracji nowego użytkownika.
Warunki początkowe	Użytkownik wprowadził nazwę użytkownika i hasło.
Wynik pozytywnego zakończenia	Poprawna rejestracja użytkownika oraz uzyskanie Tokena.
Wynik negatywnego zakończenia	Zwrócenie kodu błędu oraz wyświetlenie komunikatu o błędzie "Nazwa użytkownika jest już zajęta".
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Urządzenie mobilne wysyła zapytanie do serwera zawierające nazwę użytkownika i hasło. 2. Serwer weryfikuje dane oraz zwraca Token. 3. Zapisanie zaszyfrowanych danych użytkownika. 4. Aplikacja przechodzi do widoku głównego.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Urządzenie mobilne wysyła zapytanie do serwera zawierające nazwę użytkownika i hasło. 2. Serwer weryfikuje dane i zwraca kod błędu - powtórzona nazwa użytkownika. 3. Aplikacja wyświetla komunikat o błędzie "Nazwa użytkownika jest już zajęta".

Źródło własne

Tabela 5.11: Scenariusz UC-8.

Nazwa	Zaloguj użytkownika.
Identyfikator	UC-8.
Aktor	Aplikacja Mobilna.
Zdarzenie inicjujące	Rozpoczęcie procesu logowania użytkownika.

Warunki początkowe	Użytkownik wprowadził kod PIN.
Wynik pozytywnego zakończenia	Poprawne zalogowanie użytkownika oraz uzyskanie Tokena.
Wynik negatywnego zakończenia	Brak.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Urządzenie odszyfrowuje nazwę użytkownika i hasło. 2. Urządzenie mobilne wysyła zapytanie do serwera zawierające nazwę użytkownika i hasło. 2. Serwer weryfikuje dane oraz zwraca Token. 3. Aplikacja przechodzi do widoku głównego.
Przebieg alternatywny przypadku użycia	Brak.

Źródło własne

Tabela 5.12: Scenariusz UC-9.

Nazwa	Pobierz dane o urządzeniach IoT.
Identyfikator	UC-9.
Aktor	Aplikacja Mobilna.
Zdarzenie inicjujące	Czynność wykonywana cyklicznie o zadanym interwale czasowym.
Warunki początkowe	Użytkownik zalogowany.
Wynik pozytywnego zakończenia	Poprawnie pobrane dane o urządzeniach IoT, wyświetlanie odczytów sensorów w aplikacji mobilnej.
Wynik negatywnego zakończenia	Wyświetlenie komunikatu o błędzie: "Błąd połączenia z serwerem".

Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Aplikacja mobilna wysyła zapytanie do serwera podpisane Tokenem użytkownika. 2. Serwer zwraca aktualne dane z sensorów. 3. Aktualne dane zostają wyświetlane w aplikacji mobilnej.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Aplikacja mobilna wysyła zapytanie do serwera podpisane Tokenem użytkownika. 2. Serwer zwraca kod błędu. 3. Aplikacja mobilna wyświetla komunikat o błędzie "Błąd połączenia z serwerem".

Źródło własne

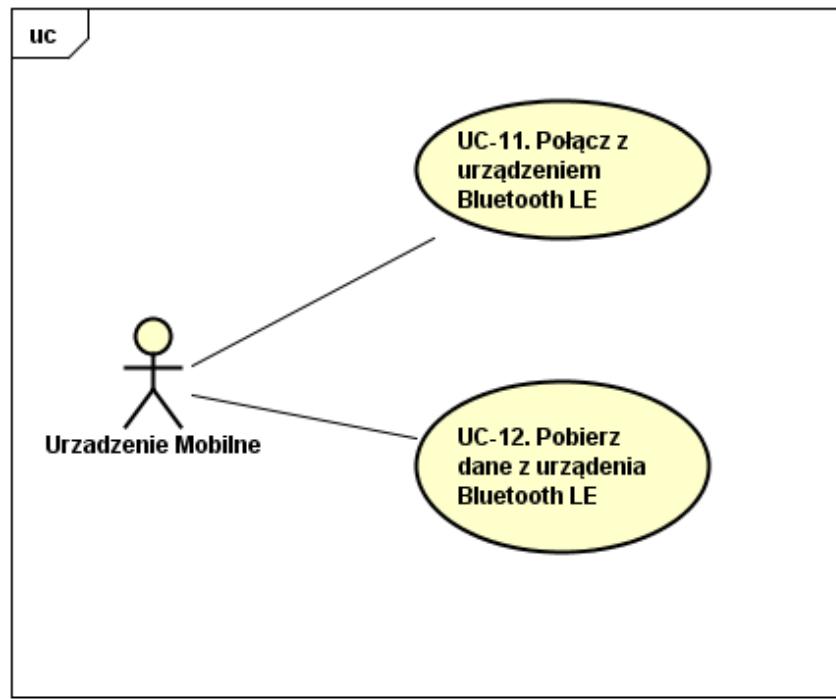
Tabela 5.13: Scenariusz UC-10.

Nazwa	Zapisz dane z sensora IoT.
Identyfikator	UC-10.
Aktor	Urządzenie IoT Internet.
Zdarzenie inicjujące	Wykonanie odczytu z sensoru.
Warunki początkowe	Urządzenie IoT Internet połączone z serwerem.
Wynik pozytywnego zakończenia	Zapisanie aktualnych danych z urządzenia IoT Internet.
Wynik negatywnego zakończenia	Wyświetlenie komunikatu o błędzie: "Błąd połączenia z serwerem".
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1.Urządzenie IoT Internet wysyła dane z aktualnym odczytem sensoru. 2. Serwer zapisuje aktualny odczyt urządzenia IoT Internet.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1.Urządzenie IoT Internet wysyła dane z aktualnym odczytem sensoru. 2. Serwer zwraca kod błędu.

Źródło własne

Poniższy diagram przypadków użycia został zaprojektowany dla urządzenia IoT Bluetooth.

Rysunek 15: Diagram przypadków użycia - Urządzenie IoT Bluetooth



Źródło własne

Opis przypadków użycia dla powyższego diagramu zawarłem w tabelach poniżej.

Tabela 5.14: Opis UC-11.

Nazwa	Połącz z urządzeniem Bluetooth LE.
Identyfikator	UC-11.
Aktor	Aplikacja Mobilna.
Zdarzenie inicjujące	Czynność wykonywana automatycznie w przypadku braku połączenia z zapisanym urządzeniem Bluetooth LE.
Warunki początkowe	Użytkownik zalogowany, moduł bluetooth włączony.
Wynik pozytywnego zakończenia	Nawiązanie połączenia z urządzeniem Bluetooth LE, zmiana statusu połączenia w widoku głównym.
Wynik negatywnego zakończenia	Zmiana statusu połączenia w widoku głównym na rozłączony.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Aplikacja mobilna nawiązuje połączenie się z zapisanym urządzeniem Bluetooth LE. 2. Zmiana status połączenia na połączony w widoku głównym.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Aplikacja mobilna nawiązuje połączenie się z zapisanym urządzeniem Bluetooth LE. 2. Zmiana status połączenia na rozłączony w widoku głównym.

Źródło własne

Tabela 5.15: Scenariusz UC-12.

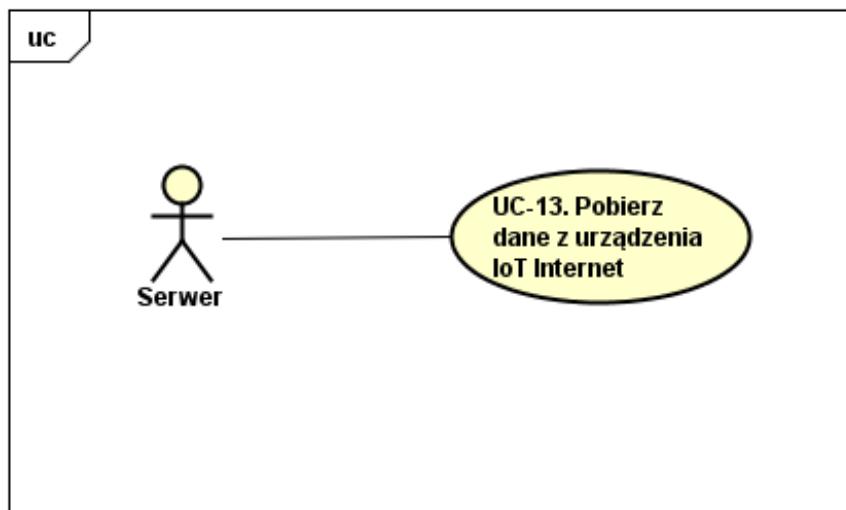
Nazwa	Pobierz dane z urządzenia Bluetooth LE.
Identyfikator	UC-12.
Aktor	Aplikacja Mobilna.
Zdarzenie inicjujące	Czynność wykonywana automatycznie przez urządzenie mobilne w przypadku zmiany danych z urządzenia Bluetooth LE.
Warunki początkowe	Użytkownik zalogowany, moduł bluetooth włączony. Urządzenie mobilne połączone z urządzeniem Bluetooth LE oraz znajdujące się w widoku głównym lub AR.

Wynik pozytywnego zakończenia	Odczyt nowych danych z urządzenie Bluetooth LE.
Wynik negatywnego zakończenia	Brak.
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Aplikacja mobilna otrzymuje notyfikację o zmianie wartości odczytu sensora urządzenia Bluetooth LE. 2. Nowa wartość odczytu z sensora zostaje wyświetlone w ekranie głównym aplikacji lub w widoku AR.
Przebieg alternatywny przypadku użycia	Brak.

Źródło własne

Ostatni diagram przypadków użycia został zaprojektowany dla urządzenia IoT Internet.

Rysunek 16: Diagram przypadków użycia - urządzenie IoT Internet



Źródło własne

Opis przypadków użycia dla powyższego diagramu zawarłem w tabelach poniżej.

Tabela 5.16: Scenariusz UC-13.

Nazwa	Pobierz dane z urządzenia IoT Internet.
Identyfikator	UC-13.
Aktor	Serwer.
Zdarzenie inicjujące	Pobranie odczytu z sensoru.
Warunki początkowe	Urządzenie IoT Internet połączone z serwerem.
Wynik pozytywnego zakończenia	Poprawny odczyt danych z urządzenia IoT Internet.
Wynik negatywnego zakończenia	Wyświetlenie komunikatu o błędzie: "Błąd połączenia z urządzeniem IoT Internet".
Przebieg przypadku użycia	<ol style="list-style-type: none"> 1. Serwer odpytuje o aktualny odczyt z sensora IoT Internet. 2. Urządzenie IoT Internet zwraca dane z aktualnym odczytem sensoru. 3. Serwer zapisuje zwrócony odczyt urządzenia IoT Internet do bazy danych.
Przebieg alternatywny przypadku użycia	<ol style="list-style-type: none"> 1. Serwer odpytuje o aktualny odczyt z sensora IoT Internet. 2. Wyświetlenie komunikatu o błędzie: "Błąd połączenia z urządzeniem IoT Internet"

Źródło własne

Dodatkowo opracowałem wymagania poza funkcjonalne, które projektowany system musi spełnić.

Tabela 5.17: Wymagania poza funkcjonalne.

Nr.	Opis wymagania poza funkcjonalnego
1.	Wykorzystanie komunikacji między urządzeniem służącym do zobrazowania, a urządzeniami IoT, o poborze energii w granicach 0.01–0.50 W.
2.	Możliwość działania systemu bez dostępu do Internetu.
3.	Szyfrowanie danych wrażliwych, które przechowywane będą lokalnie w urządzeniach.
4.	Działanie systemu w tle bez utraty aktualnych danych.
5.	Dostarczenie interfejsu użytkownika, pozwalającego na intuicyjne nawigowanie bez konieczności dostarczenia dodatkowych instrukcji użytkowania.

Źródło własne

5.3 Diagram - aktywności

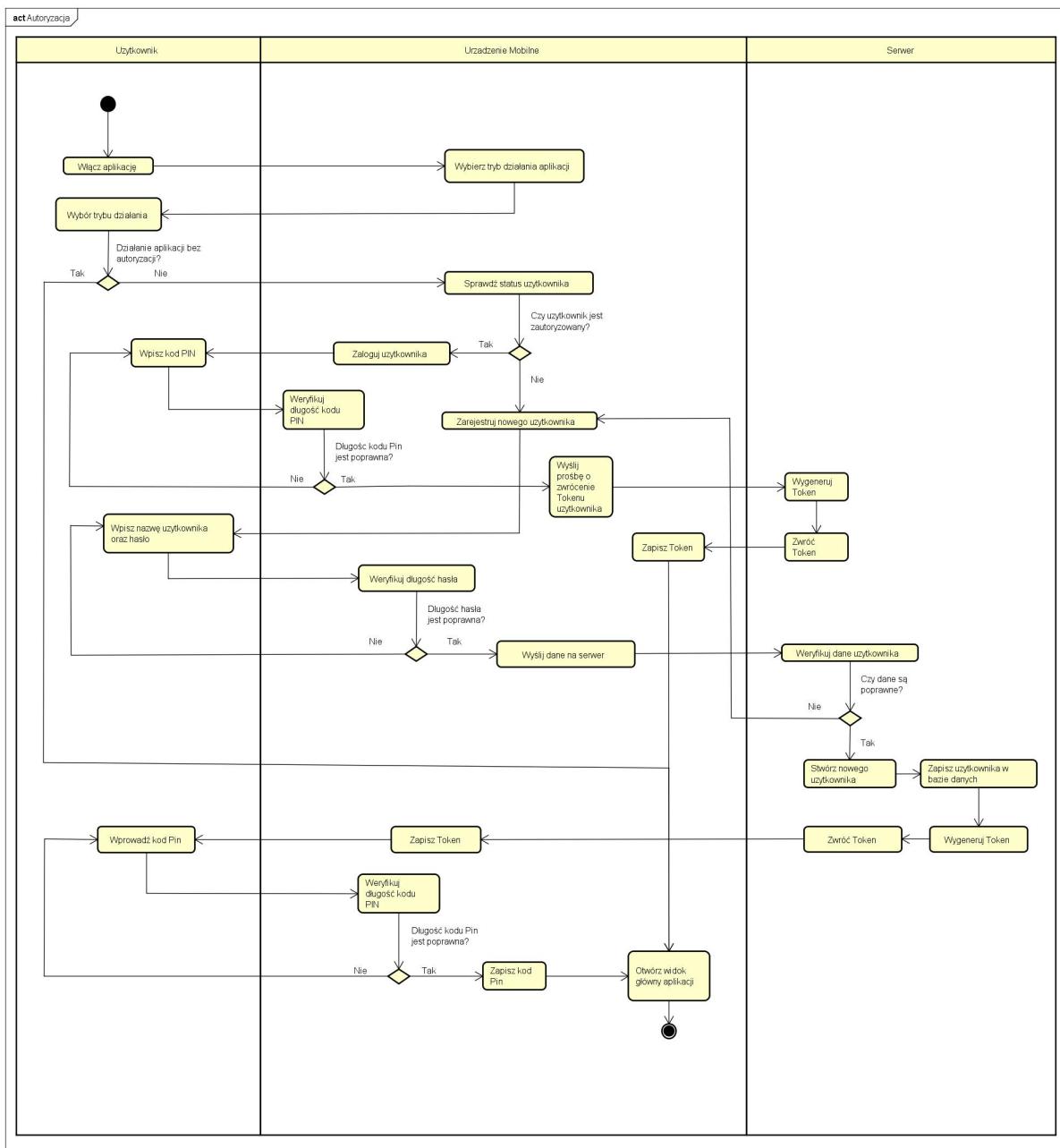
Diagram aktywności służy do modelowania procesów, zdarzeń lub czynności. W przypadku projektowanego systemu podzieliłem jego działanie na trzy główne czynności: autoryzacja, zarządzenie urządzeniami IoT, wyświetlenie widoku AR. Wyróżniłem również na diagramie każdy komponent projektowanego systemu w postaci tzw. swimlane.

Pierwszy z diagramów prezentuje proces autoryzacji (Rys. 17). Użytkownik w pierwszej kolejności wybiera tryb działania aplikacji. Ma do wyboru tryb gościa oraz autoryzowanego użytkownika. W tym pierwszym przypadku aplikacja przechodzi od razu do ekranu głównego. Natomiast w drugim przypadku użytkownik dokonuje procesu rejestracji lub logowania. Proszony jest również o podanie kodu PIN, którym będzie dokonywał procesu logowania oraz zgody na wykorzystywanie autoryzacji poprzez odcisk palca.

Drugi diagram przedstawia proces, w którym użytkownik definiuje które z dostępnych urządzeń Bluetooth oraz tych pochodzących z serwera będą wizualizowane w widoku AR (Rys. 18). Użytkownikowi udostępniany jest skaner urządzeń Bluetooth oraz komponent mapowy, za pośrednictwem którego możliwy jest wybór urządzeń, których odczyty zostaną pobrane z serwera. Widok główny aplikacji zawiera natomiast listę wybranych i zapisanych urządzeń IoT.

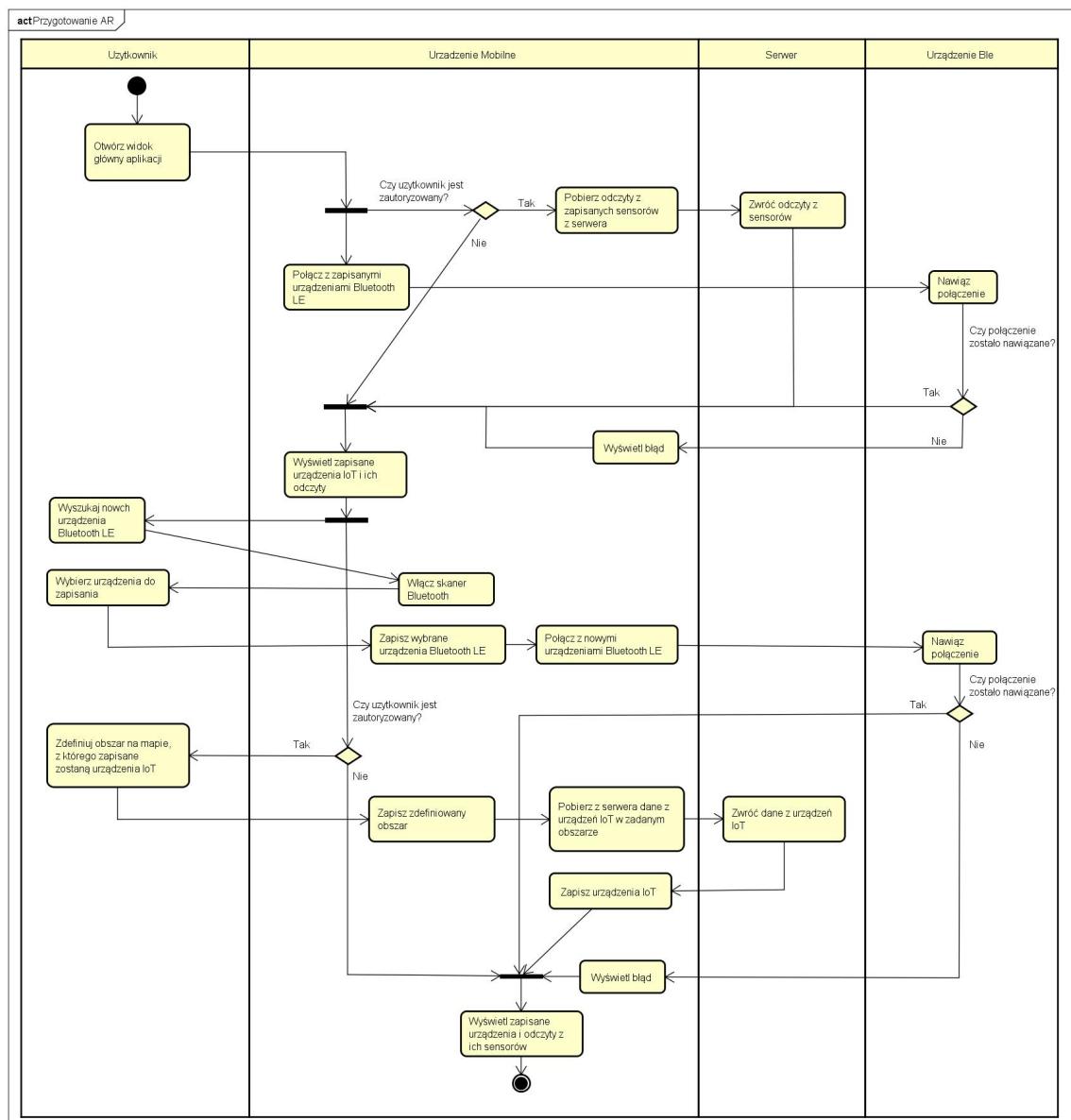
Ostatni diagram zawiera proces przygotowania i wyświetlania widoku Rozszerzonej Rzeczywistości (Rys. 19). Na całość tego procesu składa się wiele czynników, między innymi: aktualizacja lokalizacji użytkownika, orientacji w przestrzeni (azymut oraz pułap), pobranie danych urządzeń IoT Bluetooth oraz z serwera. Za generowanie sceny AR odpowiada widok z wbudowanej kamery oraz openGL ES. Połączenie wszystkich danych pozawala na wyświetlanie widoku Rozszerzonej Rzeczywistości.

Rysunek 17: Diagram aktywności - autoryzacja



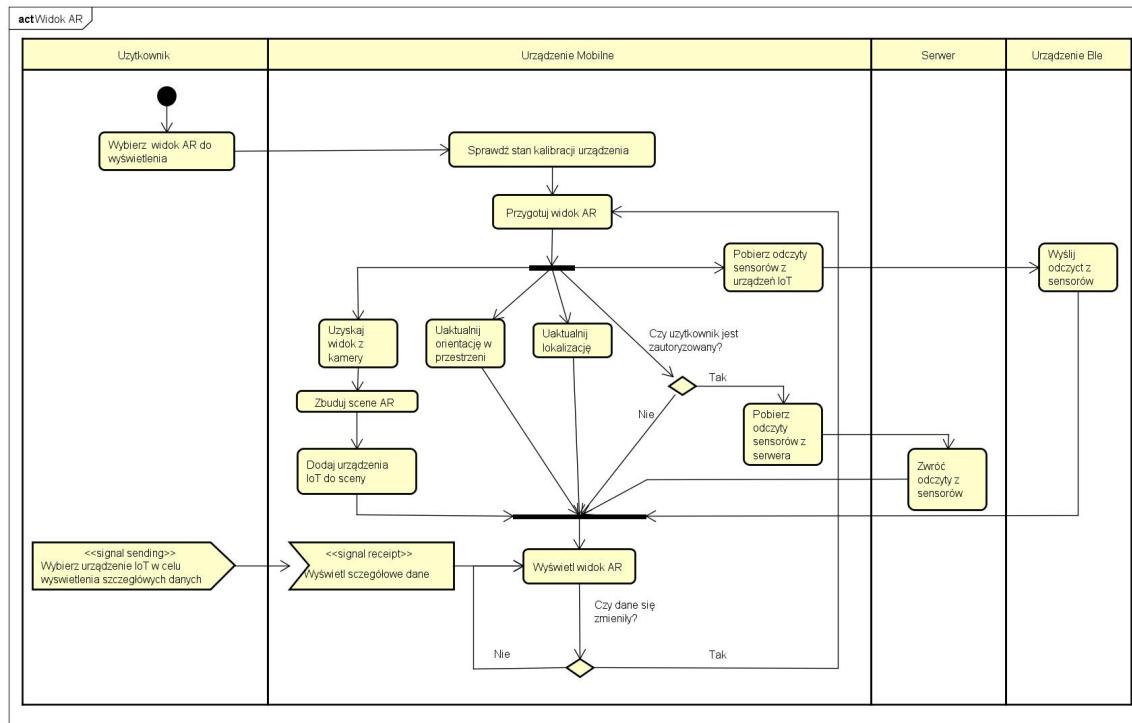
Źródło własne

Rysunek 18: Diagram aktywności - przygotowanie widoku AR



Źródło własne

Rysunek 19: Diagram aktywności - widok AR



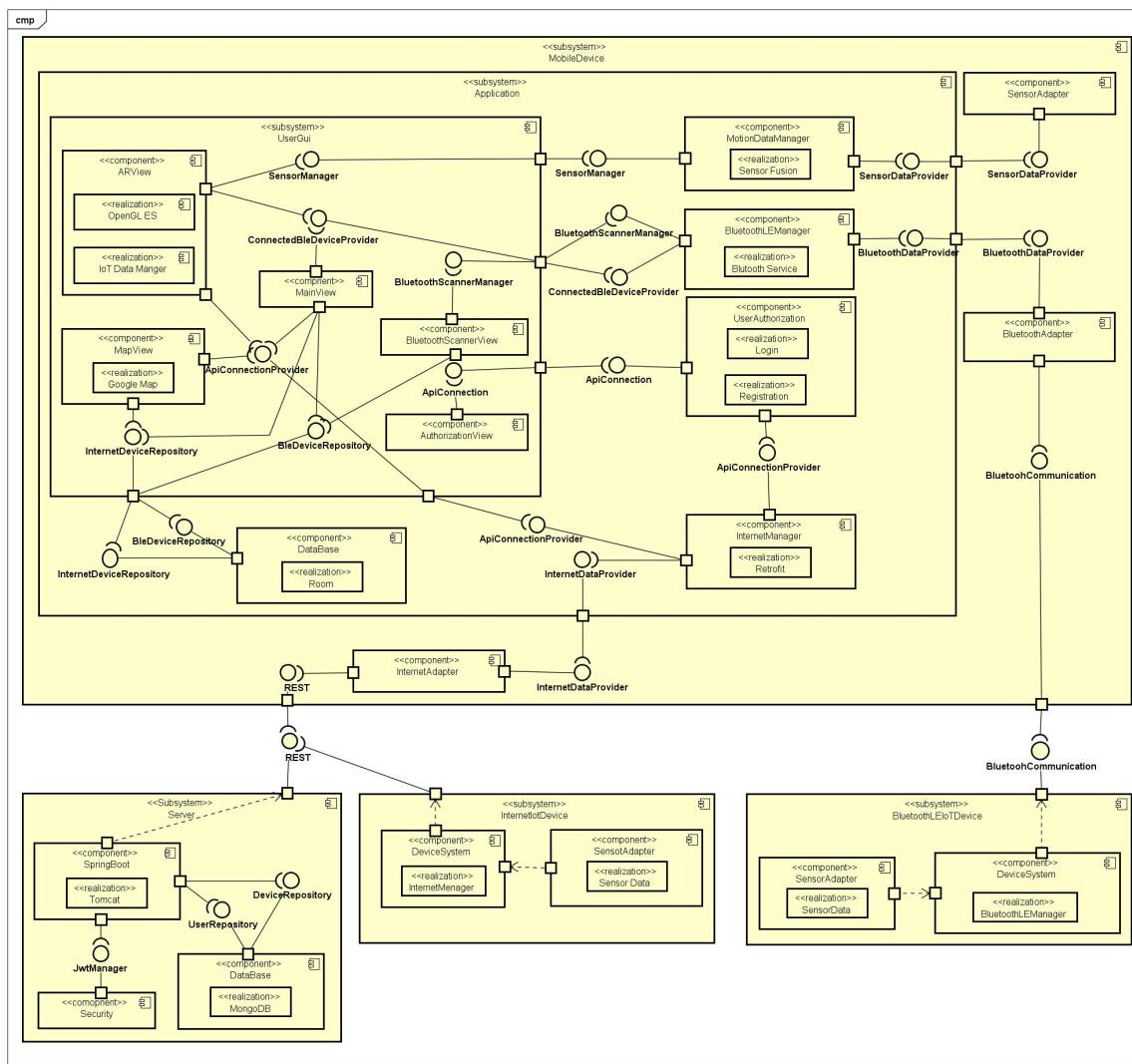
Źródło własne

5.4 Architektura systemu

Projektowany system informatyczny składać się będzie z czterech głównych fizycznych podsystemów:

- urządzenie mobilne,
 - serwer,
 - urządzenie IoT wykorzystujące komunikację Bluetooth LE,
 - urządzenie IoT wykorzystujące komunikację HTTP.

