STRONA TYTUŁOWA

SPIS TREŚCI

Wstęp

Postępujący rozwój miniaturyzacji urządzeń elektronicznych spowodował, że na przestrzeni ostatnich kilku lat coraz większym zainteresowaniem na rynku telefonii komórkowej cieszą się smartfony. Zgodnie z danymi opracowanymi przez firmę analityczną IDC, ich sprzedaż wzrosła w 2011 roku o około 80% w porównaniu z rokiem 2010, a już w 2013 roku ilość sprzedanych przez producentów smartfonów egzemplarzy była wyższa niż ilość telefonów komórkowych[1]. Liczba ta wciąż się powiększa, w związku z czym oszacowanie, że do 2020 roku niemal wszyscy użytkownicy telefonów komórkowych będą posiadali smartfony nie wydaje się być zbytnim wyolbrzymieniem. Urządzenia te łączą w sobie funkcje zarówno klasycznego telefonu komórkowego jak i komputera kieszonkowego (*ang. personal digital assistant*). Osiągnięte tym samym kompaktowość i poręczność przy jednocześnie wysokiej jakości oferowanych usług, stały się powodem dla którego smartfony powoli zastępują urządzenia codziennego użytku: aparaty cyfrowe, odtwarzacze mp3, czytniki e-booków i wiele innych. Można również przypuszczać, że z czasem zaczną poważnie konkurować ze sprzętem dedykowanym do badań medycznych. Według raportu Fjord Trends 2016 jednym z dziesięciu najistotniejszych trendów najbliższych lat będą właśnie tzw. osobiste systemy zdrowotne wykorzystujące, oprócz smartfonów, *inteligentną elektronikę do noszenia* (ang. *wearable electronics*, np. Smartwatch, Google Glass etc.) w celu pozyskiwania danych na temat stanu użytkownika[2].

Niniejsza praca zakłada dwa podstawowe cele. Pierwszy dotyczy stworzenia przycisku, który współpracowałby ze smartfonami działającymi pod kontrolą mobilnego systemu operacyjnego Android. Drugi polega na napisaniu aplikacji korzystającej z funkcjonalności takiego przycisku, która dedykowana by była m.in. dla ludzi z problemami neurologicznymi. Jest to praca koncepcyjna z pogranicza dziedzin elektroniki i informatyki, zakładająca możliwość rozszerzenia w ten sposób funkcji smartfonów i wykorzystanie ich w aplikacjach niosących wsparcie w opiece lub leczeniu osób starszych.

Rozdział pierwszy zawiera

1. Wstęp teoretyczny

1.1. Cel i założenia pracy

Podstawą niniejszej pracy jest zrealizowanie dwóch zadań. Pierwsze z nich polega na skonstruowaniu przycisku zewnętrznego do smartfona wyposażonego w system operacyjny Android. Na jego realizację składać się będzie przegląd istniejących możliwości nawiązania połączenia z urządzeniem mobilnym, omówienie kryteriów jakimi należy się kierować podczas wyboru jednej z nich, a także opis elementów niezbędnych do skonstruowania prototypu. Funkcjonalność przycisku będzie wykorzystywana przez aplikację mobilną, której napisanie jest zadaniem drugim. W wyniku naciśnięcia przycisku uruchomiony zostanie wbudowany w telefon moduł GPS w celu odczytania współrzędnych geograficznych użytkownika. Następnie mają one zostać przesłane w postaci wiadomości SMS pod wskazany wcześniej numer telefonu. Aplikacja umożliwi użytkownikowi stworzenie oraz zapisanie w pamięci telefonu szablonów, na które składać się będzie treść wiadomości oraz numer, pod który ma ona zostać wysłana wraz ze współrzędnymi geograficznymi. Jedną z najważniejszych właściwości aplikacji ma być umożliwienie jej pracy w tle, aby skorzystanie z funkcji zewnętrznego przycisku nie wymagało bezpośredniej interakcji użytkownika ze smartfonem. Napisanie aplikacji wymaga zapoznania się z mobilnym systemem operacyjnym Android oraz narzędziami programistycznymi, z których należy skorzystać, przeanalizowania dostępnych na rynku aplikacji związanych z poruszaną w niniejszej pracy tematyką, zaplanowania wyglądu interfejsu dostosowanego do potrzeb użytkownika.