Rysunek 20: Diagram komponentów



Źródło własne

W związku z faktem, iż urządzenie mobilne jest najważniejszą częścią projektowanego systemu, zdecydowałem się na jego szczegółowy opis. Wyodrębniłem następujące podsystemy:

- MobileDevice,

- Application,
- UserGui.

Podsystem "MobileDevice" jest symboliczną reprezentacją urządzenia mobilnego. Posiada on abstrakcyjne komponenty: Adapter, BluetoothAdapter oraz InternetAdapter. Udostępniają one interfejsy: SensorDataProvider, BluetoothDataProvider oraz InternetDataProvider umożliwiające przepływ danych na poziomie Hardware - Aplikacja.

W podsystemie "Application" wyodrębniłem następujące komponenty, które następnie zaimplementowałem:

- MotionDataManger - jest to komponent odpowiadający za określenie pozycji uprzędzenia w przestrzeni. Wykorzystuje do tego sensory: akcelerometr, żyroskop oraz czujnik magnetyczny. Na ich podstawie określić można azymut oraz pułap urządzenia. Komponent ten udostępnia interfejs SensorManger.
- BluetoothLEManager - komponent udostępniający interfejsy: BluetoothScannerManager oraz ConnectedBleDeviceProvider. Pierwszy z nich odpowiada za włączenie skanera urządzeń Bluetooth LE, drugi natomiast za nawiązanie połączenia z danym urządzeniem Bluetooth, monitorowanie stanu połączenia, automatyczne odnawianie połączenia oraz wymianę danych z urządzenia Bluetooth LE.
- InternetManager - komponent, którego zadaniem jest udostępnienie interfejsu pozwalającego na komunikację aplikacji mobilnej z zewnętrznym serwerem. Udostępniany interfejs posiada zdefiniowane metody.
- UserAuthorization - komponent odpowiedzialny za autoryzację użytkownika. Wykorzystuję interfejs ApiService, udostępniany przez komponent InternetManager, pozwala na rejestrację nowego użytkownika lub logowanie już istniejącego. W tym celu udostępnia interfejs AuthContract.
- DataBase - ostatni komponent należący do podsystemu Application jest odpowiedzialny za podstawowe operacje na danych: zapis, odczyt oraz usunięcie ich z urządzenia mobilnego. Udostępnia on dwa interfejsy BleDeviceRepository oraz InternetDeviceRepository.

Kolejnym wydzielonym przeze mnie podsystemem jest UserGui. Jest to zbiór komponentów, które odpowiadają za interfejs użytkownika, wyświetlane widoki. Wykorzystują on interfejsy udostępniane przez uprzednio opisane komponenty.

- ARView - główny komponent całej aplikacji. Odpowiada on za wizualizację urządzeń IoT oraz odczytów wartości pochodzących z ich sensorów. Wykorzystuje do tego widok AR, który jest połączeniem widoku z kamery oraz nałożonej sceny zbudowanej dzięki OpenGL ES. Korzysta on z interfejsu SensorManager, ConnectedBleDeviceProvider, ApiConnectionProvider.
- MapView - komponent, dzięki któremu użytkownik może zdefiniować, wykorzystując do tego komponent mapowy, obszar z którego pobrane mają zostać odczyty sensorów urządzeń IoT. Wykorzystuje on interfejsy ApiConnectionProvider oraz InternetDeviceRepository.
- AuthorizationView - jest to komponent pozwalający użytkownikowi dokonać procesu autoryzacji poprzez rejestrację lub logowanie. Korzysta z połączenia Internetowego dlatego wykorzystuje interfejs ApiConnection.
- BluetoothScannerView - komponent odpowiadający za widok, pozwalający użytkownikowi na wyszukiwanie nowych urządzeń Bluetooth, połączenie z nim oraz zapisanie ich w pamięci urządzenia mobilnego. Wykorzystuje interfejsy BluetoothScannerManager oraz BleDeviceRepository.
- MainView - to ostatni komponent wchodzący w skład w UserGui. Udostępnia on użytkownikowi możliwość usuwania zapisanych urządzeń IoT oraz monitorowanie stanu ich połączenia. Komponent ten korzysta z czterech interfejsów: ApiConnectionProvider, BleDeviceRepository, InternetDeviceRepository oraz ConnectedBleDeviceProvider.

Serwer, który jest integralną częścią systemu, aczkolwiek nie obligatoryjną, odpowiada za autoryzację użytkowników, agregowanie oraz udostępnianie odczytów z urządzeń IoT. Posiada następujące komponenty:

- SpringBoot - jest to komponent wykorzystujący framework Spring Boot. Posiada szereg automatycznych konfiguracji umożliwiających wystawienie REST API. Korzysta z interfejsów UserRepository oraz DeviceRepository w celu zapisu danych.

Natomiast, aby dokonać autoryzacji użytkownika, wykorzystuje interfejs JwtManager.

- DataBase - komponent pozwalający wykonywać podstawowe operacje na danych (CRUD) dotyczące użytkownika oraz urządzeń IoT. W tym celu została użyta nierelacyjna baza danych MongoDB, udostępniając interfejsy: UserRepository oraz DeviceRepository.
- Security - serwer (wykorzystując ten komponent) pozwala na autoryzację użytkowników. Oznacza to, że udostępnia interfejs JwtManager, dzięki któremu można dokonać procesu rejestracji lub logowania użytkownika, a następnie wykonywania tylko autoryzowanych zapytań. W tym celu wykorzystuję technologię JWT.

Urządzenia IoT są to wszelkie urządzenia posiadające różnego typu sensory oraz wykorzystujące jeden z wielu protokołów komunikacyjnych w celu wymiany danych. W projektowanym systemie korzystam z urządzeń wykorzystujących komunikację HTTP oraz Bluetooth. Posiadają one abstrakcyjne komponenty:

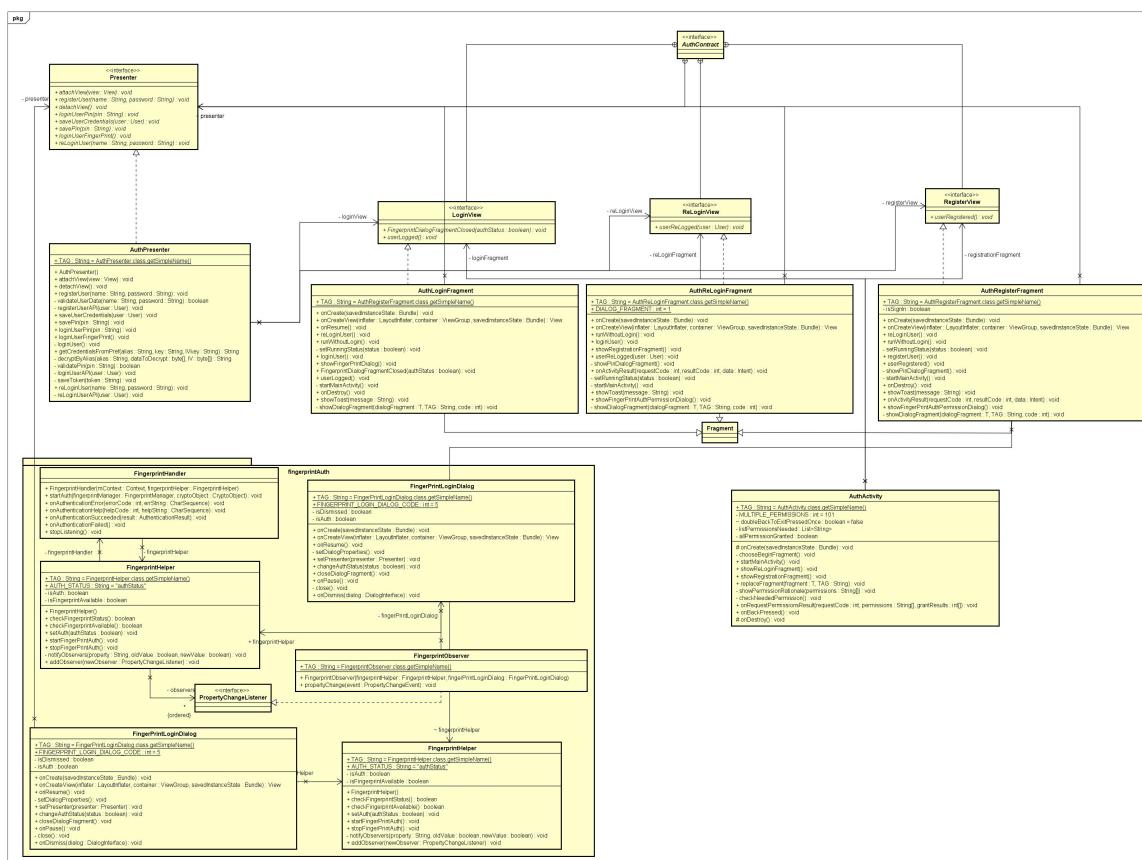
- SensorAdapter - komponent, który pozwala na wymianę danych między dołączonymi sensorami a systemem operacyjnym urządzenia IoT.
- DeviceSystem - abstrakcyjny komponent, który reprezentujący oprogramowanie zarządzające danym urządzeniem IoT.

5.5 Diagram - klas

Diagramy klas zbudowałem dla głównych pakietów, które pozwalają na realizację przypadków użycia:

- pakiet authManger - diagram zawierający klasy, dzięki którym dokonywany jest proces autoryzacji. Aplikacja mobilna umożliwia użytkownikowi rejestrację nowego konta, logowanie oraz autoryzację kodem PIN lub odciskiem palca. Każda z dostępnych możliwości autoryzacji posiada oddzielny widok: AuthLoginFragment, AuthReLoginFragment oraz AuthRegisterFragment. Rolę zarządzającą procesem autoryzacji użytkownika odgrywa AuthPresenter. W przypadku, gdy użytkownik wyraził zgodę na autoryzację poprzez odcisk palca, logowanie przeprowadzone jest przez klasy z pakietu fingerAuth.

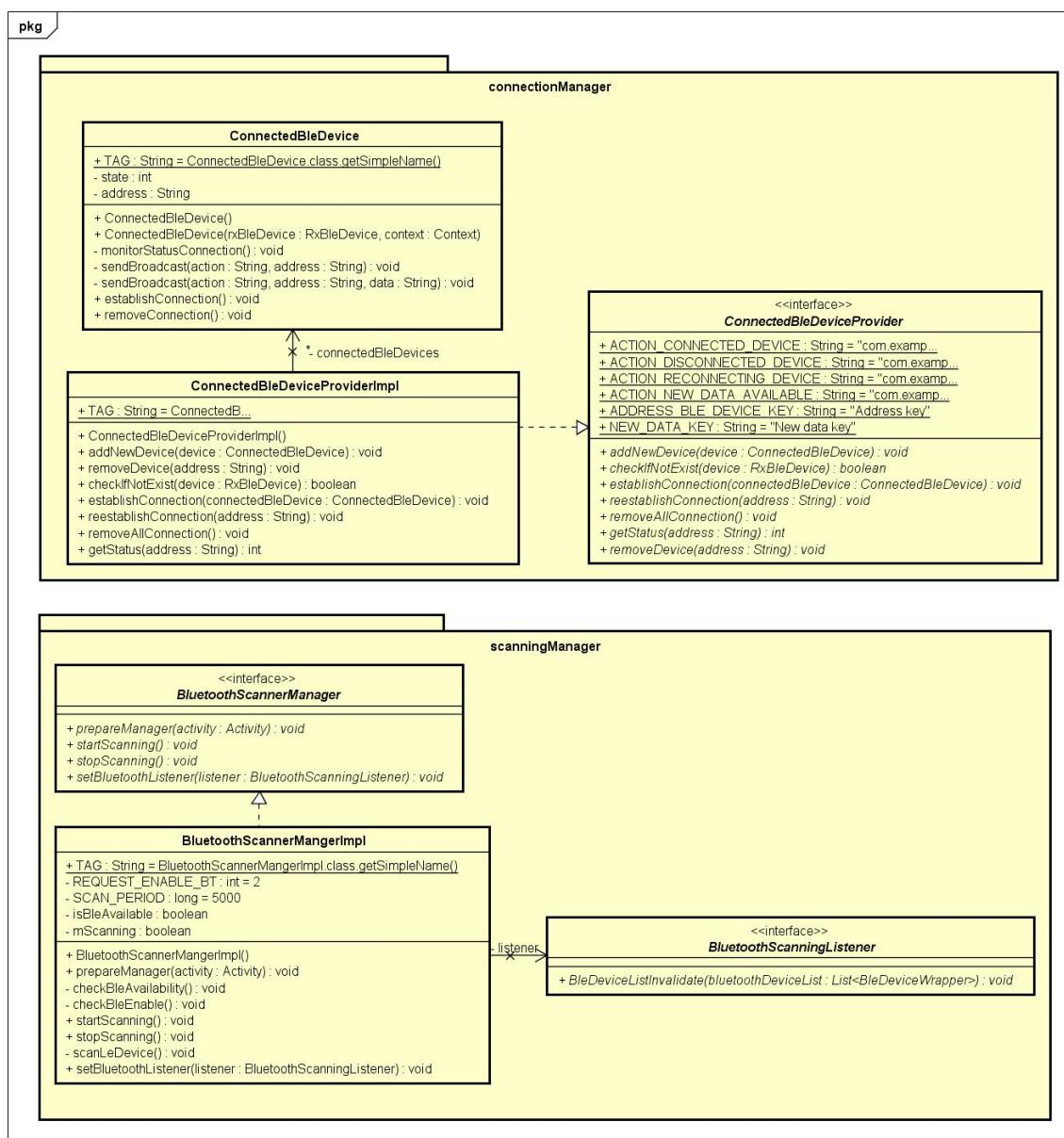
Rysunek 21: Diagram klas dla pakietu authManager



Źródło własne

- pakiet bluetoothManager - zawiera dwa odrębne pakiety: connectionManager oraz scanningManager. Pierwszy z nich pozwala na łączenie się z nowymi urządzeniami IoT, automatyczne odnawianie połączenia w przypadku rozłączenia oraz rozłączenie. W tym celu wykorzystywana instancja obiektu RxBleDevice, która wykorzystując koncepcję RxJava, pozwala na zarządzanie urządzeniami IoT. Drugi z pakietów dostarcza mechanizm wyszukiwania nowych urządzeń IoT Bluetooth. Poprzez BluetoothScanningListener odpowiednim klasom dostarczana jest lista znalezionych urządzeń IoT.

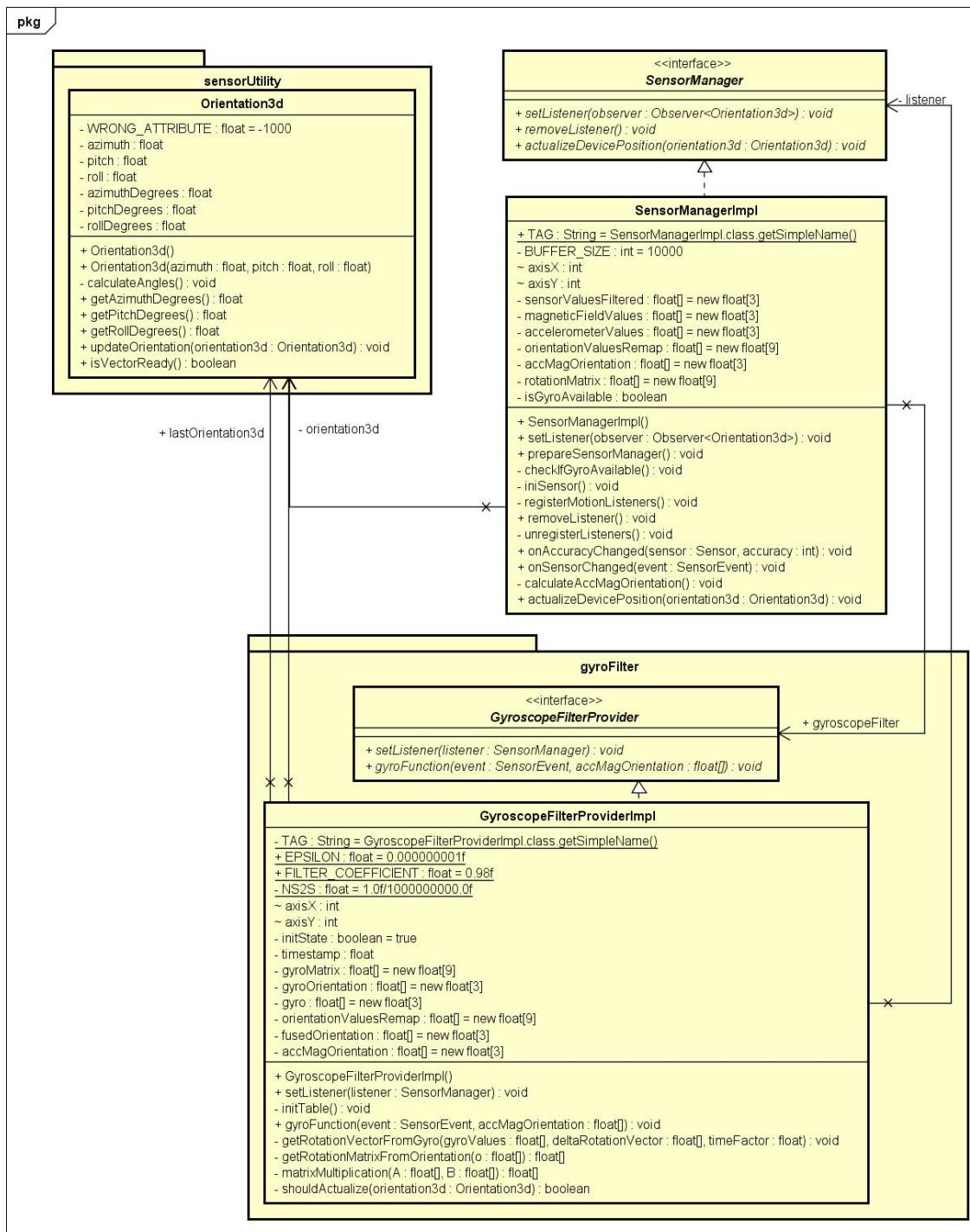
Rysunek 22: Diagram klas dla pakietu bluetoothManager



Źródło własne

- pakiet motionSensor - zbiór klas dostarczający interfejs SensorManager, odpowiadający za informowanie o zmianie położenia urządzenia w przestrzeni, poprzez wywołanie metody actualizeDevicePosition(Orientation3d orientation3d). W przypadku, gdy urządzenie posiada żyroskop, wykorzystywane są klasy z pakietu gyroFilter w celu zwiększenia dokładności podczas określania położenia w przestrzeni.

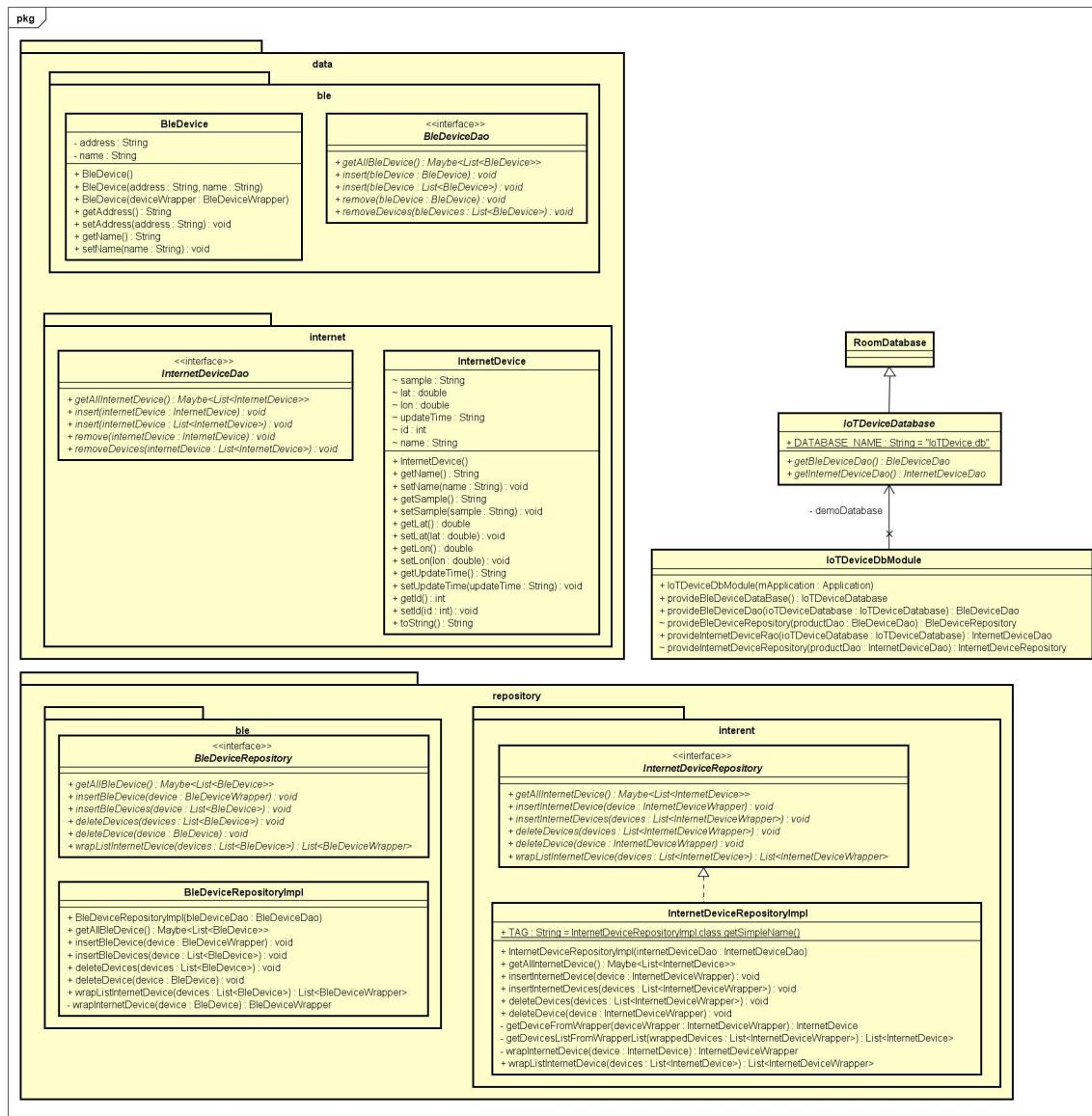
Rysunek 23: Diagram klas dla pakietu motionSensor



Źródło własne

- pakiet DataBase - diagram zawiera dwa pakiety: data oraz repository. Odpowiadają one za dostarczenie jednolitego interfejsu do komunikacji pomiędzy źródłem danych a aplikacją. Ponadto w głównym pakiecie znajdują się klasy opowiadające za inicjalizację instancji nie relacyjnej bazy danych - Room.

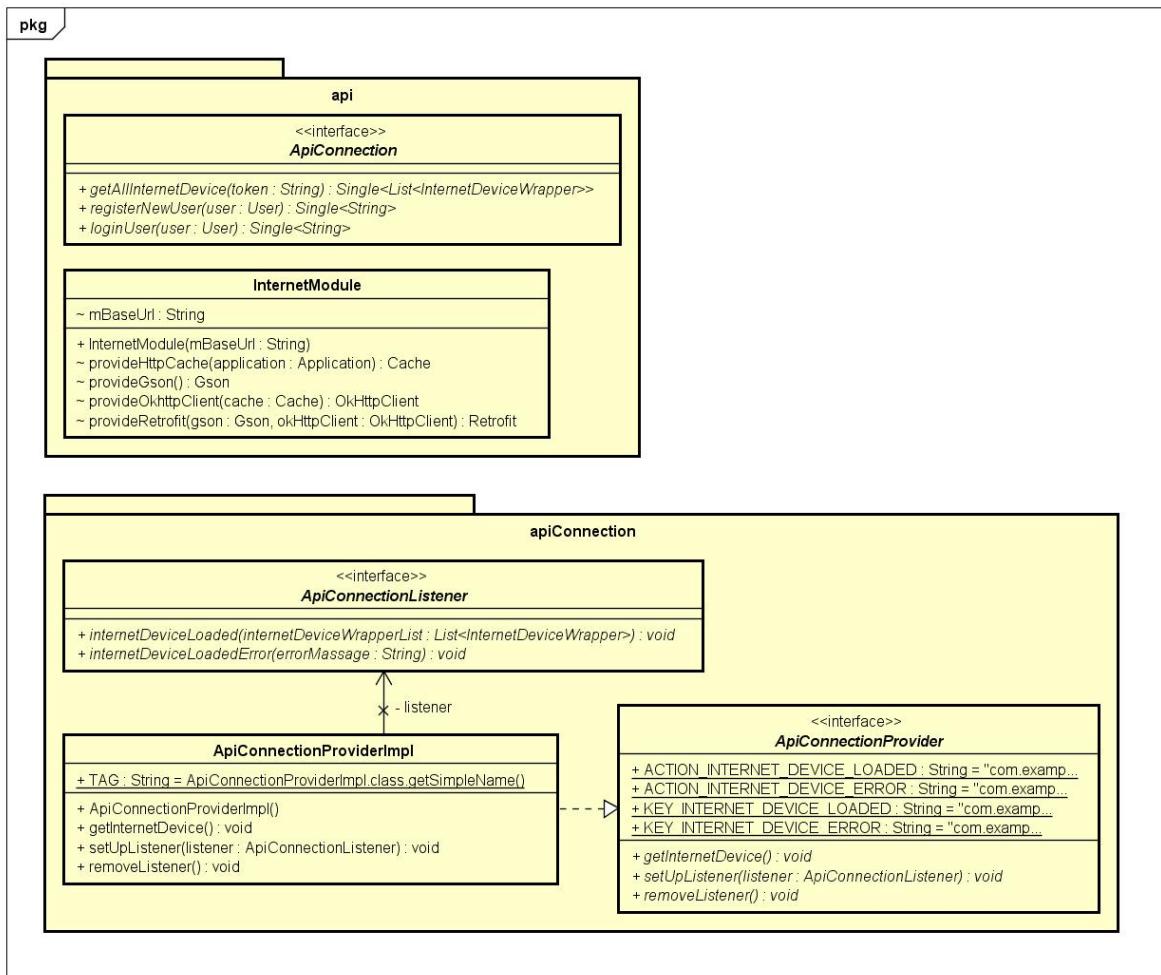
Rysunek 24: Diagram klas dla pakietu DataBase



Źródło własne

- pakiet internetManager - klasy wchodzące w skład jego podpakietu dostarczają interfejs ApiConnection, który umożliwia wymianę danych między urządzeniem mobilnym a zewnętrznym serwerem. Natomiast pakiet apiConnection pozwala różnym modułom aplikacji na pobieranie danych dotyczących urządzeń IoT, wykorzystując do tego listenera.

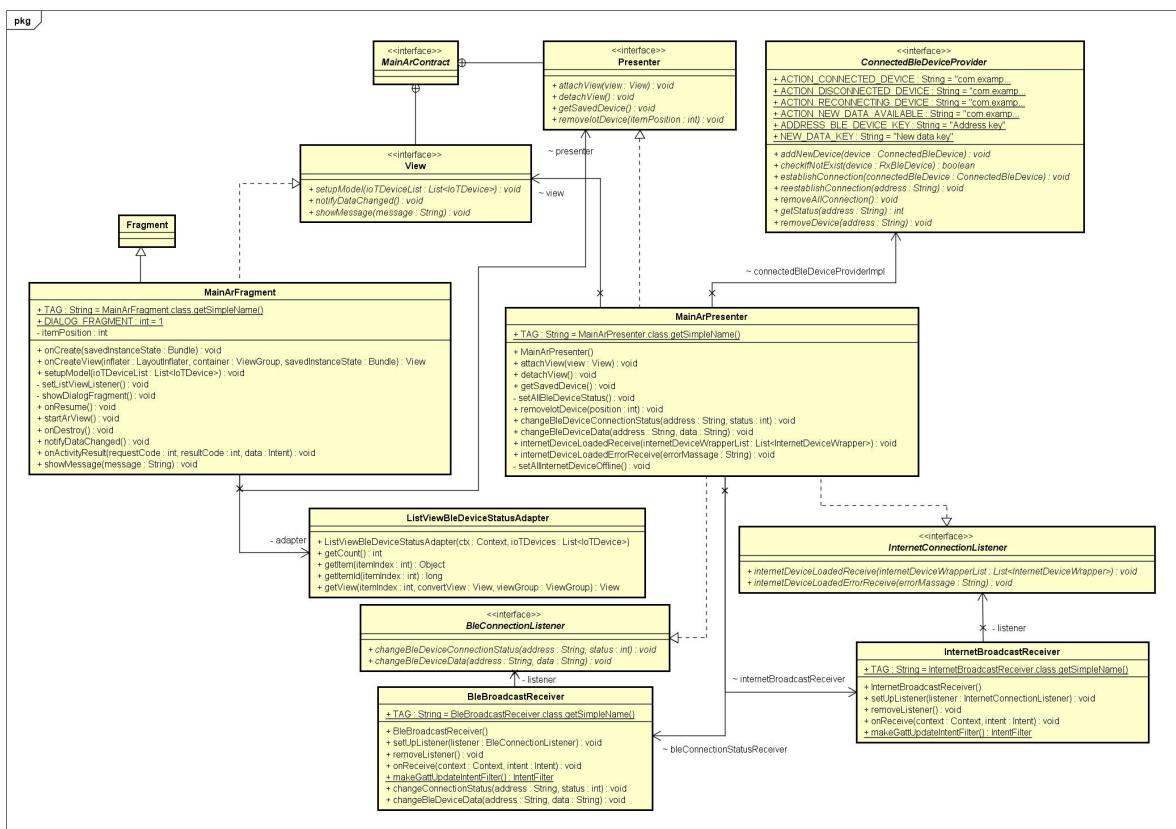
Rysunek 25: Diagram klas dla pakietu internetManager



Źródło własne

- pakiet mainView - w skład tego pakietu wchodzi klasa MainArFragment, odpowiedzialna za generowanie głównego widoku aplikacji oraz MainArPresenter, która zarządza jej treścią. Wykorzystuje w tym celu interfejsy: BleConnectionListener, InternetConnectionListener oraz ConnectedBleDeviceProvider, które dzięki zarejestrowanym broadcastRevisers otrzymują intenty z poszczególnych modułów. Prezenter agreguje oraz odpowiednio przetwarza otrzymane dane, a następnie przekazuje do widoku.

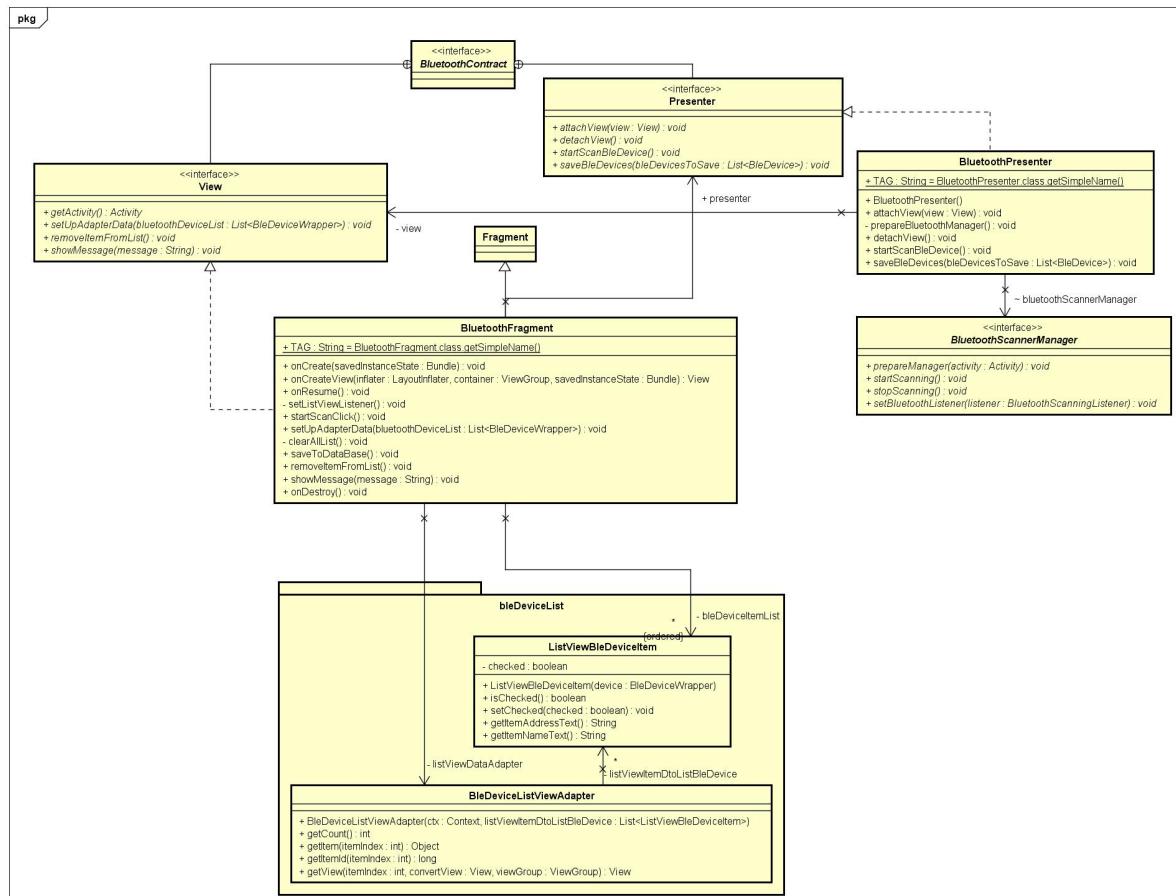
Rysunek 26: Diagram klas dla pakietu mainView



Źródło własne

- pakiet bluetoothScannerView - pakiet zawierający klasy, które odpowiadają za wyświetlanie, w formie listy, znalezionych urządzeń IoT poprzez Bluetooth. W celu uruchomienia procesu wyszukiwania urządzeń IoT wykorzystywany jest interfejs BluetoothScannerManager. Pakiet bleDeviceList zawiera pomocnicze klasy, służące do wyświetlania urządzeń IoT w odpowiednim formacie listy.

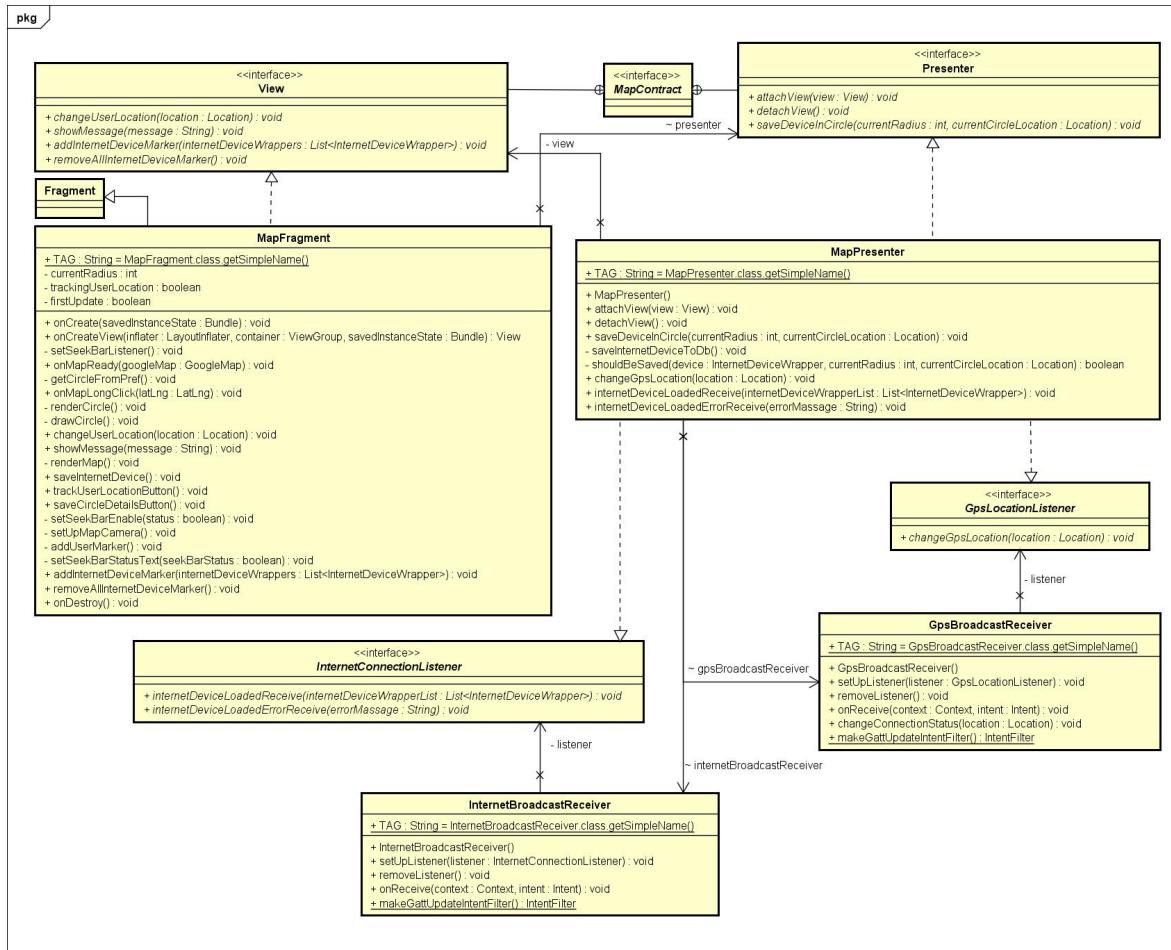
Rysunek 27: Diagram klas dla pakietu bluetoothScannerView



Źródło własne

- pakiet mapView - klasy znajdujące się w tym pakiecie, wykorzystując interfejsy InternetConnectionListener oraz GpsLocationListener, za pośrednictwem klasy MapFragment generują widok mapy świata. Użytkownik ma możliwość zdefiniowania obszaru na mapie, z którego odczyty z czujników mają być wyświetlane w widoku AR.

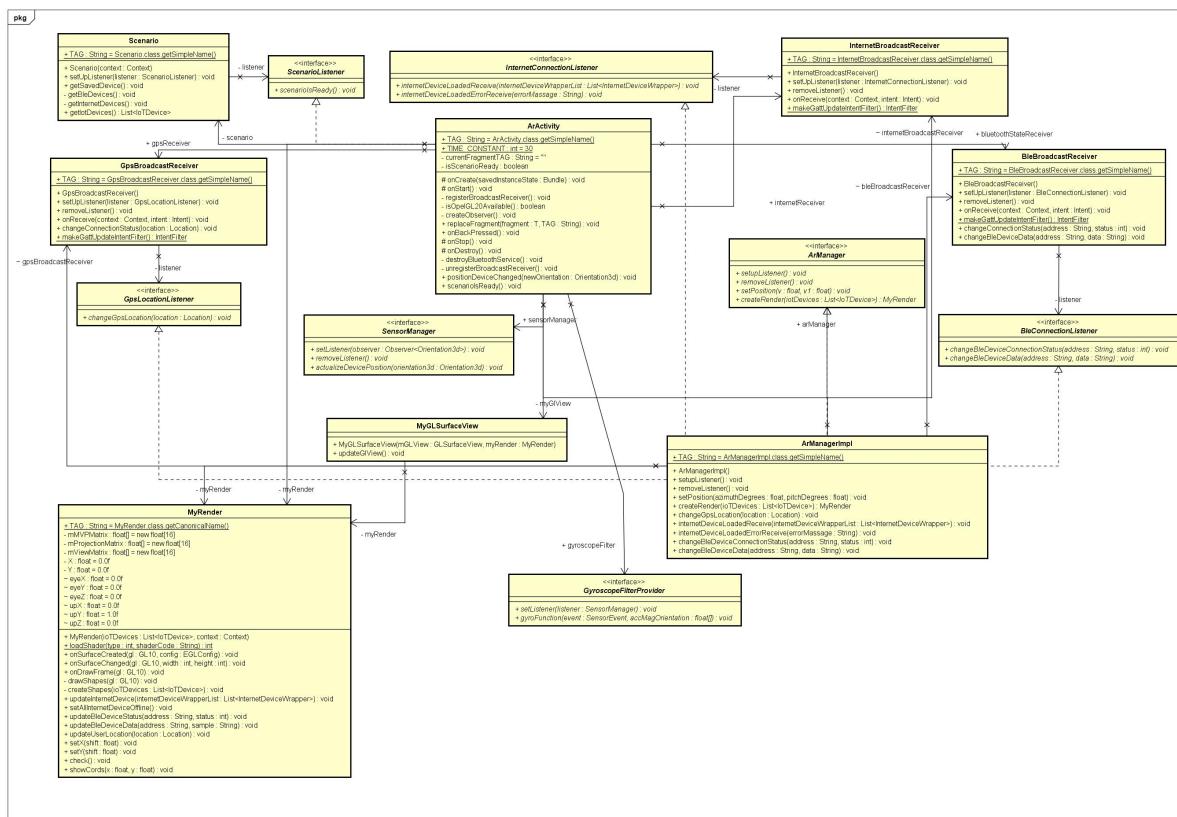
Rysunek 28: Diagram klas dla pakietu mapView



Źródło własne

• pakiet arEngine - zbiór klas odpowiadających za wyświetlanie widoku AR oraz aktualizowanie go. Jego głównym elementem jest klasa ArManagerImpl. Zarządza ona danymi pochodząymi z urządzeń IoT za pośrednictwem protokołu HTTP oraz modułu Bluetooth. Wykorzystuje w tym celu klasy Broadcast Receiver: InternetBroadcastReceiver, GpsBroadcastReceiver oraz BleBroadcastReceiver, które po zarejestrowaniu sygnalizują zmianę danych pochodzących z danego modułu poprzez Listener. Natomiast w celu rejestrowania zmian w położeniu urządzenia w przestrzeni wykorzystywany jest interfejs SensorManager oraz GyroscopeFilterProvider przez klase ArActivity. Kolejną ważną klasą jest MyRender. Jej zadaniem jest zarządzanie sceną OpenGL oraz renderowaniem jej. Każdorazowo po zamianie jakiejkolwiek z danych urządzeń IoT m czy też w wyniku zmiany położenia urządzenia mobilnego w przestrzeni scenam jest ponownie budowana. Odpowiada za to funkcja onDrawFrame(GL10 gl).

Rysunek 29: Diagram klas dla pakietu arEngine



Źródło własne

5.6 Diagram - sekwencji

W celu zobrazowania interakcji zachodzących pomiędzy częściami systemu posłużyłem się diagramami sekwencji. Wyróżniłem sześć głównych procesów w działaniu aplikacji mobilnej dla której opracowałem diagramy wraz z opisami. Z racji formatu i wielkości diagramu w pierwszej kolejności umieszczę opis diagramu, a następnie sam diagram w następującej kolejności:

- diagram sekwencji dla procesu: autoryzacja jako gość,
- diagram sekwencji dla procesu: rejestracja nowego użytkownika,
- diagram sekwencji dla procesu: logowanie,
- diagram sekwencji dla procesu: wyszukanie urządzeń IoT Bluetooth,
- diagram sekwencji dla procesu: wyszukanie urządzeń IoT z serwera,
- diagram sekwencji dla procesu: wyświetlanie widoku AR.

Pierwszy z diagramów (Rys. 30), z racji tego iż nie wymaga podawania żadnych danych przez użytkownika jest bardzo prosty i zwięzły. Użytkownik autoryzuje się jako gość, przez co dostęp do możliwości pobierania danych z sensorów urządzeń IoT z serwera jest zablokowany.

Drugi diagram (Rys. 31) przedstawia proces rejestrowania nowego użytkownika. Użytkownik podaje nazwę użytkownika a następnie hasło. Dane zostają wysłane na serwer w celu sprawdzenia, czy użytkownik o takich danych już nie istnieje. W przypadku, gdy dane są poprawne, użytkownik jest zapisywany w bazie danych oraz generowany zostaje token, który jest zwracany do aplikacji mobilnej. Następnie dane użytkownika zostają zaszyfrowane, a dalej lokalnie zapisane w pamięci urządzenia. Użytkownik jest proszony o podanie kodu pin, którym będzie się w przyszłości logował. Kolejnym elementem jest potwierdzenie wyrażenia zgody na wykorzystanie czujnika linii papilarnych w celu autoryzacji. Po przeprowadzeniu procesu rejestracji aplikacja przechodzi do ekranu głównego.

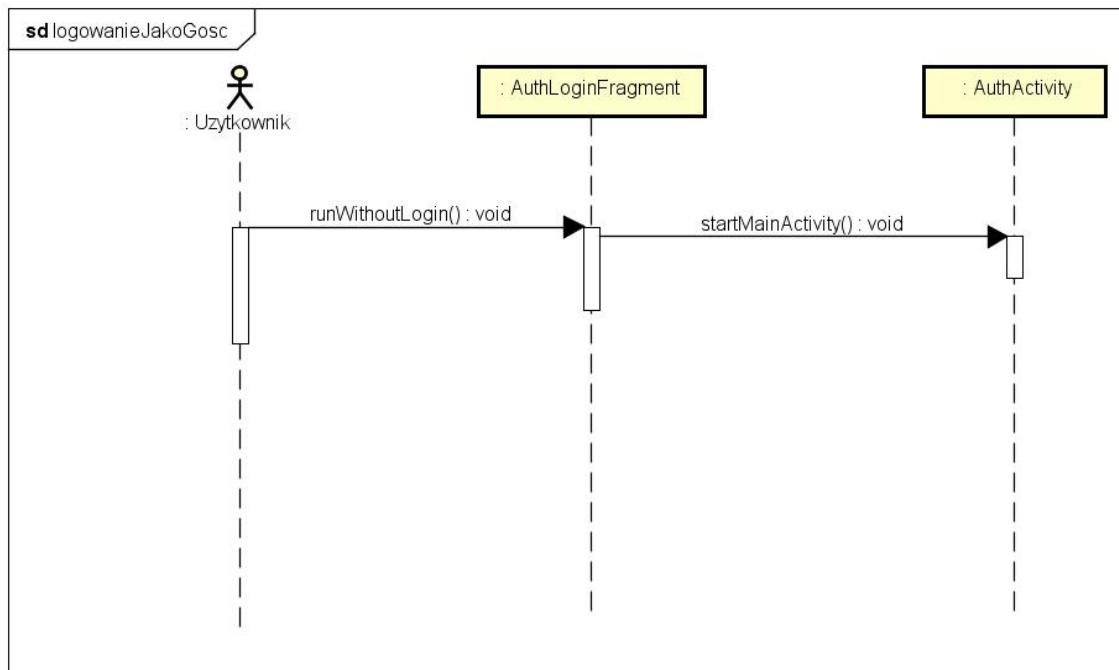
Logowanie jest trzecim procesem, który został zobrazowany w postaci diagramu sekwencji na Rys. 32. Po podaniu przez użytkownika kodu Pin, zapisane dane w pamięci urządzenia zostają rozszyfrowane i przesłane na serwer w celu weryfikacji. Potwierdzenie autentyczności danych skutkuje przejście aplikacji do ekranu głównego.