1.2. Urządzenia mobilne typu Smartfon

Smartfony to kieszonkowe urządzenia wielofunkcyjne należące do rodziny telefonów komórkowych. Stały się bardzo powszechne w roku 2012 wraz z trwającym rozwojem standardu telefonii komórkowej 4G. Mimo, że pojęcie „smart phone" zostało użyte po raz pierwszy w 1995 roku, koncepcja połączenia urządzeń zdolnych do zaoferowania zarówno klasycznych funkcji telefonu (wysyłanie wiadomości SMS oraz wykonywanie połączeń) jak i prowadzenia obliczeń została zaprezentowana przez Nikolę Teslę, a opatentowana w 1974 roku przez Teodora Paraskevakosa. Jednym z pierwszych udanych prototypów urządzenia wielozadaniowego, dostępnego do użytku komercyjnego, był Simon firmy IBM opracowany w 1993 roku. Dawał możliwość obsługi prostych „aplikacji mobilnych", otrzymanie oraz wysyłanie poczty e-mail, zawierał m.in. książkę adresową, kalkulator, kalendarz, ekran dotykowy. Smartfony po raz pierwszy zostały rozpowszechnione na masową skalę przez japońską firmę NTT DoCoMo dopiero w 1999 roku. Urządzenia te weszły na rynek w związku z pojawieniem się usługi i-mode, dającej możliwość korzystania z internetu mobilnego. Wykorzystanie przez nie języka C-HTML do opisu stron internetowych zwiększyło szybkość transferu danych, co umożliwiło urządzeniom dostęp do korzystania z większego zakresu usług internetu mobilnego takich jak obsługa poczty e-mail, dokonywania zakupów online, czy po prostu surfowania po sieci. Z czasem ich miejsce zajęły urządzenia zdolne do obsługi technologii sieci 3G[3,4].

Obecnie zatarciu uległa granica rozróżniająca terminy „telefon komórkowy" i „smartfon", przez co często używa się ich zamiennie. Jednak zasadnicza różnica polega na tym, że smartfony dysponują o wiele bardziej zaawansowanym systemem operacyjnym niż standardowe urządzenia służące komunikacji.

1.3. Mobilne Systemy Operacyjne

W rzeczywistości każdy smartfon korzysta z dwóch systemów operacyjnych: poziomu niskiego oraz wysokiego. System operacyjny poziomu niskiego to system (ang. Real-Time Operating System), który posiada własne zasoby i odpowiada m.in. za kodowanie i modulację sygnału radiowego, który umożliwia komunikację[5,6]. Stanowi on uzupełnienie dla systemu poziomu wysokiego, o którym zwykło się myśleć w kontekście mobilnych systemów operacyjnych i który umożliwia użytkownikowi obsługę urządzenia z poziomu interfejsu i aplikacji. Systemy operacyjne wykorzystywane w smartfonach mają cechy zbliżone do tych wykorzystywanych przez komputery stacjonarne czy laptopy, z tą jednak różnicą, że w dużym stopniu wspierają obsługę ekranu dotykowego, komunikację Bluetooth, nawigację GPS, rozpoznawanie mowy i inne funkcje, użyteczne w urządzeniach mobilnych. Obecnie rozwijanych jest ponad 20 platform tego rodzaju, jednak dane statystyczne (udostępnione w sierpniu 2016 przez przedsiębiorstwo Gartner [7])

wskazują, że na rynku światowym 99,7% urządzeń sprzedanych w drugim kwartale bieżącego roku jest wyposażonych w jeden spośród 3 systemów: Android (86,2%), iOS (12,9%) lub Windows (0,6%).