Kolejny diagram (Rys. 33) prezentuje proces wyszukiwanie nowych urządzeń IoT za pośrednictwem serwera. Cyklicznie, co zadany okres czasu serwis odpytuje zewnętrzny serwer o aktualne dane pochodzące z sensorów urządzeń IoT. Po pobraniu danych wysyła intent, który odbiera zarejestrowany broadcastReveivera. W dalszej kolejności MapPresenter jest informowany poprzez listener o aktualizacji danych z urządzeń IoT.

Proces wyszukiwania nowych urządzeń IoT Bluetooth został zaprezentowany na diagramie [Rys. 34]. W tym celu wykorzystany został BluetoothScannerManager oraz jego implementacja BluetoothPresenter, za pośrednictwem BluetoothScanningListener, jest informowany o znalezieniu nowych urządzeń Bluetooth. Tę informację następnie przekazuje do widoku - BluetoothFragment.

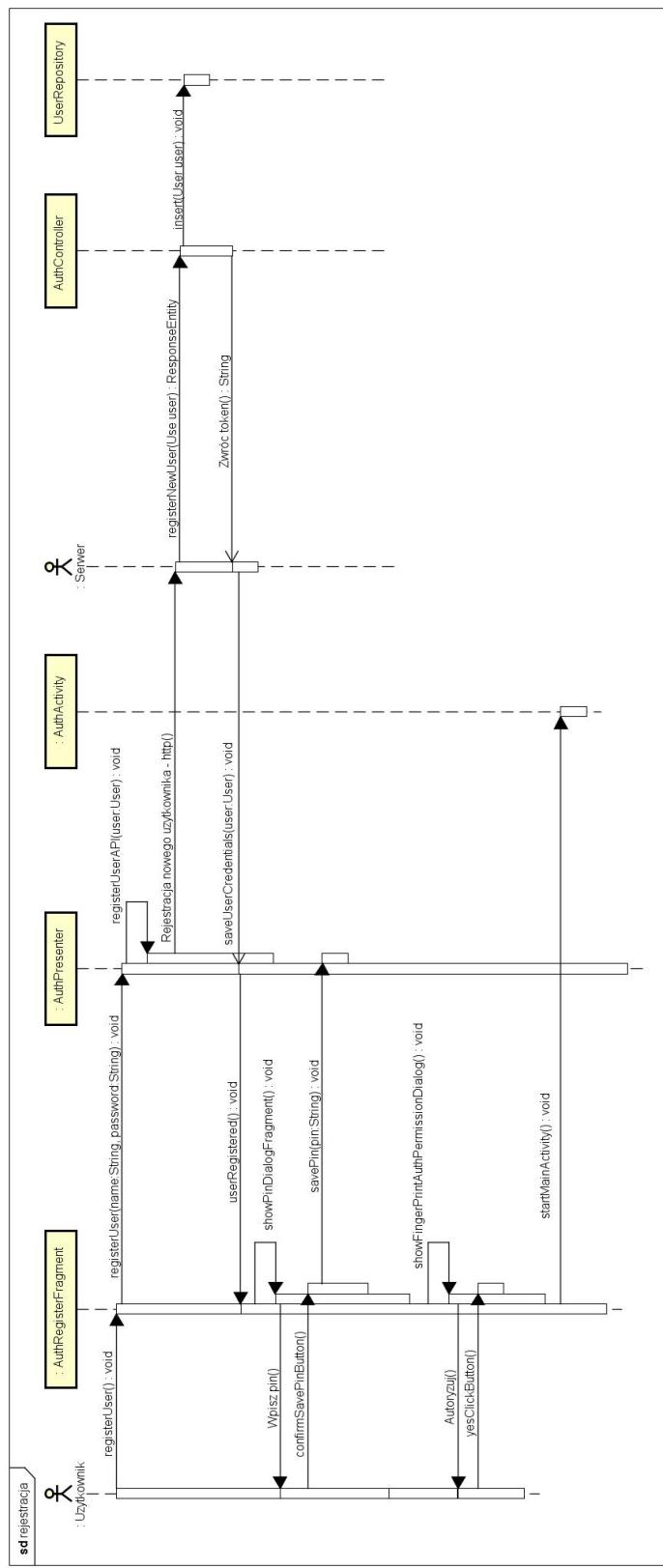
Ostatnim procesem jest wyświetlanie widoku AR [Rys. 35]. Jest to najbardziej skomplikowany diagram, ze względu na fakt występowania asynchronicznych interakcji listenerów: BleConnectionListener, GpsLocationListener, InternetConnectionListener. Dostarczają one informacji ArManager o zmianie stanu połączenia z urządzeniami IoT (zarówno z urządzeniami Bluetooth oraz z urządzeniami, których dane przechowywane są na zewnętrznym serwerze) lub o zmianie lokalizacji przebywania. ArManager następnie przekazuje informacje do klasy MyReneder, która odpowiada za renderowanie widoku AR. Dodatkowo klasa SensorManager dostarcza informacje (kąt azymut i pułap) na temat aktualnego położenia urządzenia w przestrzeni. Dane te są przetwarzane i na ich podstawie obracana zostaje kamera - zbudowanej sceny AR. Jednocześnie generowany jest ponownie widok AR z nowymi parametrami.

Rysunek 30: Diagram sekwencji - autoryzacja jako gość.



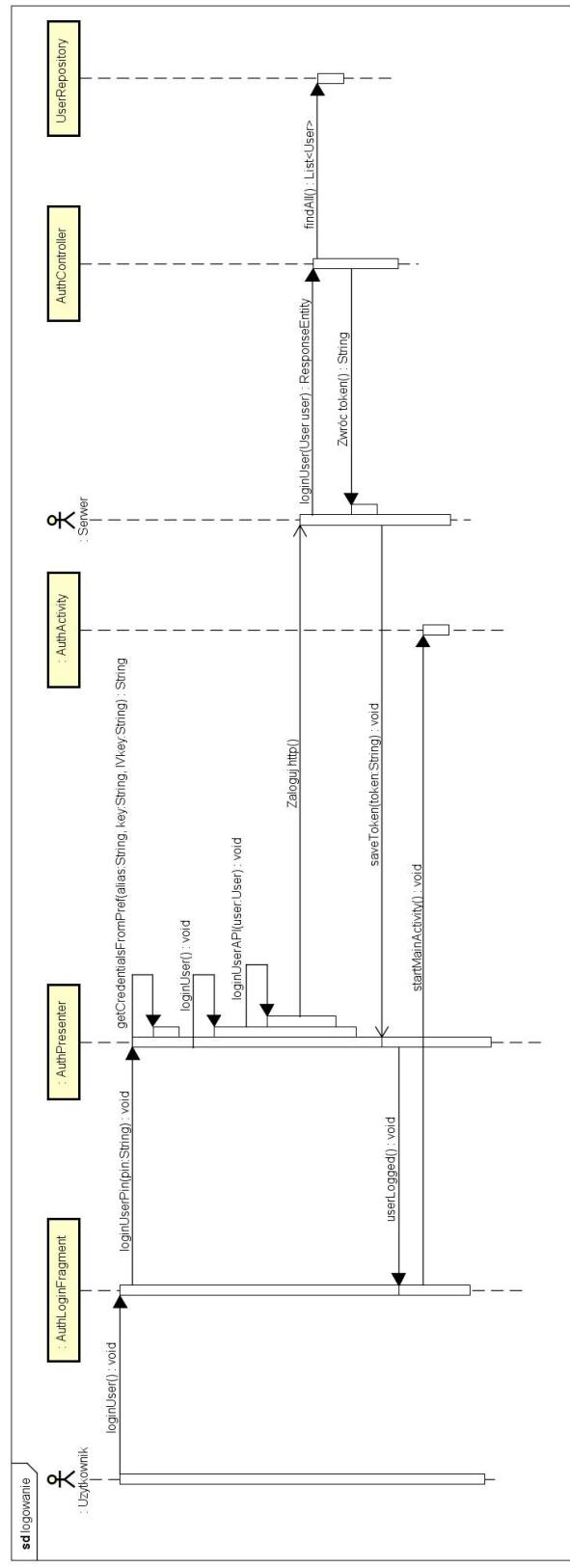
Źródło własne

Rysunek 31: Diagram sekwencji - rejestracja nowego użytkownika.



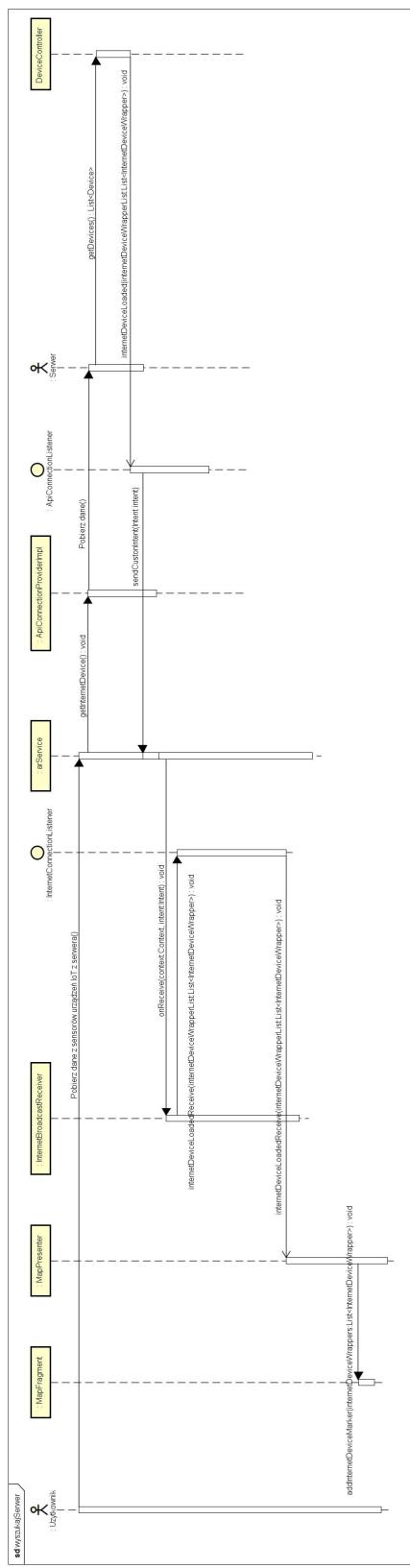
Źródło własne

Rysunek 32: Diagram sekwencji - logowanie użytkownika.



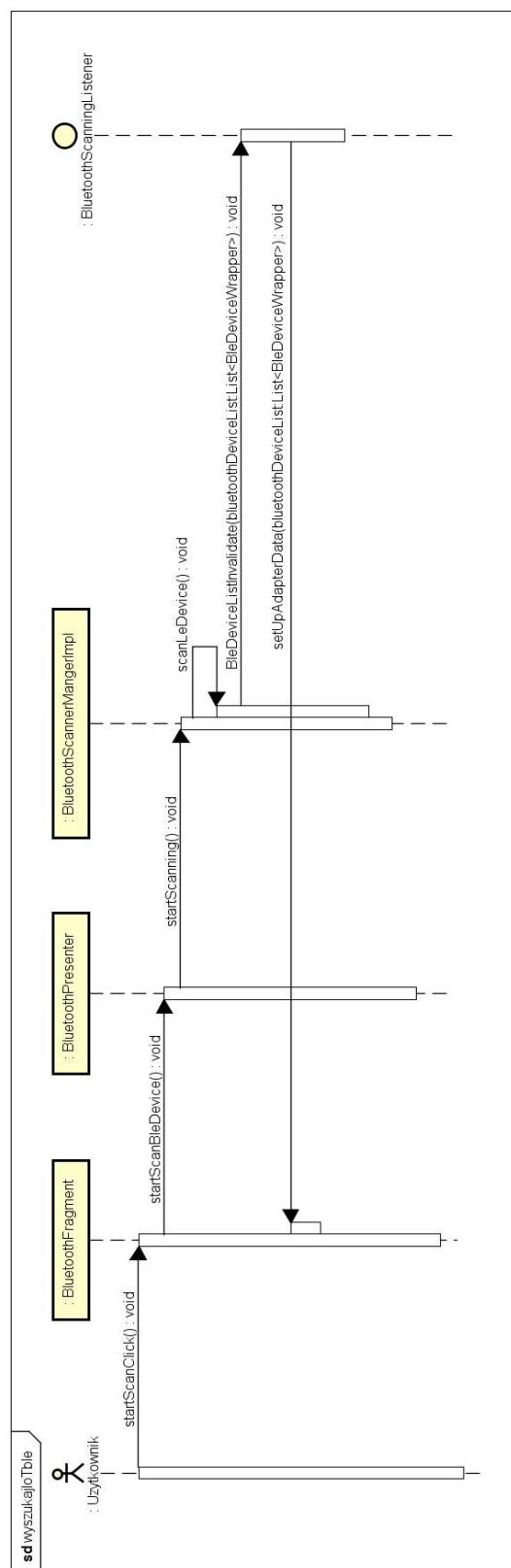
Źródło własne

Rysunek 33: Diagram sekwencji - wyszukanie urządzeń IoT z serwera.



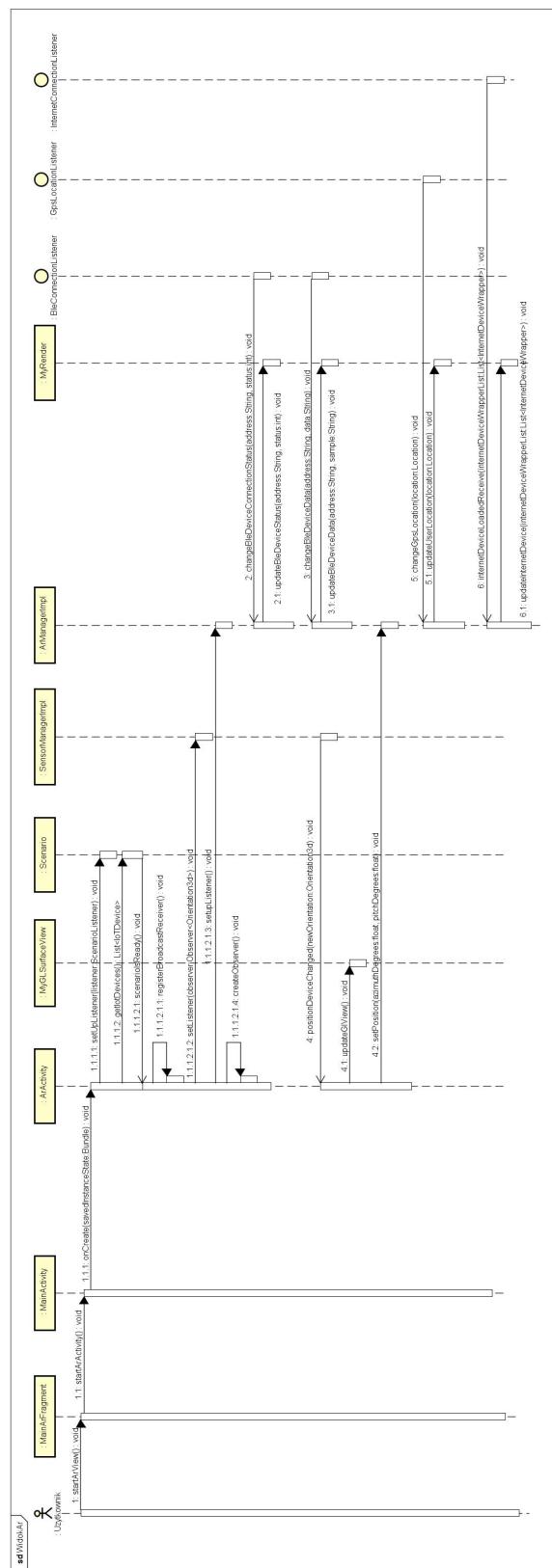
Źródło własne

Rysunek 34: Diagram sekwencji - wyszukanie urządzeń IoT Bluetooth.



Źródło własne

Rysunek 35: Diagram sekwencji - wyszukanie urządzeń IoT z serwera.



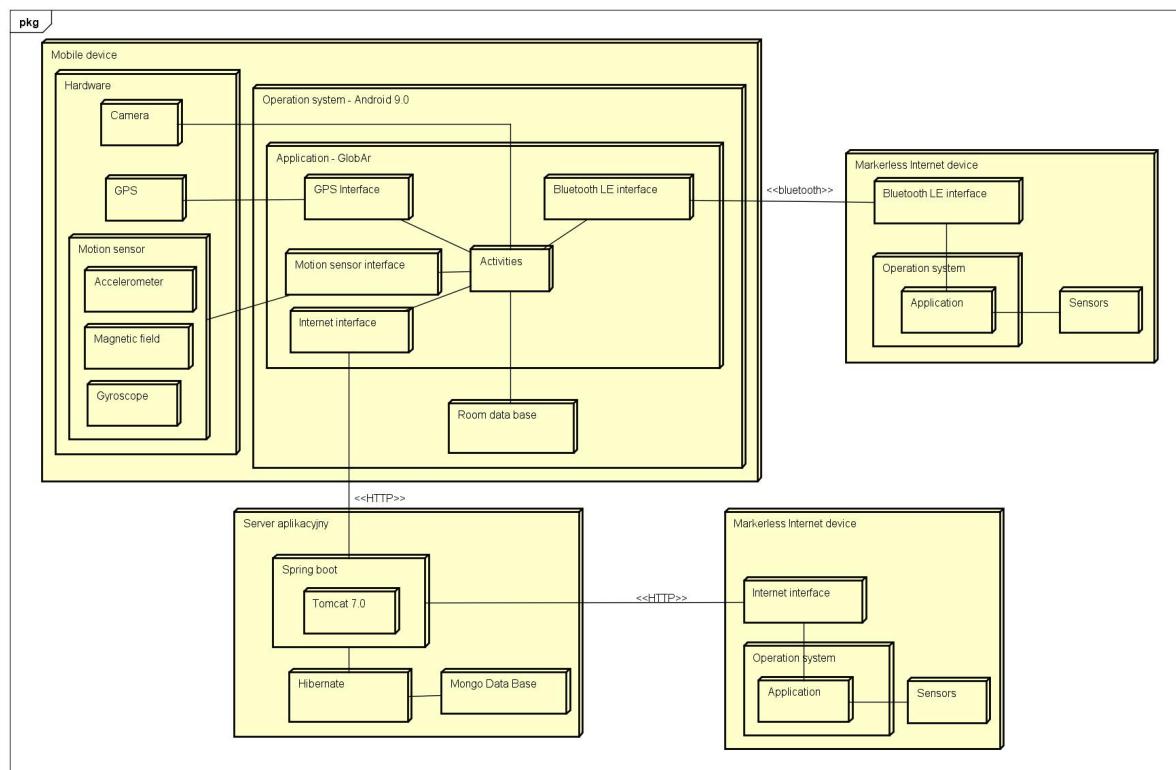
Źródło własne

5.7 Diagram - wdrożenia

Diagram wdrożenia reprezentuje fizyczne rozmieszczenie składników systemu. W opracowanym przeze mnie systemie jego głównym elementem jest urządzenie mobilne, które posiada ograniczenia systemowe (Android w wersji min. 4.3) oraz sprzętowe (sensory ruchu, kamera, moduł GPS). Drugą składową systemu są urządzenia IoT. Dokonałem podziału na te urządzenia, które wykorzystują komunikację Bluetooth LE oraz te, które korzystają z protokołu HTTP w celu udostępniania odczytów z wbudowanych lub podłączonych sensorów. Każde z tych urządzeń musi posiadać interfejs umożliwiający komunikację oraz system operacyjny, zarządzający całością urządzenia. Ostatnim elementem jest serwer aplikacyjny wykorzystujący framework Spring Boot wraz z Tomcat 7. Wykorzystując protokół HTTP dokonuje autoryzacji użytkowników aplikacji mobilnych. W celu agregowania danych z urządzeń IoT wykorzystuje bazę danych Mongo DB.

Diagram zawierający szczegółową reprezentację środowiska wdrożenia został umieszczony poniżej.

Rysunek 36: Diagram wdrożenia



Źródło własne

Rozdział 6

Implementacja oraz środowisko wytwarzające

W tym rozdziale omówione zostaną technologie wykorzystane podczas budowania systemu oraz kluczowe, aspekty tego procesu.

6.1 Wykorzystane technologie oraz narzędzia

Z racji tego, iż system składa się z trzech odrębnych elementów: urządzenie mobilne, serwer oraz urządzenia IoT, wykorzystywać będę różne środowiska wytwarzające. Korzystanie z każdego z nich jest bezpłatne, bądź producent udostępnia darmową licencję dla studentów.

W związku z użyciem urządzeń mobilnych wykorzystujących system operacyjny Android, zdecydowałem się na wybór środowiska programistycznego Android Studio [42]. Jest to dedykowane środowisko IDE udostępniające deweloperom wiele narzędzi ułatwiających pracę nad oprogramowaniem. Między innymi jest to wirtualny emulator urządzeń mobilnych, dzięki któremu można testować oprogramowanie bez konieczności wgrywania go na fizyczne urządzenie, czy też inteligentne podpowiedzi podczas pisania kodu - proponowane nazwy obiektów, zestawienie funkcji jakie można wywołać na danym obiekcie. Android Studio dostarcza również dedykowane narzędzie do tworzenia widoków aplikacji na zasadzie umieszczania gotowych komponentów (listy, przyciski, grafiki itp.).

Podczas projektowania i implementacji oprogramowania serwera, z racji tego, iż korzystam z framework'u Spring Boot jako środowisko wytwarzające wybrałem IntelliJ IDEA [43]. Pozwala on na wygenerowanie szablonu aplikacji Spring Boot, z podstawową

konfiguracją, przygotowanymi klasami uruchomiającymi oraz kontrolerami. Udostępnia on ponadto te same opcje wspomagające programistę co Android Studio.

W przypadku urządzeń IoT podczas testowania wykorzystałem moduł Genuino 101, który posiada Bluetooth Low Energy. W celu projektowania i wgrywania oprogramowania wykorzystałem program Arduino [44]. Umożliwia on komplikację kodu C lub C++, a następnie śledzenie pracy urządzenia, dzięki monitorowi portu szeregowego. Sam program zawiera wiele przydatnych bibliotek, które można dołączyć do projektu.

Dokumentację techniczną sporządziłem w programie Astah [45]. Pozwala ona na budowanie wszystkich kluczowych diagramów. Ponadto istnieje możliwość wgrania kodu dowolnej aplikacji i wygenerowania diagramu klas dla każdego z pakietu automatycznie. Następnie podczas konstruowania diagramów sekwencji możemy wybierać klasy i ich metody na podstawie wcześniej wygenerowanych diagramów klas.

Całą pracę napisałem w programie Texmaker w języku LaTeX [46]. Środowisko to pozwala uporządkować tekst, dostarczając możliwość generowanie między innymi tabel oraz listingów, automatycznych ich spisów.

6.2 Kluczowe aspekty programistyczne

Urządzenie mobilne, które generuje widok Rozszerzonej Rzeczywistości, jest głównym elementem projektowanego systemu, dlatego w tym podrozdziale przedstawię i omówię kluczowe aspekty programistyczne aplikacji mobilnej. Serwer oraz urządzenia IoT dostarczają tylko dane, które następnie są wyświetlane przez urządzenie mobilne. Z racji tego, że ich implementacje mogą być różne, ale dostarczać one muszą odpowiednie interfejsy, zaprezentuję i omówię te wymagane.

Podczas projektowania aplikacji mobilnej posłużyłem się wzorcem projektowym MVP (model-view-presenter). Takie podejście pozwala na odseparowanie widoku (GUI) od logiki aplikacji. Presenter zarządza widokiem i wyświetlonymi treściami przez niego. Model natomiast dostarcza dane, które będą przetwarzane.

W celu wykorzystania mechanizmu wstrzykiwania zależności (dependency injection) wykorzystałem bibliotekę Dagger 2. Takie podejście pozwala na zachowanie piątej zasady SOLID, wg której klasa powinna zależeć od abstrakcji, a nie od konkretnej instancji. Przykładem wykorzystania wstrzykiwania zależności jest pakiet `sensorManager` (Listing 6.1.). Interfejs `SensorManager` udostępnia trzy metody, natomiast funkcje, które znajdują się w klasie implementującą ten interfejs są prywatne. Logika jest niedostępna dla klas, które powołują instancję obiektu `SensorManager`. Dzięki klasie `SensorManagerModule`, która jest abstrakcyjna, mechanizm wstrzykiwania zależności zawsze w miejsce interfejsu `SensorManager` wstrzyknie jego implementację, klasę `SensorManagerImpl`, o ile powoływany obiekt zostanie opatrzony adnotacją `@Inject`.

Listing 6.1: Kod interfejsu `SensorManager` i jego implementacja.

```

public interface SensorManager {
    void setListener(Observer<Orientation3d> observer);
    void removeListener();
    void actualizeDevicePosition(Orientation3d orientation3d);

    @Module
    abstract class SensorManagerModule {
        @Binds
        public abstract SensorManager
        sensorManagerProvider(SensorManagerImpl mySensorManager);
    }
}

public class SensorManagerImpl implements
SensorManager, SensorEventListener{
    public static final String TAG = SensorManagerImpl.class.
    getSimpleName();
    private final int BUFFER_SIZE = 10000;

    @Inject
    public Logger logger;
}

```

```

@Inject
public Context context;

@Inject
public GyroscopeFilterProvider gyroscopeFilter;

int axisX;
int axisY;
Observer<Orientation3d> observer;
PublishSubject<Orientation3d> ps =
    PublishSubject.create();
private android.hardware.SensorManager mSensorManager;
private Sensor accelerometerSensor;
private Sensor magneticFieldSensor;
private Sensor gyroscopeSensor;
private Orientation3d orientation3d;

@Inject
public SensorManagerImpl() { ... }

@Override
public void setListener(Observer<Orientation3d> observer)
{ ... }

public void prepareSensorManager() { ... }

private void checkIfGyroAvailable() { ... }

private void iniSensor() { ... }

private void registerMotionListeners() { ... }

@Override
public void removeListener() { ... }

private void unregisterListeners() { ... }

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy)
{ ... }

@Override
public void onSensorChanged(SensorEvent event) { ... }

private void calculateAccMagOrientation() { ... }

@Override
public void actualizeDevicePosition
(Orientation3d orientation3d) { ... }

}

```

Źródło własne

W aplikacji mobilnej wykorzystałem paradygmat programowania reaktywnego, który jest rozszerzoną wersją wzorca projektowego Observer Pattern. W tym celu posłużyłem się biblioteką RxJava, która dostarcza API pozwalające budować mobilną aplikację wykorzystującą asynchroniczne strumienie danych. Koncepcja programowania reaktywnego polega na zdefiniowaniu obiektu producenta (Observable) (źródło danych) oraz konsumenta (Observer) (obiekt przetwarzający otrzymane dane).

Zarówno dla producenta jak i konsumenta możemy wybrać wątek, w którym dane mają być emitowane oraz przetwarzane. Jest to kluczowa kwestia z perspektywy systemu operacyjnego Android. Standardowo do wykonywania czasochłonnych operacji wykorzystywana jest klasa dziedzicząca po `AsyncTask`. Pozwala ona wykonywać operacje w tle, jednakże wymagana jest referencja do miejsca wywołania, co może powodować wycieki pamięci oraz błędy. Ponadto tylko wątek główny aplikacji może ingerować w widoki aplikacji. RxJava dostarcza kilka opcji wyboru wątku, co pokaże na przykładzie pobierania danych z bazy danych (producent emittuje dane na osobnym wątku, natomiast konsument odbiera dane na wątku głównym zmieniając widok) oraz zapisywaniu danych (producent emittuje dane na wątku głównego, ponieważ korzysta z danych z widoku, natomiast konsument odbiera dane na osobnym wątku zapisując dane do bazy danych) (Listing 6.2.).

Listing 6.2: Kod pobierania i zapisywania danych do bazy danych.

```
| @Override  
| public void getSavedDevice() {  
|     IoTDeviceList.clear();  
|  
|     bleRepository.getAllBleDevice()  
|         .subscribeOn(Schedulers.io())  
|         .observeOn(AndroidSchedulers.mainThread())  
|         .subscribe(new MaybeObserver<List<BleDevice>>() {  
|             @Override  
|             public void onSubscribe(Disposable d) {..}  
|  
|             @Override  
|             public void onSuccess(List<BleDevice> bleDevices) {  
|                 IoTDeviceList.addAll(bleRepository.  
|                     wrapListInternetDevice(bleDevices));  
|                 setAllBleDeviceStatus();  
|                 view.notifyDataChanged();  
|             }  
|             @Override  
|             public void onError(Throwable e) {...}  
|  
|             @Override  
|             public void onComplete() {...}  
|         });  
|     }  
  
|     private void saveInternetDeviceToDb() {  
|         Completable.fromAction(() ->  
|             internetRepository.insertInternetDevices  
|             (internetDeviceToSave))  
|             .observeOn(AndroidSchedulers.mainThread())  
|             .subscribeOn(Schedulers.io())  
|             .subscribe(new CompletableObserver() {  
|                 @Override  
|                 public void onSubscribe(Disposable d) {...}  
|  
|                 @Override
```

```
| public void onComplete() {
|     view.showMessage("Dodano poprawnie, " +
|         internetDeviceToSave.size() + " rekordow");
| }
| @Override
| public void onError(Throwable e) {
|     logger.log(TAG, e.getMessage());
|     view.showMessage(context.getString(R.string.api_error));
| }
| };
```

Źródło własne

6.3 Implementacja oprogramowania - urządzenie mobilne

W tym podroziale przedstawię najważniejsze fragmenty aplikacji od strony kodu, które dostarczają funkcji zdefiniowanych w postaci przypadków użycia. Skupię się na sposobie generowania widoku rozszerzonej rzeczywistości oraz dostarczaniu danych pochodzących z urządzeń IoT.

W celu połączenia się z urządzeniami Bluetooth, monitorowania statusu ich połączenia oraz otrzymywania danych zaimplementowałem klasę `ConnectedBleDevice` (Listing 6.3.). Korzysta ona z wcześniej omówionego podejścia reaktywnego do programowania. Każda zmiana powoduje wysłanie informacji (intent), która zawiera szczegół akcji i klucz. Zarejestrowane `broadcastReceiver` w odpowiednich modułach odbierają te wiadomości, a następnie odpowiednio przetwarzają. Takie podejście eliminuje powtarzanie kodu w wielu miejscach.

Listing 6.3: Kod połączenia z urządzeniem Bluetooth oraz monitorowanie jego statusu.

```

private void monitorStatusConnection() {
    String address = rxBleDevice.getMacAddress();
    connectionState = rxBleDevice.observeConnectionStateChanges()
        .subscribe(connectionState -> {
            Log.i(TAG, "Status changed: " + connectionState.name()
                + ", device name: " + rxBleDevice.getName());
            switch (connectionState.name()) {
                case Constants.CONNECTED:
                    this.state = Constants.CONNECTED_STATUS;
                    this.sendBroadcast(ConnectedBleDeviceProvider
                        .ACTION_CONNECTED_DEVICE
                        , address);
                    break;
                case Constants.DISCONNECTED:
                    this.state = Constants.DISCONNECTED_STATUS;
                    this.sendBroadcast(ConnectedBleDeviceProvider
                        .ACTION_DISCONNECTED_DEVICE
                        , address);
                    break;
                default:
                    this.state = Constants.DISCONNECTED_STATUS;
                    this.sendBroadcast(ConnectedBleDeviceProvider
                        .ACTION_DISCONNECTED_DEVICE
                        , address);
                    break;
            }
        },
        throwable -> logger.log(TAG, throwable.getMessage()));
}

public void establishConnectionAndReceiveData() {
    connection = rxBleDevice
        .establishConnection(false)
        .flatMap(rxBleConnection -> rxBleConnection

```

```

    . setupNotification(UUID
        . fromString(ADDRESS_PATTERN)))
    . doOnNext(notificationObservable -> {})
    . flatMap(notificationObservable ->
        notificationObservable)
    . subscribe( bytes ->
        sendBroadcast(ConnectedBleDeviceProvider
            .ACTION_NEW_DATA_AVAILABLE,
            rxBleDevice.getMacAddress()
            , String.valueOf(Ints.convertDataToInt(bytes))), 
        throwable -> logger.log(TAG,
            throwable.getCause() + " " +
            throwable.getMessage())));
}
}

```

Źródło własne

Również w przypadku pobierania danych z serwera klasa `ApiConnectionProviderImpl` wykorzystuje API RxJava. Sam proces jest subskrybowany na wątku pochodząącym z klasy `Schedulers`, natomiast otrzymane dane analizowane i wyświetlane są w wątku głównym aplikacji `AndroidSchedulers.mainThread()` (Listing 6.4.).

Listing 6.4: Kod pobierania danych z sensorów urządzeń IoT z serwera.

```

@Override
public void getInternetDevice() {
    ApiConnection apiConnection
    = retrofit.create(ApiConnection.class);
    apiConnection.getAllInternetDevice(TOKEN_PATTERN
        + ConfigApp.TOKEN)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribeWith(new DisposableSingleObserver
<List<InternetDeviceWrapper>>(){
        @Override
        public void onSuccess(List<InternetDeviceWrapper>
            internetDeviceWrappers){
            logger.log(TAG, "Internet device loaded, size: " +
                internetDeviceWrappers.size());
            if(listener!=null) listener.internetDeviceLoaded
                (internetDeviceWrappers);
        }
        @Override
        public void onError(Throwable e) {
            logger.log(TAG, e.getMessage());
            if(listener!=null)
                listener.internetDeviceLoadedError(e.getMessage());
        }});
}

```

Źródło własne

W celu określenia lokalizacji użytkownika aplikacji zaimplementowałem klasę `GpsProviderImpl` (Listing 6.5.). Dane o nowej lokalizacji są dostarczane asynchronicznie, zawsze wtedy, gdy użytkownik przemieści się. Z tego właśnie powodu wykorzystałem instancję generycznej klasy `PublishSubject<T>`. Pozwala ona emitować asynchronicznie dane przy użyciu metody `<T> void onNext(T t)`. Wszystkie obiekty `Observer<T>`, które subskrybują obiekt emitujący dane, w momencie wyemitowania nowych danych wywołują metodę `<T> void onNext(T t)`. Przykład obiektu subskrybującego lokalizację znajduje się w klasie `BluetoothService` (Listing 6.6.)

Listing 6.5: Kod klasy odpowiadającej za określenie lokalizacji użytkownika.

```
public class GpsProviderImpl implements GpsProvider {
    @Inject
    public Context context;
    @Inject
    public LocationManager locationManager;
    @Inject
    ConfigApp configApp;
    @Inject
    Logger logger;
    private String providerGPS;
    private Observer<Location> observer;
    private PublishSubject<Location> ps = PublishSubject.create();

    LocationListener locationListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            ps.onNext(location);
            logger.log(TAG, "updateLocation");
        }
        {...}
    };
    @Inject
    public GpsProviderImpl() {this.observer = null;}
    public void setGpsListener(Observer<Location> observer) {
        if (observer != null) {
            this.observer = observer;
            ps.subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(observer);
            providerGPS = this.getProviderName();
            locationManager.requestLocationUpdates
                (providerGPS, 0, 0, locationListener);
        }
    }
    @Override
    public void removeGpsListener() {
        if (observer != null) {
            locationManager.removeUpdates(locationListener);
        }
    }
}
```

```

    ps.onComplete();
    this.observer = null;
}
}

private String getProviderName() {
    Criteria criteria = new Criteria();
    criteria.setPowerRequirement(Criteria.POWER_MEDIUM);
    criteria.setAccuracy(Criteria.ACCURACY_MEDIUM);
    criteria.setSpeedRequired(true);
    criteria.setSpeedAccuracy(Criteria.ACCEPTABLE_MEDIUM);
    criteria.setBearingRequired(false);
    criteria.setAltitudeRequired(false);
    criteria.setCostAllowed(false);
    String providerName =
        locationManager.getBestProvider(criteria, true);
    logger.log(TAG, providerName);
    return providerName;
}
}

```

Źródło własne

Listing 6.6: Kod klasy serwisu - fragment subskrybowania na zmiany lokalizacji.

```

public class BluetoothService extends Service
    implements ApiConnectionListener{
    ...
    private void createGpsObserver() {
        logger.log(TAG, "GPS observer created");
        gpsObserver = new Observer<Location>() {
            @Override
            public void onSubscribe(Disposable d) { ... }

            @Override
            public void onNext(Location location) {
                Intent intent =
                    new Intent(GpsProvider.ACTION_LOCATION_CHANGED);
                intent.
                    putExtra(GpsProvider.KEY_LOCATION_CHANGED, location);
                sendCustomBroadcast(intent);
            }

            @Override
            public void onError(Throwable e)
            { logger.log(TAG, " Error received" + e.toString()); }

            @Override
            public void onComplete()
            { logger.log(TAG, "All data emitted."); }
        };
    }
}

```

Źródło własne

Klasa `ArManagerImpl` odpowiada za zażądzamoe danymi pochodząymi z urządzeń Bluetooth, Serwera oraz lokalizacji. Wszystkie dane otrzymywane są za pośrednictwem zarejestrowanych `BroadcastReveiver`: `GpsBroadcastReceiver`, `InternetBroadcastReceiver`, `BleBroadcastReceiver` oraz listenerów. Ponadto w klasie tej powoływana jest instancja obiektu `MyRender`, który odpowiada za renderowanie widoku AR. Każdorazowo po zmianie danych widok jest odświeżany poprzez wywołanie odpowiedniej metody. Ponadto, w klasie tej znajduje się metoda `setPosition(float azimuthDegress, float pithcDegress)`. Jej zadaniem jest odpowiednie ustawnienie kamery w zbudowanej scenie w technologii OpenGL. Kompletną implementację klasy `MyRender` zaprezentuję na listingu 6.8.

Listing 6.7: Kod klasy zarządzającej widokiem AR.

```
public class ArManagerImpl implements ArManager,
| GpsLocationListener, InternetConnectionListener,
| BleConnectionListener {
|
| @Inject
| GpsBroadcastReceiver gpsBroadcastReceiver;
| @Inject
| InternetBroadcastReceiver internetBroadcastReceiver;
| @Inject
| BleBroadcastReceiver bleBroadcastReceiver;
| @Inject
| Logger logger;
| @Inject
| Context context;
| private MyRender myRender;
|
| @Inject
| public ArManagerImpl() {}
|
| @Override
| public void setupListener() {
|     gpsBroadcastReceiver.setUpListener(this);
|     internetBroadcastReceiver.setUpListener(this);
|     bleBroadcastReceiver.setUpListener(this);
| }
|
| @Override
| public void removeListener() {
|     gpsBroadcastReceiver.removeListener();
|     internetBroadcastReceiver.removeListener();
|     bleBroadcastReceiver.removeListener();
| }
|
| @Override
| public void setPosition( float azimuthDegrees ,
| float pitchDegrees ) {
|     myRender.setX(azimuthDegrees);
|     myRender.setY(360 - pitchDegrees );
| }
|
| @Override
| public MyRender createRender( List<IoTDevice> iotDevices ) {
|     myRender = new MyRender(iotDevices , context );
| }
```

```

    return myRender;
}

@Override
public void changeGpsLocation(Location location) {
    logger.log(TAG, "Position changed");
    myRender.updateUserLocation(location);
}

@Override
public void internetDeviceLoadedReceive
(List<InternetDeviceWrapper>
internetDeviceWrapperList) {
    myRender.updateInternetDevice(internetDeviceWrapperList);
}

@Override
public void internetDeviceLoadedErrorReceive
(String errorMassage) {
    myRender.setAllInternetDeviceOffline();
}

@Override
public void changeBleDeviceConnectionStatus(String address,
int status) {
    myRender.updateBleDeviceStatus(address, status);
}

@Override
public void changeBleDeviceData(String address,
String data) {
    myRender.updateBleDeviceData(address, data);
}
}

```

Źródło własne

Klasą, która generuje widok AR, jest `MyRender`. Główna funkcja `onDrawFrame(GL10 gl)` odpowiada za budowanie sceny wraz z umiejscowieniem kamery. W celu określenia kąta w kierunku którego zwrócona powinna być kamera, wykorzystałem współrzędne sferyczne. Wynika to z faktu, iż scena ma kształt sfery, kamera umieszczona jest wewnątrz jej środka, a obiekty umieszczone są na jej powierzchni. Wraz z zmianą położenia urządzenia mobilnego w przestrzeni obrana jest kamera na podstawie wyliczonego kąta azymutu oraz pułapu. Takie podejście gwarantuje zbliżone właściwości percepcyjne do odbieranego naturalnego świata.

Listing 6.8: Kod klasy `MyRender`.

```

public class MyRender implements GLSurfaceView.Renderer {

    ...
    private final float[] mMVPMatrix = new float[16];
    private final float[] mProjectionMatrix = new float[16];
    private final float[] mViewMatrix = new float[16];

```

```

| private final float eyeX = 0.0f;
| private final float eyeY = 0.0f;
| private final float eyeZ = 0.0f;
|
| private final float upX = 0.0f;
| private final float upY = 1.0f;
| private final float upZ = 0.0f;
|
| public MyRender( List<IoTDevice> IoTDevices , Context context )
| { ... }
|
| @Override
| public void onSurfaceCreated(GL10 gl , EGLConfig config) {
|     GLES20.glClearColor(0.0f , 0.0f , 0.0f , 0.0f );
|     this.createShapes( IoTDevices );
| }
|
| @Override
| public void onDrawFrame(GL10 gl) {
|     final float R = Constants.SCENE_R;
|     GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT
|         | GLES20.GL_DEPTH_BUFFER_BIT);
|     float x = R * (float) Math.cos(Math.toRadians(Y))
|             * (float) Math.cos(Math.toRadians(X));
|     float z = R * (float) Math.sin(Math.toRadians(X))
|             * (float) Math.cos(Math.toRadians(Y));
|     float y = R * (float) Math.sin(Math.toRadians(Y));
|     Matrix.setLookAtM(mViewMatrix , 0 , eyeX , eyeY , eyeZ , x , y , z ,
|         upX , upY , upZ );
|     Matrix.multiplyMM(mMVPMatrix , 0 , mProjectionMatrix , 0 ,
|         mViewMatrix , 0 );
|     this.drawShapes(gl );
| }
|
| private void drawShapes(GL10 gl) {
|     float[] scratch = new float[16];
|     for ( BleDeviceShape singleFrame : bleDeviceShapes ) {
|         Matrix.multiplyMM(scratch , 0 , mMVPMatrix
|             , 0 , singleFrame.getmModelMatrix() , 0 );
|         singleFrame.draw(scratch , gl );
|     }
|     for ( InternetDeviceShape singleFrame : internetDeviceShapes )
|     {
|         Matrix.multiplyMM(scratch , 0 , mMVPMatrix
|             , 0 , singleFrame.getmModelMatrix() , 0 );
|         singleFrame.draw(scratch , gl );
|     }
| }
| }
```

Źródło własne

Rozdział 7

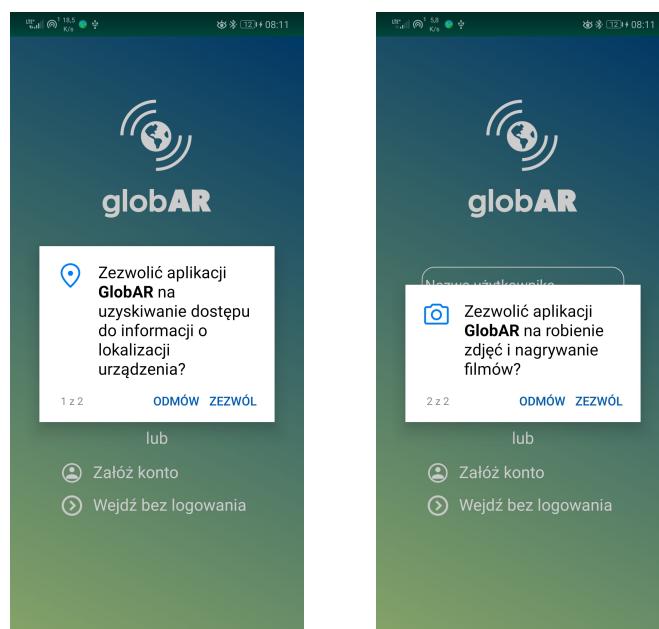
Testy

Treścią tego rozdziału jest wykonanie testu zaprojektowanego systemu pod względem użytkowym.

7.1 Autoryzacja -logowanie

Podczas pierwszego włączenia aplikacji użytkownik jest proszony o przyznanie uprawnień do wykorzystywania (przez aplikację mobilną), informacji o lokalizacji użytkownika (Rys. 37 a). Ponadto konieczny jest również dostęp do aparatu w celu wyświetlenia widoku AR (Rys. 37 b).

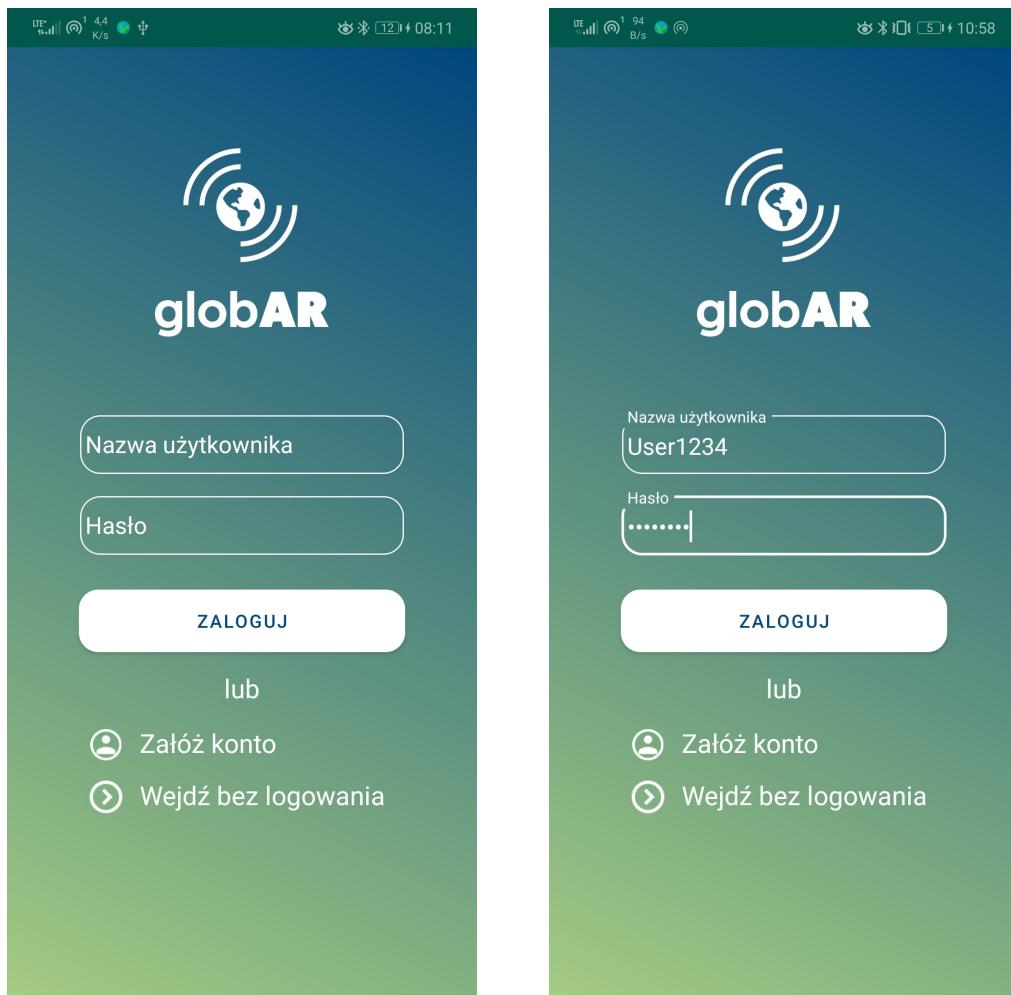
Rysunek 37: Rysunki przedstawiające przydzielanie uprawnień aplikacji.