Dane te należy połączyć z informacją na temat dystrybucji smartfonów we wspomnianym okresie – 12.9% światowej sprzedaży należy do Apple Inc., czyli twórców iOS (niegdyś znanego jako iPhone OS). To właśnie ich iPhone pierwszej generacji zaprezentowany w 2007 roku odpowiada za obecny wygląd większości smartfonów: duży ekran dotykowy, brak fizycznej klawiatury i pojedynczy przycisk pod ekranem. Produkty tej firmy cieszą się dużą popularnością wśród Amerykanów. Największym konkurentem dla Apple Inc. wykorzystującym Android OS jest firma Samsung, do której należą 22.3% sprzedaży smartfonów na światowym rynku. Oprócz niej Android OS znajduje się także w urządzeniach kilku innych producentów, m.in. Asus, Huawei, HTC, LG, Motorola czy Sony, co znajduje swoje odbicie w danych zawartych na *Wykresie 1*.

1.4. Android OS

Android jest mobilnym systemem operacyjnym, który od 2005 roku jest rozwijany przez Google Inc. Na rynku pojawił się po raz pierwszy w 2007 roku wraz z ogłoszeniem powstania sojuszu biznesowego o nazwie Open Handset Alliance, który zrzesza przedsiębiorstwa związane z branżą telekomunikacyjną – producentów układów scalonych, telefonów komórkowych i oprogramowania, operatorów sieci komórkowych. Ich głównym celem jest rozwijanie otwartych standardów dla urządzeń mobilnych. Oznacza to, że posiadacz praw autorskich stwarza innym użytkownikom możliwość poznania, modyfikowania i dystrybucji swoich rozwiązań bez ograniczeń. Początkowo OHA obejmowało współpracą 34 firmy, obecnie jest ich ponad 80 (w tym Samsung Electronics, T-Mobile, Google, HTC, Huawei, Texas Instruments etc.) [8]. Android, zgodnie z założeniami OHA, rozwijany jest jako system otwarty (*ang. open source*), co stało się główną przyczyną jego powszechności, dynamiki rozwoju i dostępności. Zarówno jądro Linux, na którym oparty jest system, jak i znaczna część komponentów wykorzystywanych przez Android OS objętych jest licencją GNU GPL. Oznacza to dla użytkowników i deweloperów, że zyskują możliwość rozpowszechniania niezmienionej kopii programu, ale również ulepszonych przez siebie wersji wśród pozostałych członków społeczności. Dodatkowo producenci telefonów należący do OHA objęci są zakazem tworzenia urządzeń niekompatybilnych z Androidem, sam Google natomiast wkłada wiele starań w jego promowanie, czego przykładem jest konkurs na najbardziej nowatorską aplikację zorganizowany w 2008 roku – pula nagród wynosiła ponad 10 milionów dolarów. W efekcie w połowie roku 2013 usługa Google Play (niegdyś Android Market) dawała użytkownikom dostęp do miliona aplikacji. Aplikacja to program komputerowy, w tym wypadku

napisany na urządzenie mobilne, zawierający instrukcje, które są wykonywane przez system. W efekcie telefon realizuje za jej sprawą zadania na potrzeby użytkownika. Do tworzenia aplikacji mobilnych na platformę Android używany jest popularny, obiektowy język programowania o nazwie Java [9]. Wykorzystywane są także pliki XML, głównie w celu przechowywania danych. W listopadzie 2007 OHA opublikowało zestaw narzędzi programistycznych (*ang. Software Development Kit*). Do ich obsługi skorzystano z przeznaczonego do tego celu zintegrowanego środowiska programistycznego (*ang. Integrated Developer Environment*).

1.5. Andorid SDK

SDK zawiera narzędzia przydatne programiście do tworzenia i testowania aplikacji: dokumentację, skompilowane biblioteki, debugger, emulator, przykładowe kody i samouczki. Oficjalna strona dla deweloperów znajduje się pod adresem *https://developer.android.com/. Z*awiera opisy najnowszych aktualizacji, dostępnych bibliotek i funkcji, omawia ich działanie na podstawie przykładowych fragmentów kodu, zawiera samouczki podzielone na bloki tematyczne dla osób zdobywających swoje pierwsze doświadczenia w systemie Android.