Źródło własne

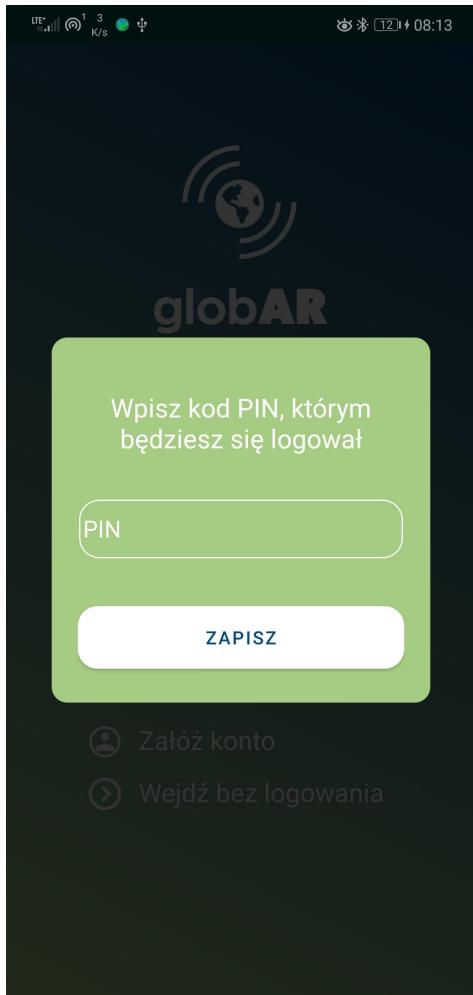
Następnie użytkownik, jeżeli posiada już konto loguje się wpisując swoje dane (Rys. 38), a następnie ustala kod PIN, którym będzie dokonywał procesu autoryzacji przy ponownych uruchomieniach aplikacji (Rys. 39 a). Jeżeli urządzenie mobilne posiada czytnik linii papilarnych, użytkownik może wyrazić zgodę na autoryzację poprzez docisk palca (Rys. 39 b). W przypadku, gdy użytkownik nie ma założonego konta dokonuje procesu rejestracji, który opisałem w podrozdziale 7.1 Autoryzacja - rejestracja nowego użytkownika. Ponadto w każdym z omawianych przypadków użytkownik może pominać proces autoryzacji i korzystać z aplikacji jako gość. W takim jednak przypadku korzystanie z komponentu mapowego oraz pobieranie danych z serwera o odczytach z sensorów urządzeń IoT jest niemożliwe.

Rysunek 38: Rysunki przedstawiające ekran logowania użytkownika.

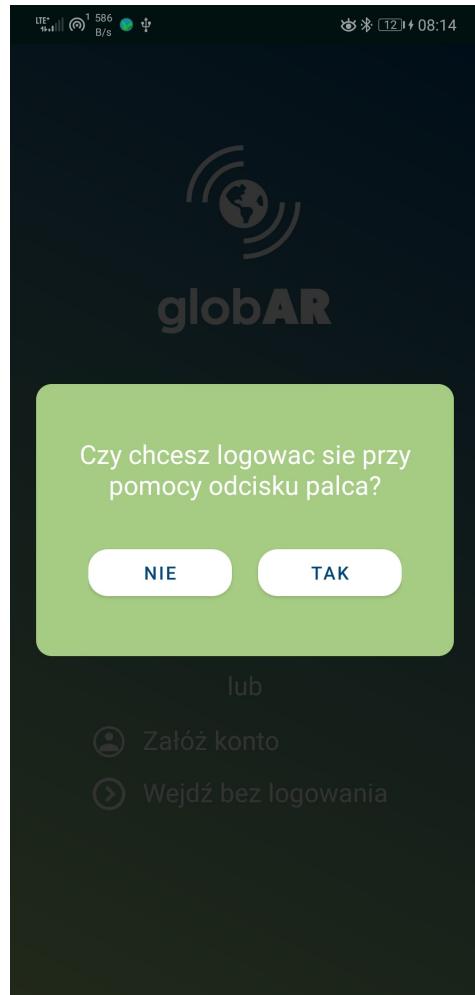


Źródło własne

Rysunek 39: Rysunki przedstawiające ekran wyboru kodu PIN oraz zgodę na autoryzację odciskiem palca.



(a) Personalizacja kodu PIN.

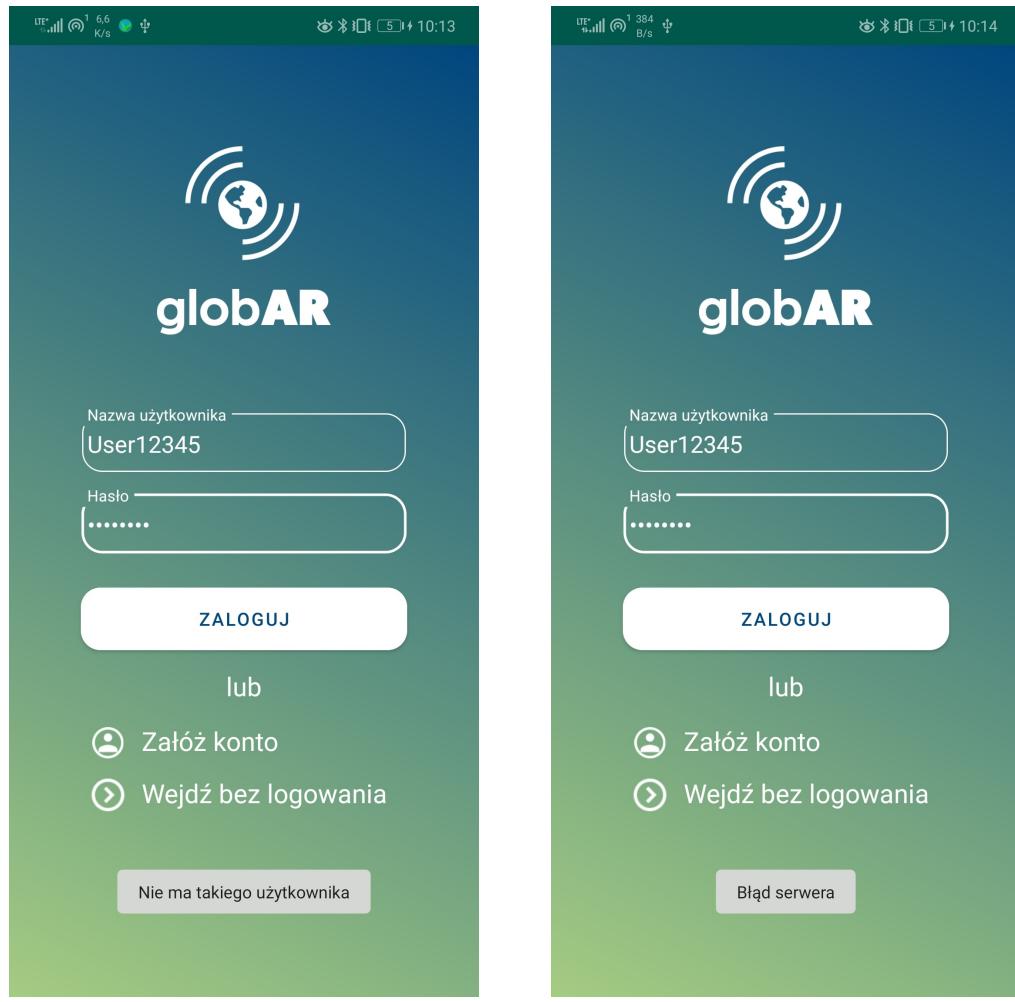


(b) Autoryzację odciskiem palca.

Źródło własne

W przypadku, gdy użytkownik poda błędne dane w procesie logowania, zostają wyświetlane komunikaty informacyjne, np.: Nie ma takiego użytkownika (Rys. 40 a). Natomiast, gdy użytkownik straci połączenie z Internetem wyświetlany zostaje komunikat: Błąd serwera (Rys. 40 b).

Rysunek 40: Rysunki przedstawiające komunikaty błędów.



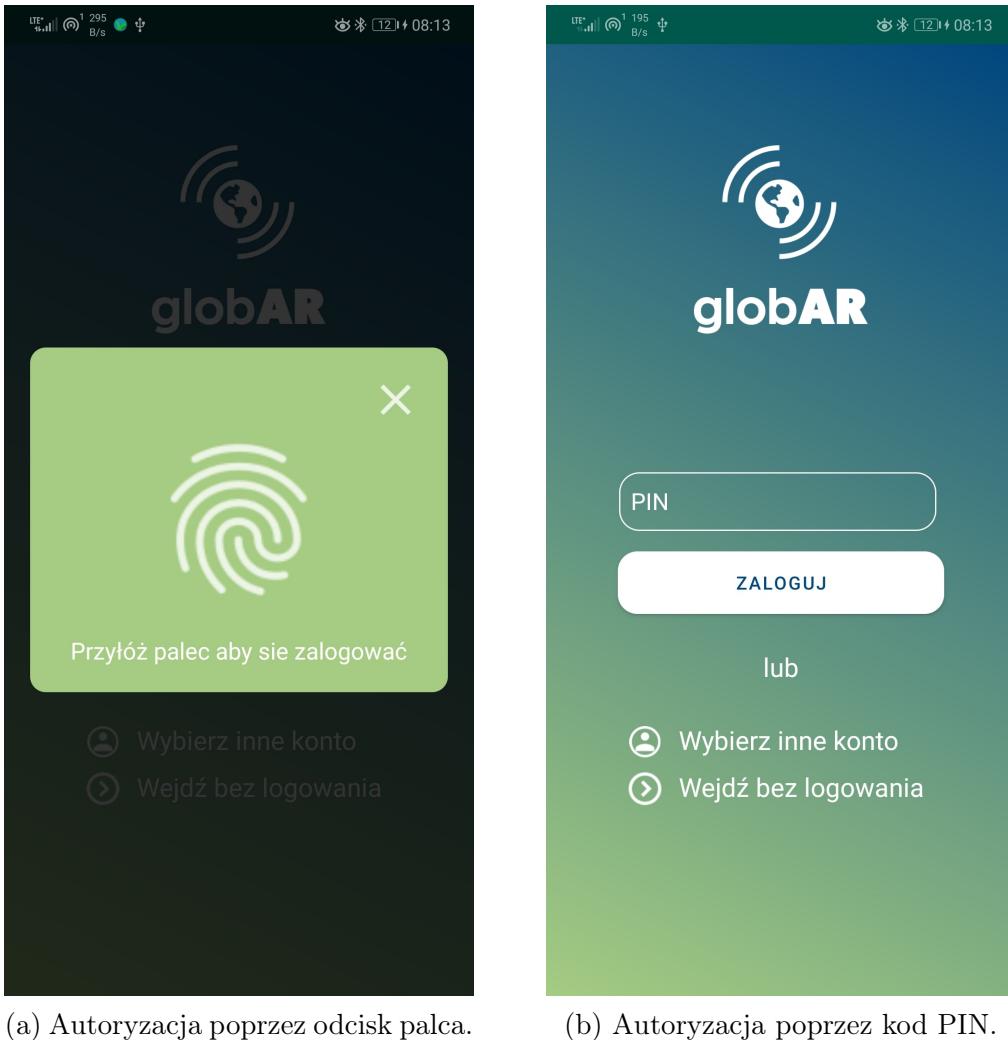
(a) Błędne dane użytkownika.

(b) Błąd serwera.

Źródło własne

Podczas kolejnych uruchomień aplikacji użytkownik, jeżeli wyraził zgodę, dokonuje autoryzację poprzez odciska palca. W przypadku braku zgody logowanie odbywa się poprzez wpisanie kodu PIN (Rys. 41).

Rysunek 41: Rysunki przedstawiające proces autoryzacji podczas kolejnych uruchomień aplikacji.



(a) Autoryzacja poprzez odcisk palca.

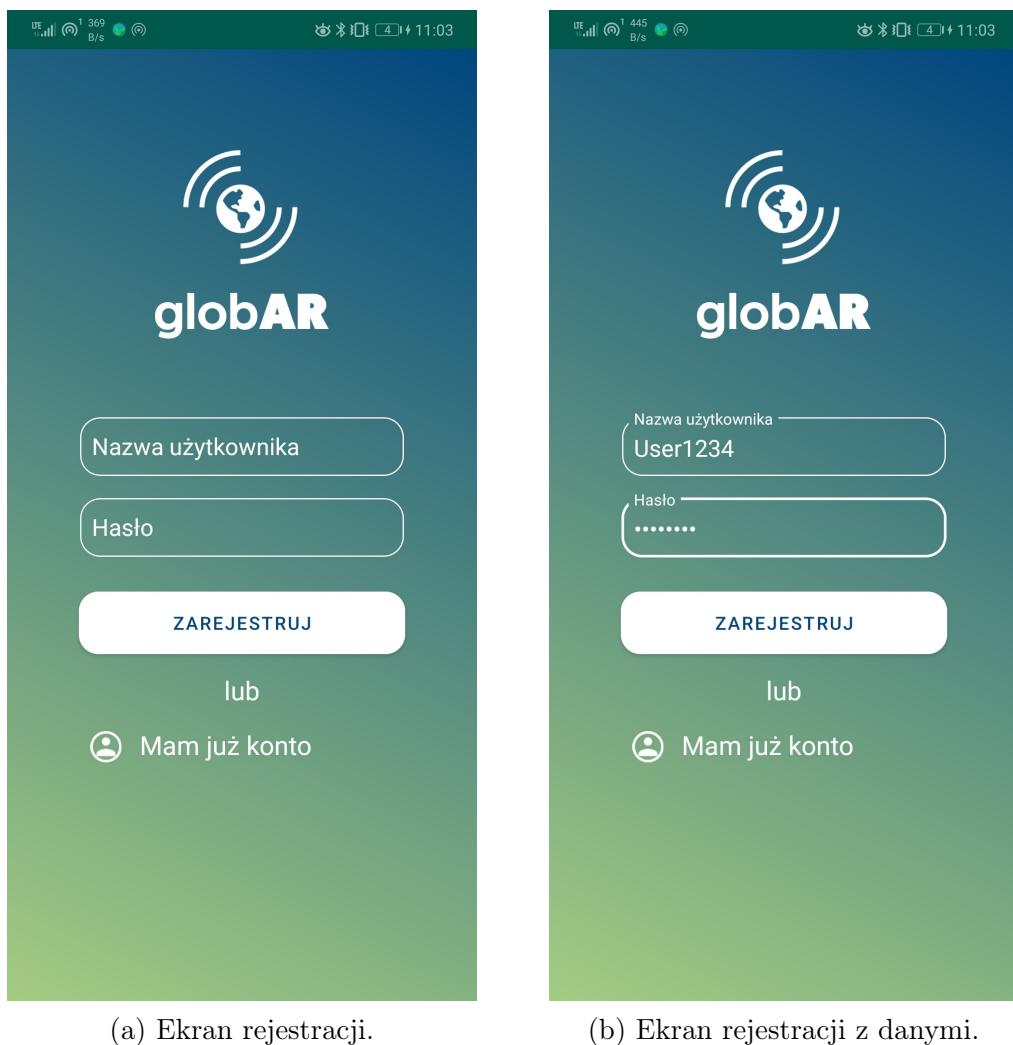
(b) Autoryzacja poprzez kod PIN.

Źródło własne

7.2 Autoryzacja - rejestracja nowego użytkownika

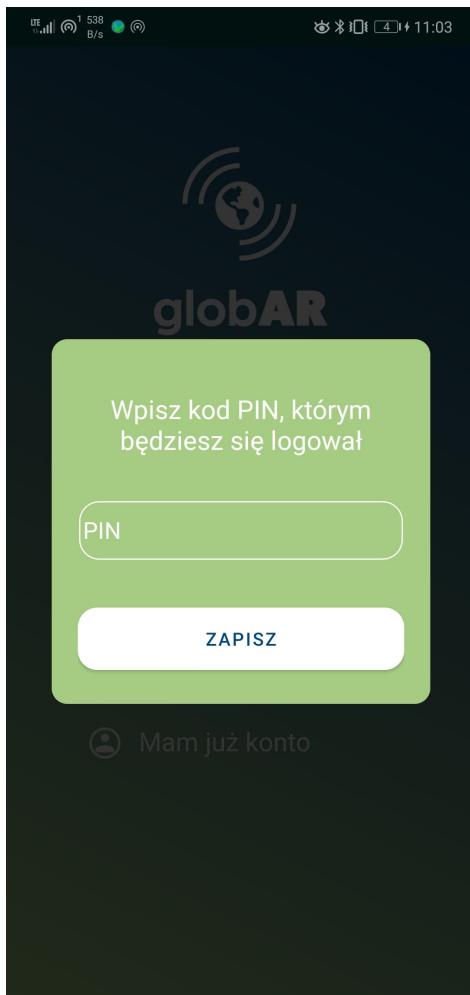
Użytkownik w celu założenia nowego konta musi podać nazwę użytkownika oraz hasło (o długości nie mniejszej niż 6 znaków) (Rys. 42 a, b). Następnie proszony jest o wybranie kodu PIN, którym będzie się logował podczas kolejnych uruchomień aplikacji (Rys. 43 a). W ostatnim etapie, jeżeli urządzenie posiada czytnik linii papilarnych, użytkownik może wyrazić zgodę na autoryzację poprzez odcisk palca (Rys. 43 b).

Rysunek 42: Rysunki przedstawiające proces rejestracji nowego użytkownika.

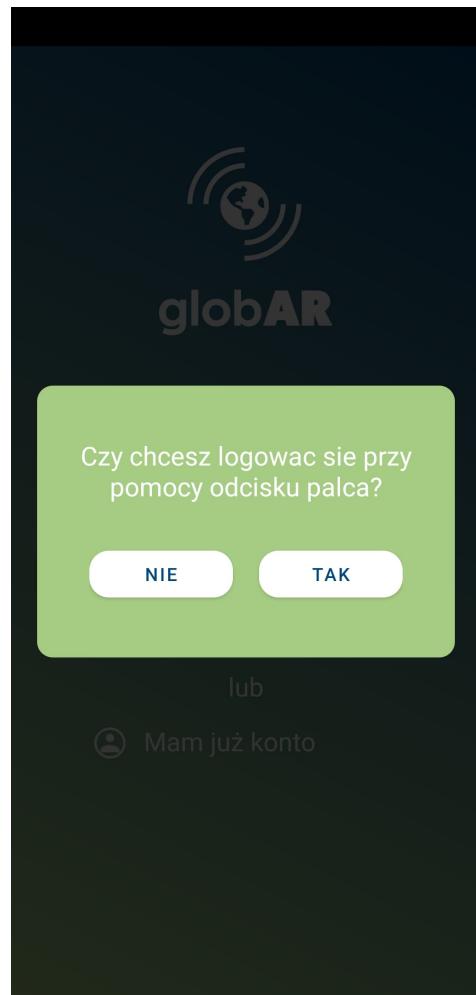


Źródło własne

Rysunek 43: Rysunki przedstawiające personalizację procesu logowania.



(a) Personalizacja kodu PIN.

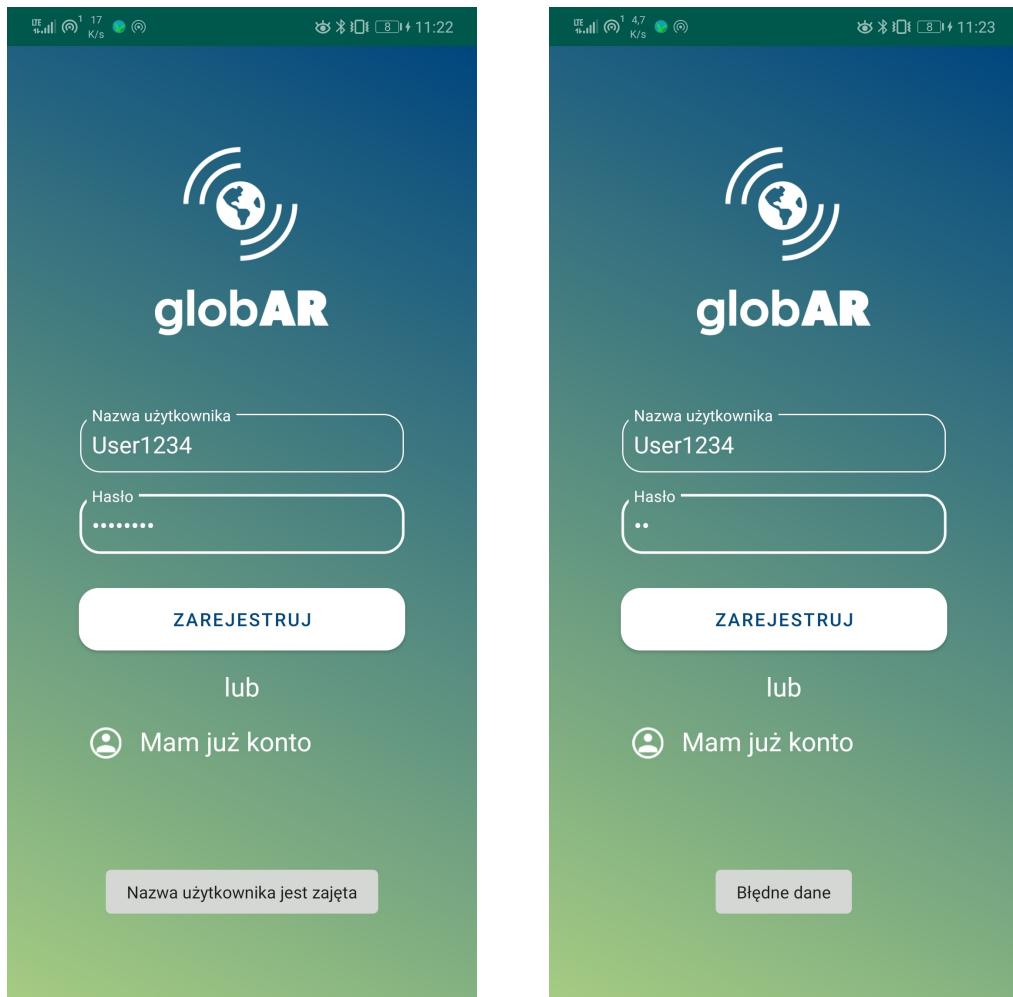


(b) Autoryzacja poprzez odcisk palca.

Źródło własne

W przypadku, gdy użytkownik wybierze nazwę, która jest już zajęta wyświetlony zostaje komunikat: Nazwa użytkownika jest zajęta (Rys 44 a). Natomiast po podaniu hasła o długości mniejszej niż 6 znaków pojawia się komunikat: Błędne dane (Rys. 44 b).

Rysunek 44: Rysunki przedstawiające personalizację procesu logowania.



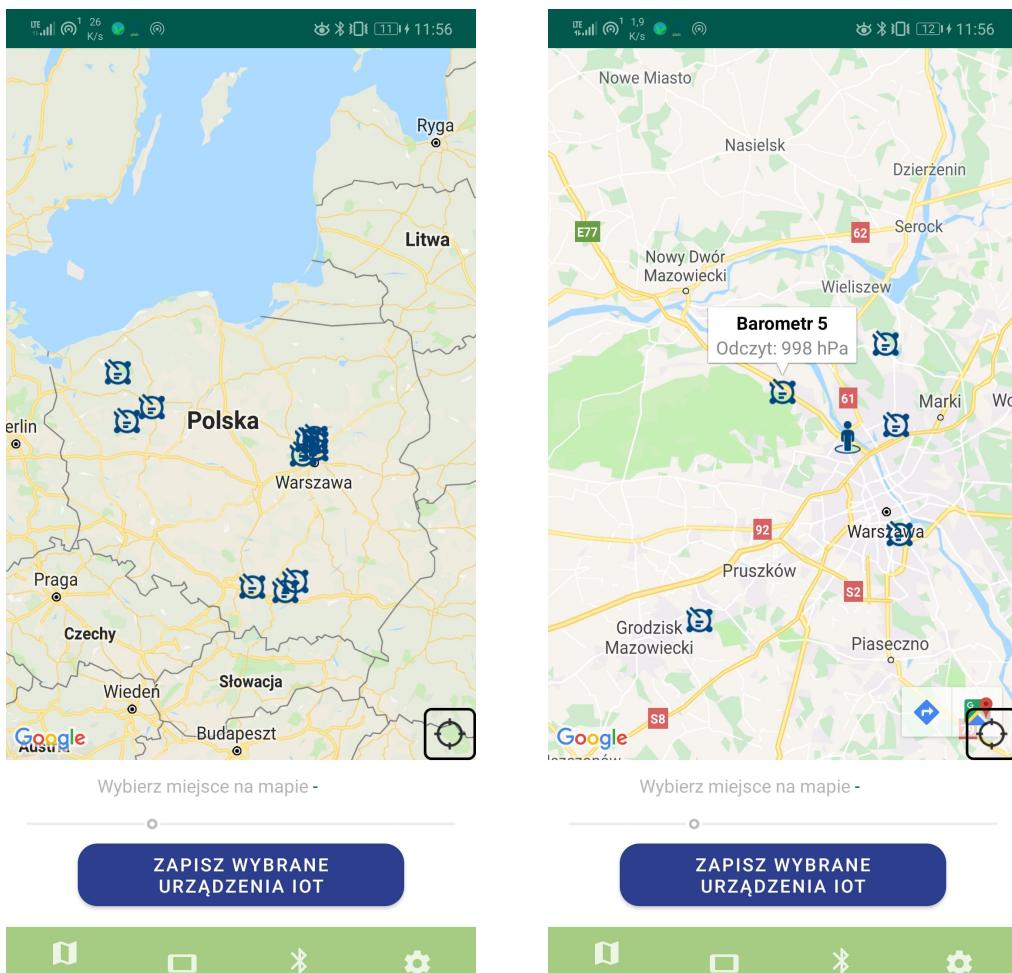
Źródło własne

7.3 Komponent mapowy

Aplikacja mobilna jest zasilana danymi o sensorach IoT z dwóch źródeł. Pierwsze z nich jest to serwer, który agreguje odczyty urządzeń IoT, natomiast drugie to urządzenia Bluetooth znajdujące się w pobliżu. W tym podrozdziale zaprezentuję pozyskiwanie danych z serwera poprzez komponent mapowy.

Po wybraniu przez użytkownika zakładki Mapa, wyświetlona zostaje mapa Google wraz z namięcionymi na nią znacznikami, które reprezentują urządzenia IoT (Rys 44 a). Po przybliżeniu na dany obszar użytkownik ma możliwość sprawdzenia szczegółu odczytu danego urządzenia przez dotknięcie jego ikony (Rys 44 b, Rys 45 a).

Rysunek 45: Rysunki przedstawiające komponent mapowy.



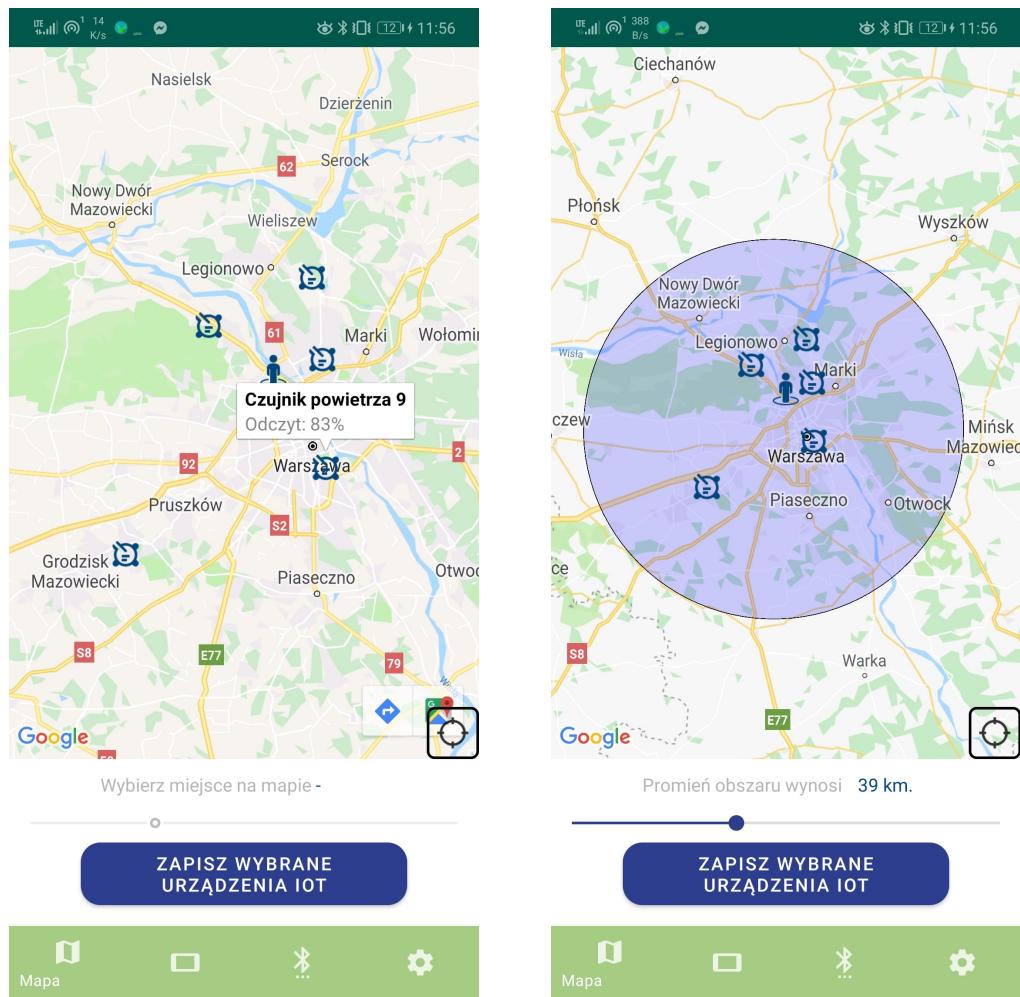
(a) Widok mapy.

(b) Lokalizacja użytkownika.

Źródło własne

Następnie użytkownik wybiera obszar oraz jego powierzchnię, skąd pobrane mają zostać odczyty z sensorów urządzeń IoT (Rys. 46 b). Wybór jest wizualizowany poprzez okrąg o wybranym promieniu na mapie. Sensory, które znajdują się w tym okręgu, zostaną zapisane i w dalszej części testu wyświetcone w widoku AR.

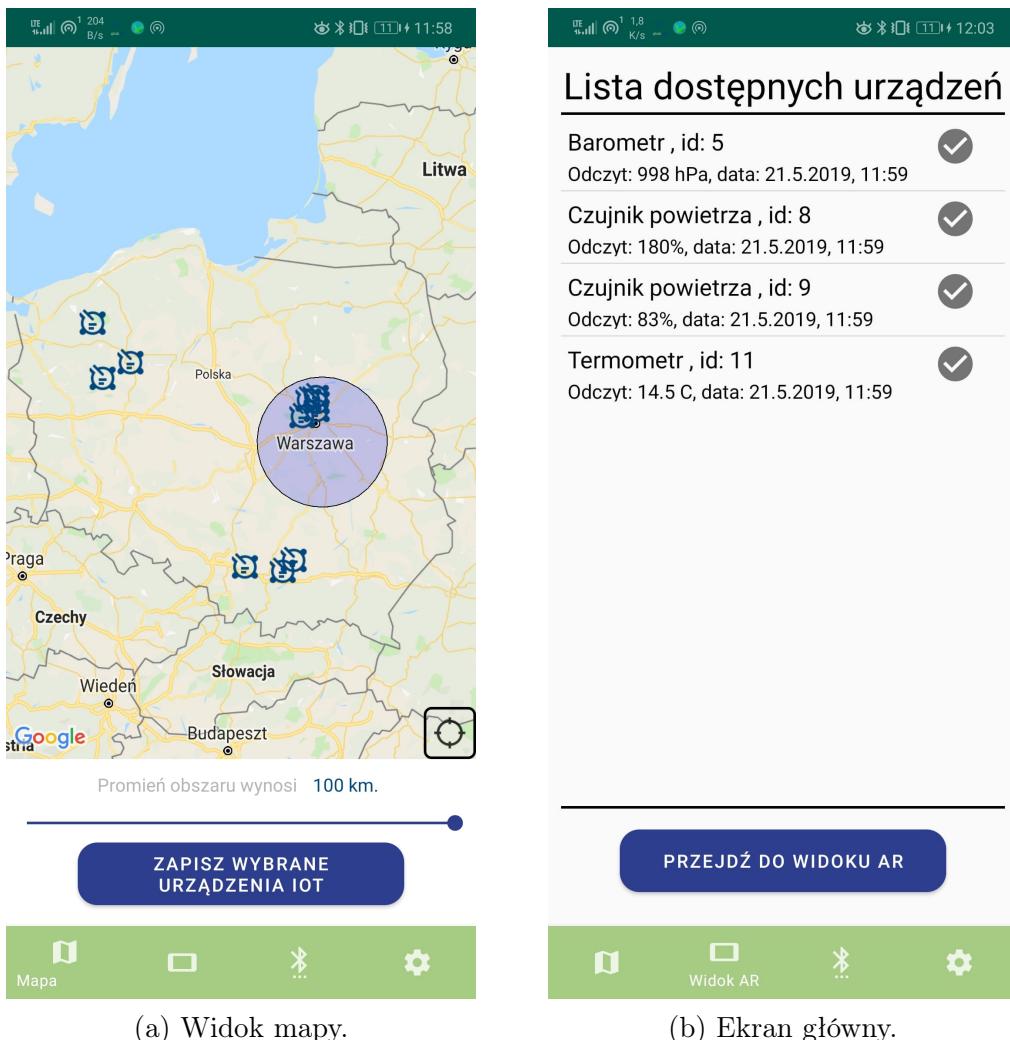
Rysunek 46: Rysunki przedstawiające komponent mapowy.



Źródło własne

Rys. 47 a przedstawia oddaloną mapę z zaznaczonym obszarem przez użytkownika, który ustalił jego promień na 100km. Na ostatnim rysunku (Rys 47 b) widać główny ekran, który zawiera listę zapisanych urządzeń IoT. W opisie widoczna jest nazwa urządzenia, odczyt oraz data wraz z godziną odczytu. Po prawej stronie każdego elementu znajduje się ikona symbolizująca status danego urządzenia.

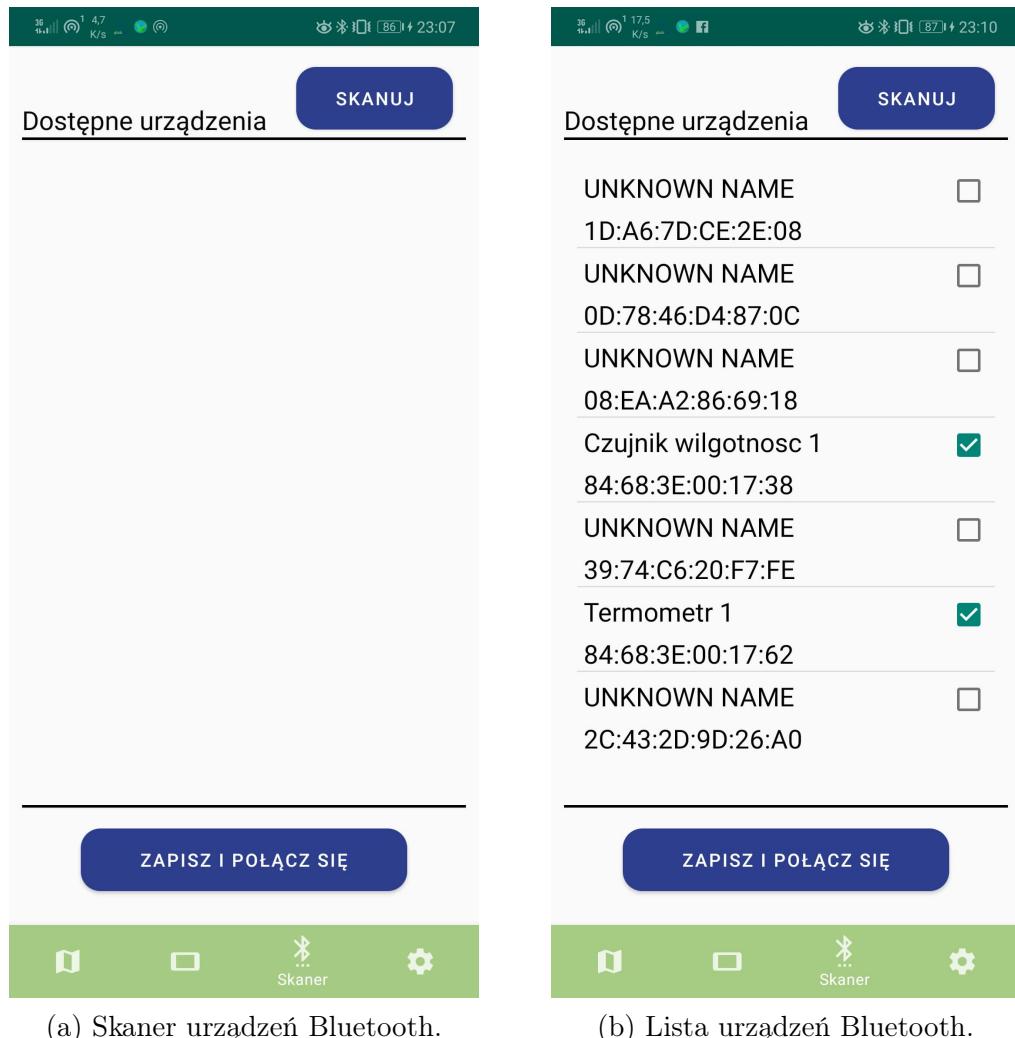
Rysunek 47: Rysunki przedstawiające komponent mapowy oraz ekran główny.



7.4 Wyszukanie urządzeń IoT Bluetooth

Kolejnym źródłem danych dla aplikacji mobilnej są urządzenia IoT Bluetooth. Użytkownik w pierwszej kolejności przechodzi do ekranu Skanera (Rys. 48 a). Następnie, po włączeniu skanera urządzeń Bluetooth, znalezione urządzenia są dodawane do listy (Rys. 48 b). Użytkownik poprzez dotknięcie nazwy danego urządzenia może wybrać do zapisania (Rys. 48 b).

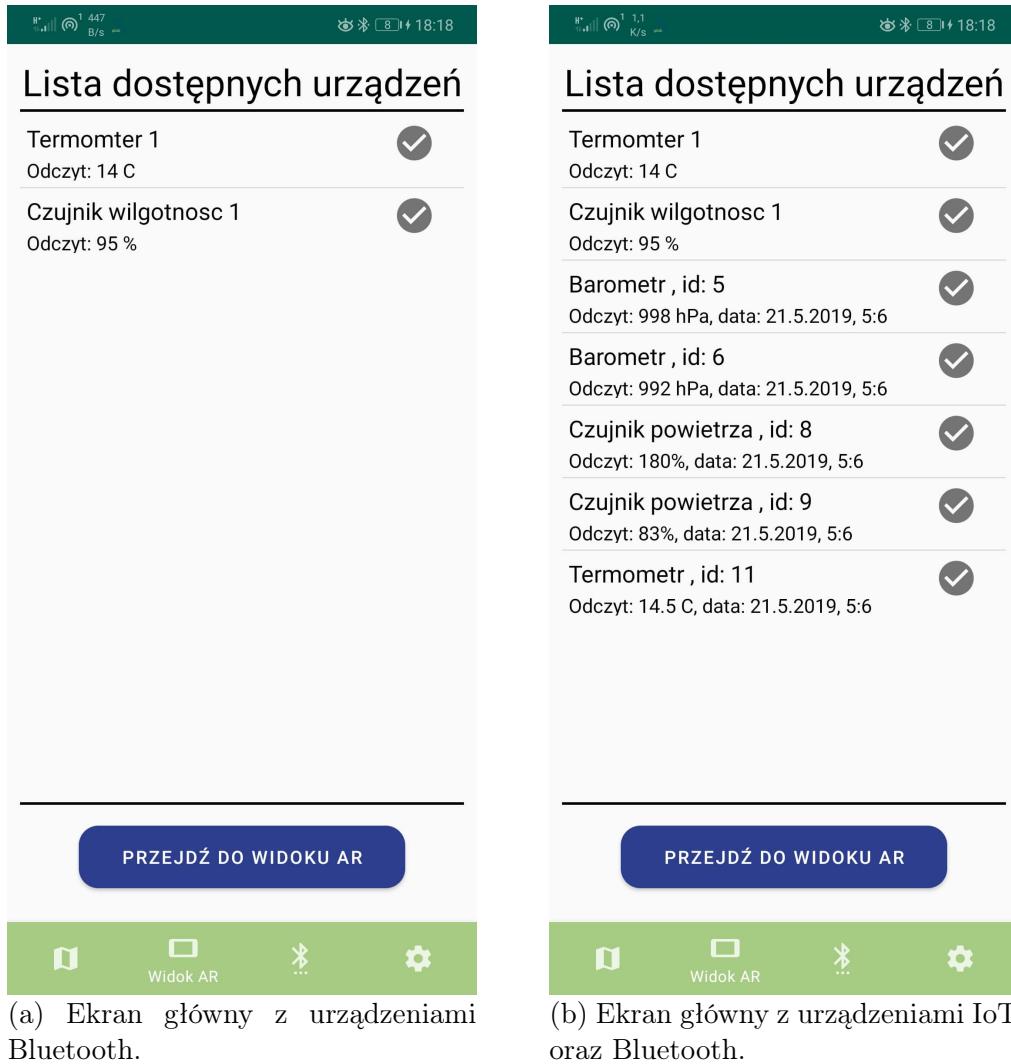
Rysunek 48: Rysunki przedstawiające skaner urządzeń IoT Bluetooth.



Źródło własne

Następnie po przejściu do ekranu głównego, lista urządzeń IoT zostaje zapełniona przez urządzenia pochodzące zarówno z serwera jak i z wcześniejszego wyszukania urządzeń Bluetooth (Rys. 49 a, b).

Rysunek 49: Rysunki przedstawiające ekran główny z urządzeniami IoT z serwera oraz Bluetooth .



(a) Ekran główny z urządzeniami Bluetooth.

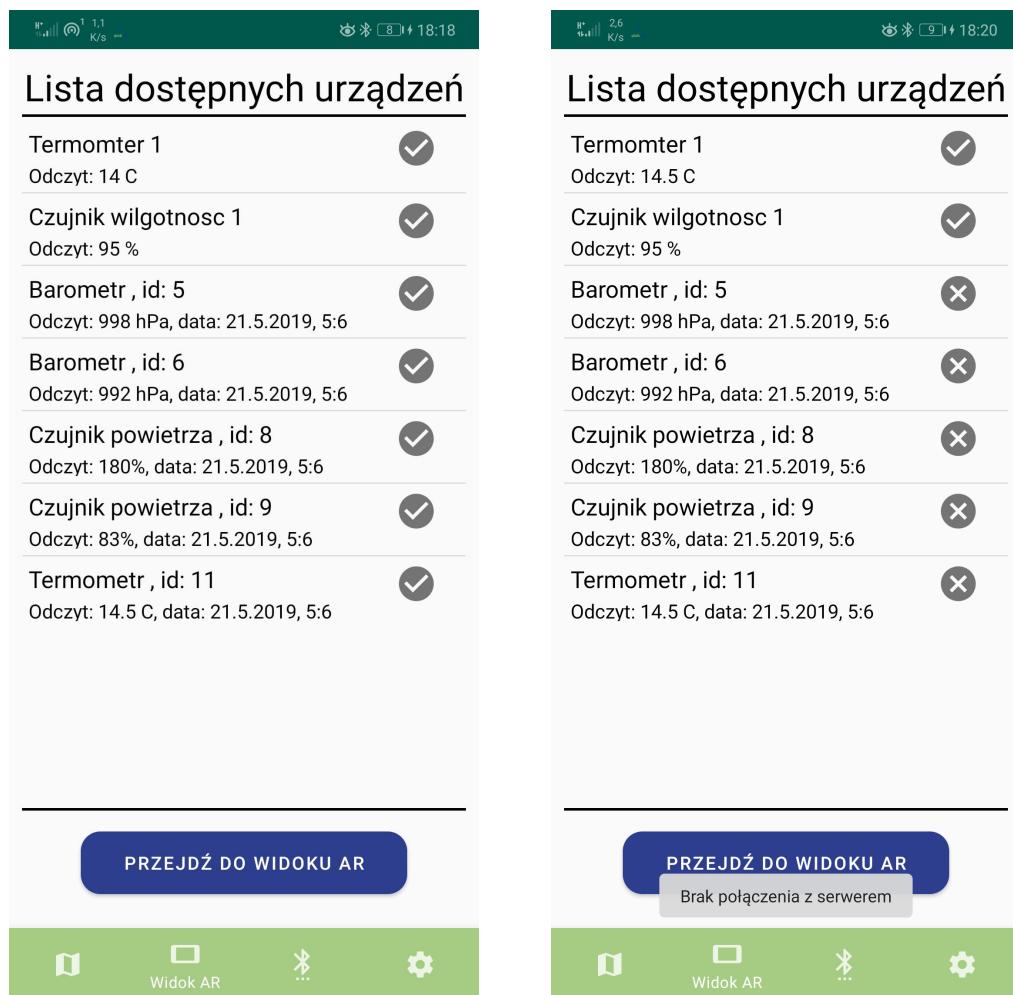
(b) Ekran główny z urządzeniami IoT oraz Bluetooth.

Źródło własne

7.5 Widok główny

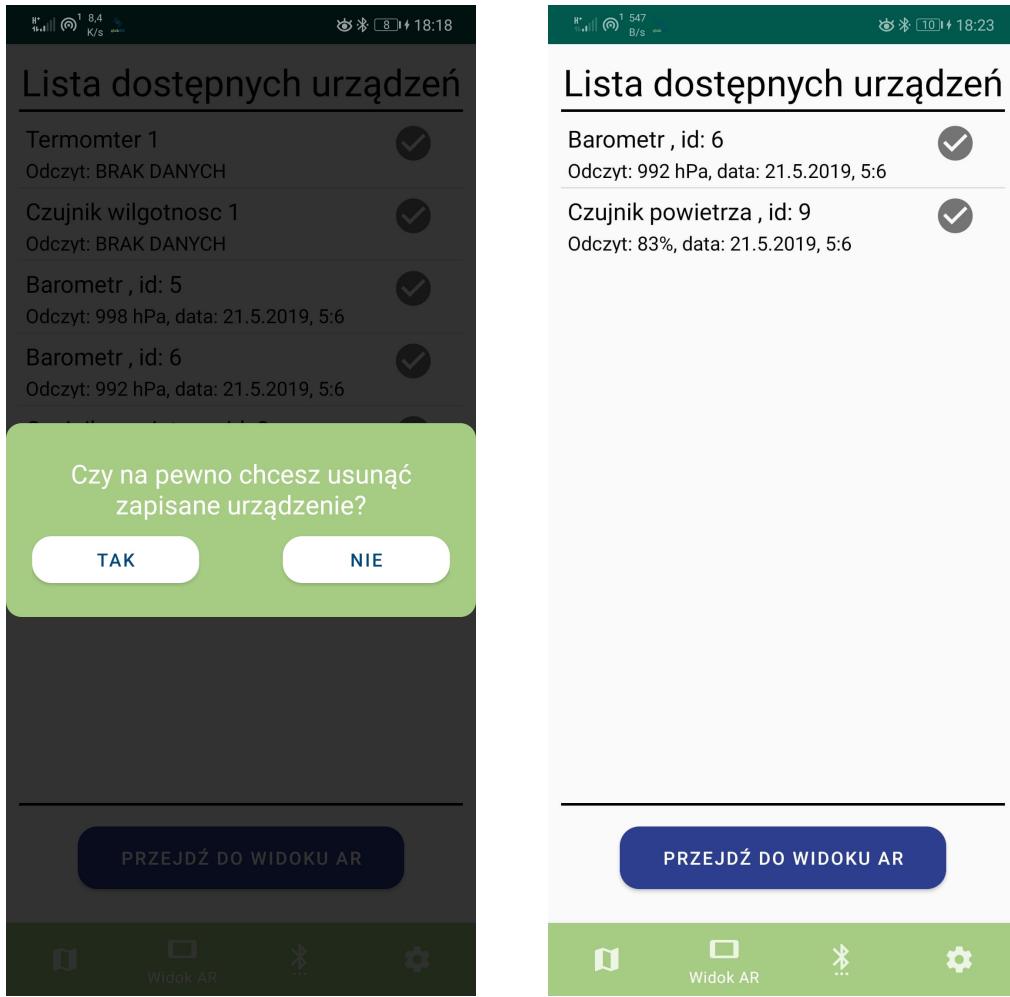
Ekran główny zawiera listę urządzeń IoT pochodzących z serwera oraz urządzeń Bluetooth. Obok nazwy urządzenia wyświetlany jest stan połączenia, a pod spodem odczyt z czujnika (Rys. 50 a). W przypadku rozłączenia z serwerem wyświetlany zostaje komunikat o błędzie oraz zmienia się ikona statusu połączenia (Rys. 50 b). Użytkownik z poziomu ekranu głównego posiada możliwość usuwania zapisanych urządzeń, po uprzednim potwierdzeniu czynności (Rys. 51 a, b).

Rysunek 50: Rysunki przedstawiające ekran główny aplikacji.



Źródło własne

Rysunek 51: Rysunki przedstawiające ekran główny aplikacji oraz okno usuwania urządzenia IoT.



źródło własne

7.6 Widok AR

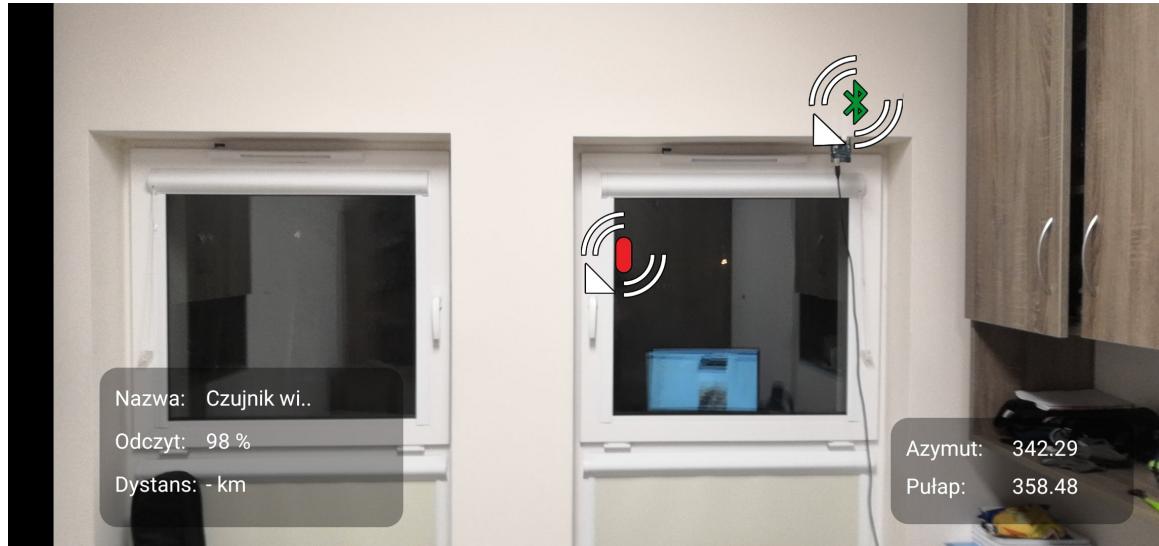
Najważniejszym elementem aplikacji mobilnej jest widok AR. Po uprzednio zdefiniowanym zbiorze urządzeń IoT do wyświetlenia, użytkownik włącza widok AR. W lewym dolnym rogu wyświetlane zostają dane dotyczące orientacji w przestrzeni, azymut i pułap, natomiast w prawym dolnym rogu szczegółowe odczyty danych z sensorów (Rys. 52). Ikoną Bluetooth zaznaczone zostały urządzenia IoT Bluetooth, natomiast zaokrąglonym prostokątem urządzenia pochodzące z serwera. Kolor zielony ikony oznacza aktywne połączenie z sensorem (Rys. 52), natomiast czerwony kolor oznacza brak połączenia (Rys. 53).

Rysunek 52: Widok AR, urządzenia aktywnie połączone.



Źródło własne

Rysunek 53: Widok AR, urządzenie Bluetooth aktywnie połączone, urządzenie z serwera rozłączone.



Źródło własne

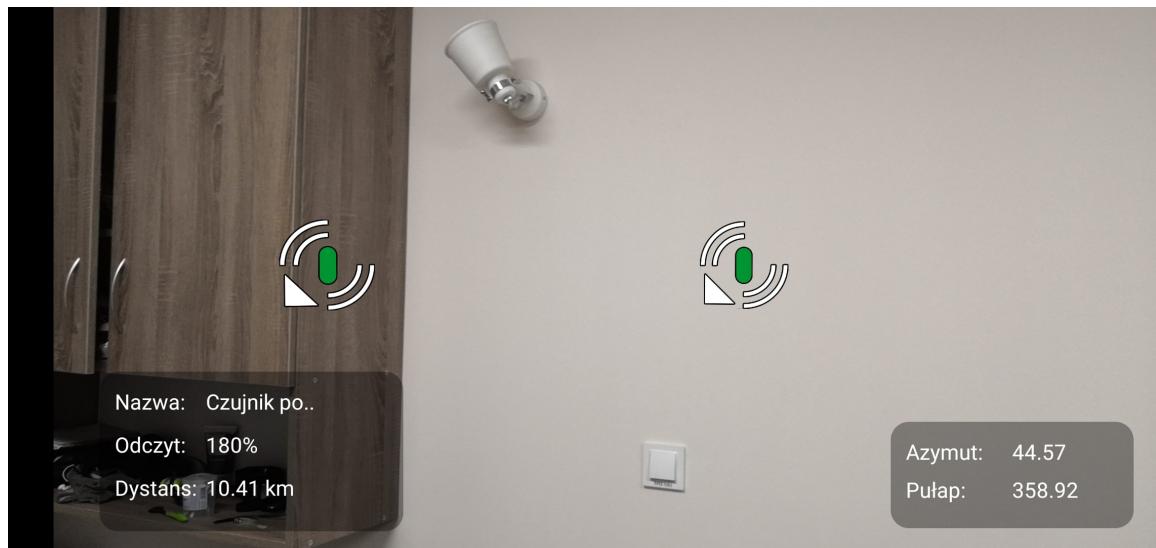
Dla urządzeń pochodzących z serwera określony został dystans, jaki dzieli go od użytkownika (Rys. 54, 55).

Rysunek 54: Widok AR, określenie dystansu do urządzenia IoT.



Źródło własne

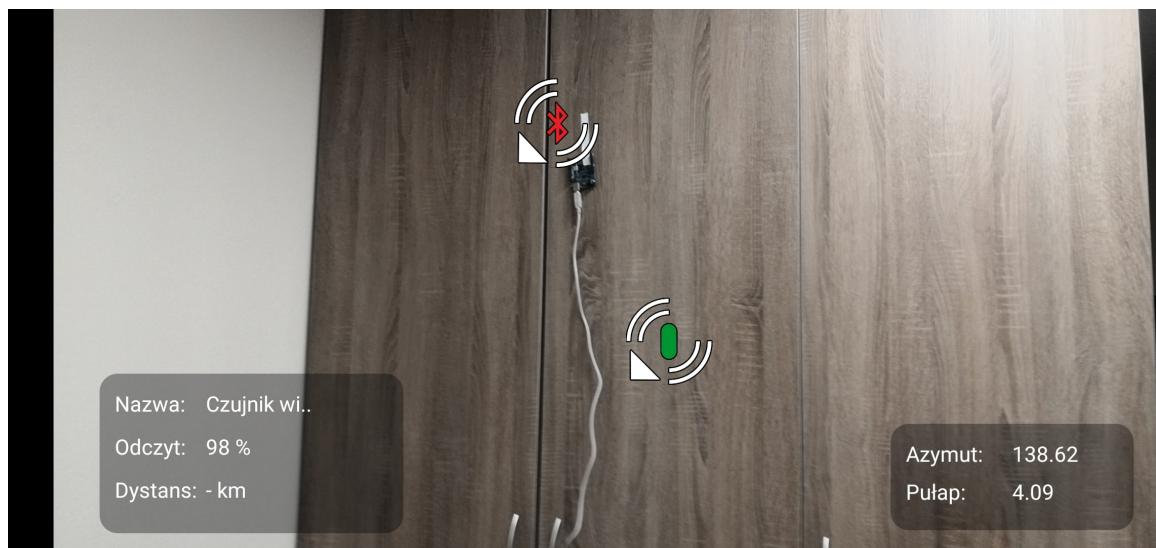
Rysunek 55: Widok AR, urządzenia IoT pochodzące z serwera.



Źródło własne

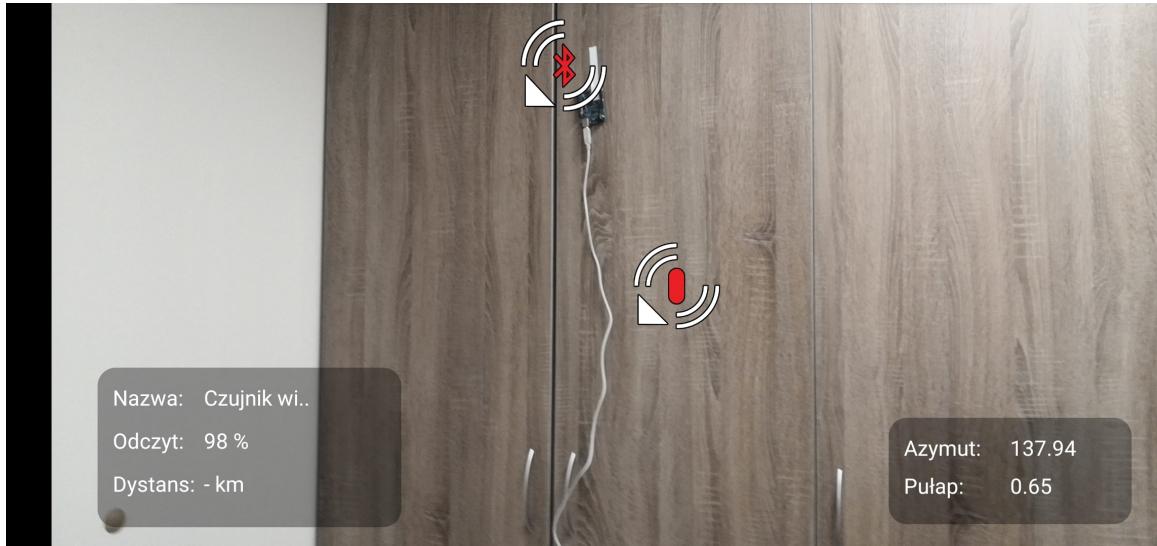
Aplikacja mobilna posiada mechanizm automatycznego odnawiania połączenia z urządzeniami Bluetooth oraz odpytuje serwer, z zadanym interwałem czasowym, o aktualne odczyty z sensorów urządzeń IoT (Rys. 56, 57, 58).

Rysunek 56: Widok AR, urządzenia Bluetooth rozłączone.



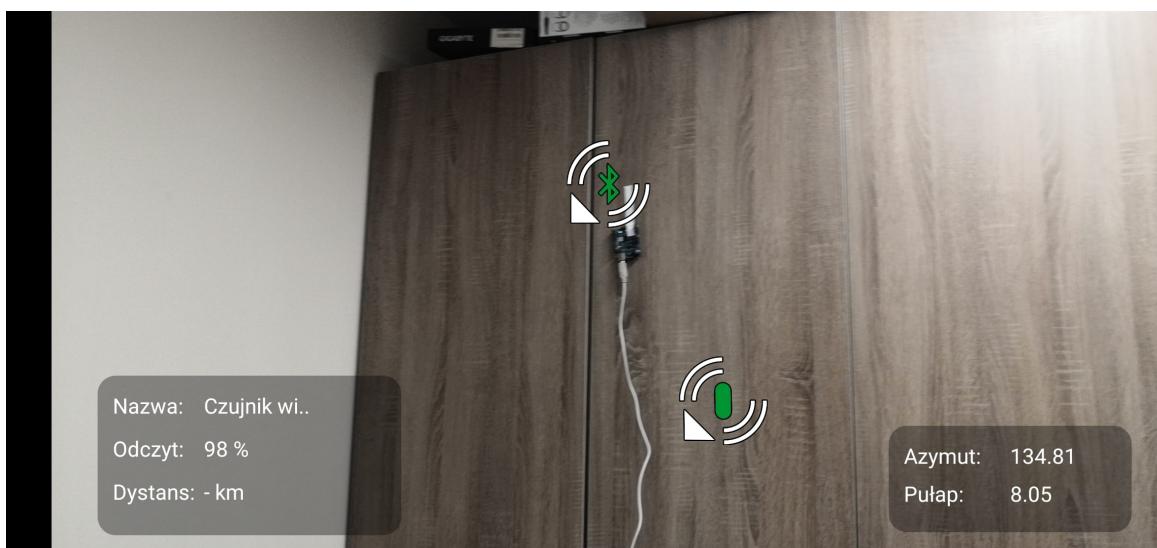
Źródło własne

Rysunek 57: Widok AR, urządzenia IoT rozłączone.



Źródło własne

Rysunek 58: Widok AR, odnowienie połączenia z urządzeniami IoT.



Źródło własne

Rozdział 8

Zakończenie

Celem pracy było zaprojektowanie oraz implementacja systemu informatycznego, który wykorzystując technologię Rozszerzonej Rzeczywistości pozwalałby użytkownikowi na wizualizację danych w czasie rzeczywistym, pochodzących z sensorów urządzeń IoT. W pierwszej kolejności dokonałem przeglądu dostępnej technologii i zdecydowałem się na wykorzystanie urządzenia mobilnego, jako urządzenia do wizualizacji danych. Wybrałem tym celu technologię AR markless.

Zaprojektowany system składa się nie tylko z urządzenia mobilnego, które jest jego głównym elementem, ale również z serwera oraz urządzeń IoT. Serwer pełni rolę agregowania i udostępniania danych z odległych sensorów oraz autoryzacji użytkowników. Natomiast urządzenia IoT można podzielić na te wykorzystujące technologię Bluetooth oraz te połączone z serwerem. Całość tworzy kompletny system informatyczny, w którym użytkownik sam może decydować, jakie zawartości mają zostać wyświetlane w widoku AR.

W celu zachowania czystości kodu oraz zasad SOLID wykorzystałem między innymi programowanie reaktywne, które znacznie pomogło mi podczas wykonywania asynchronicznych zadań. Poprzez możliwość wyboru wątków, na których dane mają być emitowane oraz subskrybowane, mogłem zarządzać zapytaniami do bazy danych oraz serwera. Dodatkowym atutem programowania reaktywnego, w kontekście zarządzania połączeniem z urządzeniami Bluetooth oraz monitorowaniem jego stanu, okazała się biblioteka `rxandroidble2.RxBleDevice`. Biblioteka ta dostarcza mechanizmy nawiązywania połączenia między urządzeniem mobilnym a urządzeniem IoT oraz monitorowaniem stanu tego połączenia. Kolejną technologią, jaką wykorzystałem, był mechanizm wstrzykiwania zależności. Takie podejście pozwala na separację części kodu oraz wydzielenie niezależnych modułów aplikacji. Późniejsze modyfikacje danego fragmentu kodu nie wpływają na całokształt aplikacji.

Sam widok Rozszerzonej Rzeczywistości zbudowałem wykorzystując technologię OpenGL ES. Zaprojektowałem scenę w kształcie sfery, na powierzchni której wyświetlane zostały urządzenia IoT. Kamera natomiast znajduje się w środku sceny, a jej obrót wyznaczyłem za pomocą równań biegunowych sfery. Takie rozwiązanie gwarantuje postrzeganie zbudowanej sceny jako naturalnego elementu świata rzeczywistego.

Aplikacja mobilna, dzięki podziałowi na wiele modułów, może w przyszłości zostać rozbudowana, powiększona o dodatkowe funkcjonalności. Mechanizm generowania widoku AR oraz wizualizowania urządzeń IoT wymaga tylko współrzędnych przestrzennych (azymut oraz pułap) oraz wartości odczytów sensorów.

Spis rysunków

1	Urządzenie AR opracowane przez Ivana Sutherlanda	11
2	Projekt "Videoplace"	11
3	Rozszerzona Rzeczywistość w firme Boeing	12
4	Marker AR	14
5	Markerless AR	14
6	Projection-based AR	15
7	Superimposition-based AR	15
8	Urządzenie mobile wykorzystujące AR	17
9	Smart glasses - Meta 2	18
10	Smart glasses - hololens	18
11	Kokpit samolotu	19
12	Soczewki AR	19
13	Diagram przypadków użycia - Urządzenie Mobilne	28
14	Diagram przypadków użycia - Serwer	37
15	Diagram przypadków użycia - Urządzenie IoT Bluetooth	41
16	Diagram przypadków użycia - urządzenie IoT Internet	43
17	Diagram aktywności - autoryzacja	47
18	Diagram aktywności - przygotowanie widoku AR	48
19	Diagram aktywności - widok AR	49
20	Diagram komponentów	50
21	Diagram klas dla pakietu authManager	54
22	Diagram klas dla pakietu bluetoothManager	55
23	Diagram klas dla pakietu motionSensor	56
24	Diagram klas dla pakietu DataBase	57
25	Diagram klas dla pakietu internetManager	58
26	Diagram klas dla pakietu mainView	59
27	Diagram klas dla pakietu bluetoothScannerView	60
28	Diagram klas dla pakietu mapView	61

29	Diagram klas dla pakietu arEngine	62
30	Diagram sekwencji - autoryzacja jako gość.	64
31	Diagram sekwencji - rejestracja nowego użytkownika.	65
32	Diagram sekwencji - logowanie użytkownika.	66
33	Diagram sekwencji - wyszukanie urządzeń IoT z serwera.	67
34	Diagram sekwencji - wyszukanie urządzeń IoT Bluetooth.	68
35	Diagram sekwencji - wyszukanie urządzeń IoT z serwera.	69
36	Diagram wdrożenia	70
37	Rysunki przedstawiające przydzielanie uprawnień aplikacji.	84
38	Rysunki przedstawiające ekran logowania użytkownika.	85
39	Rysunki przedstawiające ekran wyboru kodu PIN oraz zgodę na autoryzację odciskiem palca.	86
40	Rysunki przedstawiające komunikaty błędów.	87
41	Rysunki przedstawiające proces autoryzacji podczas kolejnych uruchomień aplikacji.	88
42	Rysunki przedstawiające proces rejestracji nowego użytkownika.	89
43	Rysunki przedstawiające personalizację procesu logowania.	90
44	Rysunki przedstawiające personalizację procesu logowania.	91
45	Rysunki przedstawiające komponent mapowy.	92
46	Rysunki przedstawiające komponent mapowy.	93
47	Rysunki przedstawiające komponent mapowy oraz ekran główny.	94
48	Rysunki przedstawiające skaner urządzeń IoT Bluetooth.	95
49	Rysunki przedstawiające ekran główny z urządzeniami IoT z serwera oraz Bluetooth	96
50	Rysunki przedstawiające ekran główny aplikacji.	97
51	Rysunki przedstawiające ekran główny aplikacji oraz okno usuwania urządzenia IoT.	98
52	Widok AR, urządzenia aktywnie połączone.	99
53	Widok AR, urządzenie Bluetooth aktywnie połączone, urządzenie z serwera rozłączone.	100
54	Widok AR, określenie dystansu do urządzenia IoT.	100
55	Widok AR, urządzenia IoT pochodzące z serwera.	101
56	Widok AR, urządzenia Bluetooth rozłączone.	101
57	Widok AR, urządzenia IoT rozłączone.	102
58	Widok AR, odnowienie połączenia z urządzeniami IoT.	102

Spis tabel

5.1	Scenariusz UC-1.	29
5.2	Scenariusz UC-2.	29
5.3	Scenariusz UC-3.	31
5.4	Scenariusz UC-4.	31
5.5	Scenariusz UC-4.1.	32
5.6	Scenariusz UC-4.2.	33
5.7	Scenariusz UC-5.1.	34
5.8	Scenariusz UC-6.	35
5.9	Scenariusz UC-6.1.	35
5.10	Scenariusz UC-7.	38
5.11	Scenariusz UC-8.	38
5.12	Scenariusz UC-9.	39
5.13	Scenariusz UC-10.	40
5.14	Scenariusz UC-11.	42
5.15	Scenariusz UC-12.	42
5.16	Scenariusz UC-13.	44
5.17	Wymagania poza funkcjonalne.	45

Spis listingów kodu

6.1	Kod interfejsu SensorManager i jego implementacja.	73
6.2	Kod pobierania i zapisywania danych do bazy danych.	75
6.3	Kod połączenia z urządzeniem Bluetooth oraz monitorowanie jego statusu.	77
6.4	Kod pobierania danych z sensorów urządzeń IoT z serwera.	78
6.5	Kod klasy odpowiadającej za określenie lokalizacji użytkownika.	79
6.6	Kod klasy serwisu - fragment subskrybowania na zmiany lokalizacji. . .	80
6.7	Kod klasy zarządzającej widokiem AR.	81
6.8	Kod klasy MyRender.	82

Bibliografia

- [1] <https://wearesocial.com/blog/2018/01/global-digital-report-2018>
- [2] <https://www.engineersgarage.com/contribution/what-is-iot-internet-of-things>
- [3] <https://www.ics.ie/news/view/1729>
- [4] [https://www.statista.com/statistics/471264/
iot-number-of-connected-devices-worldwide](https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide)
- [5] [https://www.healthcarestudies.com/article/
What-Can-Augmented-Reality-Bring-to-Medical-Students](https://www.healthcarestudies.com/article/What-Can-Augmented-Reality-Bring-to-Medical-Students)
- [6] Patrick Schueffel, „The Concise FINTECH COMPENDIUM“
- [7] <https://www.colocationamerica.com/blog/history-of-augmented-reality>
- [8] <https://www.augment.com/blog/infographic-lengthy-history-augmented-reality/>
- [9] <http://techsty.art.pl/hipertekst/cyberprzestrzen/krueger.htm>
- [10] [https://community.arm.com/graphics/b/blog/posts/
the-history-of-augmented-reality](https://community.arm.com/graphics/b/blog/posts/the-history-of-augmented-reality)
- [11] <http://sevenmediainc.com/the-history-of-augmented-reality/>
- [12] [https://www.caiservice.com/blog/
as-augmented-reality-grows-more-applications-appear](https://www.caiservice.com/blog/as-augmented-reality-grows-more-applications-appear)
- [13] [https://www.standard.co.uk/tech/2018-will-be-a-big-year-for-augmented-reality
-but-2021-will-be-bigger-a3725101.html](https://www.standard.co.uk/tech/2018-will-be-a-big-year-for-augmented-reality-but-2021-will-be-bigger-a3725101.html)
- [14] <https://thinkmobiles.com/blog/what-is-augmented-reality/>
- [15] <https://www.realitytechnologies.com/augmented-reality/>
- [16] <https://www.igreet.co/the-5-types-of-augmented-reality/>

- [17] <https://uxdesign.cc/augmented-reality-device-types-a7668b15bf7a>
- [18] <https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/>
- [19] <https://meta-eu.myshopify.com/>
- [20] <https://docs.microsoft.com/en-us/windows/mixed-reality/install-the-tools>
- [21] <https://www.skybrary.aero/index.php/Head-Up-Display>
- [22] <https://www.inverse.com/article/31034-augmented-reality-contact-lenses>
- [23] [https://neurogadget.net/2018/04/24/integrating-augmented-reality-in-contact-lenses-is-it-possible/56774](https://neurogadget.net/2018/04/24/integrating-augmented-reality-in-contact-lenses-is-it-possible/)
- [24] <https://www.nanalyze.com/2017/03/smart-contact-lenses/>
- [25] Qusay F. Hassan „Internet of Things A to Z: Technologies and Applications”
- [26] S. Patil, S. Nair, Dr. B. E. Narkhede, Prof. D. V. Pendam „A 20/20 vision of Internet of things
- [27] L. Atzori, A. Iera, G. Morabito „Understanding the Internet of Things: definitione, potentials and societal role of a fast evolving paradigm”
- [28] <https://blogs.perficient.com/2017/11/17/enhancing-energy-and-utilities-with-iot-6-real-world-examples/>
- [29] J. Macaulay, L. Buckalew, G. Chung „Internet of things in Logistics”
- [30] D. E. Zheng, W. A. Carter „Leveraging the Internet of Things for a More Efficient and Effective Military”
- [31] <https://econsultancy.com/internet-of-things-healthcare/>
- [32] <https://www.sciencedaily.com/releases/2018/02/180222090133.htm>
- [33] <https://medium.com/coinmonks/augmented-reality-applications-in-smart-cities-8f511fe5895>
- [34] <https://www.vrfocus.com/2018/05/gatwick-airportsaugmented-reality-passenger-app-wins-awards/>
- [35] <https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/>

- [36] <https://www.slant.co/topics/1321/> best-ides-for-android-development
- [37] <https://developer.android.com/guide/topics/connectivity/bluetooth-le>
- [38] <https://botland.com.pl/pl/373-czujniki-do-arduino>
- [39] <https://store.arduino.cc/genuino-101>
- [40] <https://www.arduino.cc/en/main/software>
- [41] <https://spring.io/blog/2014/03/07/deploying-spring-boot-applications>
- [42] <https://developer.android.com/studio/index.html>
- [43] <https://www.jetbrains.com/idea/>
- [44] <https://www.arduino.cc/>
- [45] <https://astah.net/>
- [46] <https://xm1math.net/texmaker/>