Na tym portalu zamieszczona jest również informacja na temat debuggera dostarczanego wraz z Android SDK. Jest to Dalvik Debug Monitor Server, który umożliwia współpracę zarówno z emulatorem jak i fizycznym urządzeniem, na którym testowana jest aplikacja. Poza możliwością śledzenia sposobu w jaki przydzielana jest pamięć dla zmiennych, czy pozyskania informacji na temat czasu wykonywania operacji, z punktu widzenia większości użytkowników jedną z najprzydatniejszych własności jest zdolność podglądu komunikatów w wierszu poleceń „logcat". Zawierają one informacje na temat błędów, które napotkano w trakcie działania aplikacji i są generowane w sposób automatyczny. Ponadto, programista może je wykorzystać używając klasy *Log*, aby wysyłać do siebie wiadomości pełniące rolę znaczników, które dostarczają informacji na temat tego w której części kodu kompilator aktualnie się znajduje oraz jaka jest chronologia wykonywanych przez niego poleceń.

Sprawdzanie efektów działania kodu i sposobu działania aplikacji bez konieczności instalowania jej na fizycznym urządzeniu jest możliwe za pomocą emulatora. Na komputerze symulować można działanie nie tylko smartfonów, ale również tabletów, telewizorów czy tzw. *„wearable"*, które mogą pracować pod nadzorem Android OS. Dużą zaletą korzystania z emulatora jest możliwość sprawdzenia działania aplikacji na różnych typach urządzeń i wersjach systemu, które może on symulować. Należy jednak pamiętać, że możliwości oferowane przez emulator różnią się w praktyce od możliwości fizycznej maszyny (szybkość wykonywania obliczeń i przesyłania danych, obsługa wielu wątków, symulowanie zdarzeń typu połączenia telefoniczne). Aplikacja, która pomyślnie przeszła testy na emulatorze, wykorzystująca bibliotekę SQLite, po zainstalowaniu

prze mnie na telefonie napotykała błąd - kod wymagał poprawienia. Istnieją emulatory zaawansowane, przykład może stanowić Genymotion, jednak nawet one nie umożliwiają testowania połączeń Bluetooth, WiFi, NFC, USB, obserwacji wejść słuchawkowego i na kartę SD [10]. Wskazane jest zatem, a w przypadku tworzenia niektórych aplikacji konieczne, żeby ich działanie przetestować na urządzeniu fizycznym. Jest to możliwe po podłączeniu kablem USB telefonu z odblokowanymi „Opcjami programisty" poprzez zakładkę „Ustawienia".

Instalacja Android SDK wymaga komputera z procesorem z rodziny x86, wyposażonego w system Windows XP (lub nowszy), MacOS X 10.4.8 (lub nowszy) lub Linux. Na platformie uruchomieniowej należy także zainstalować *Java Development Kit* (JDK) - darmowe oprogramowanie firmy Oracle umożliwiające programowanie w języku Java.

Wersje narzędzi służących pisaniu aplikacji zmieniają się wraz z rozwojem systemu i najnowsza opatrzona jest numerem 25. Pobranie aktualizacji, nowości, bądź potrzebnych modułów umożliwia instalator o nazwie *SDK Manager.* Korzystanie z niego oraz ogółu możliwości oferowanych przez SDK ułatwia zintegrowane środowisko programistyczne.

1.6. Android IDE

Zintegrowane środowisko programistyczne to aplikacja, która ułatwia pisanie i edycję kodu źródłowego, a także obsługę narzędzi wykorzystywanych do automatycznego procesu budowy oraz testowania oprogramowania. W trakcie tworzenia aplikacji, o której mowa we wstępie niniejszej pracy, wykorzystano Android Studio, które od stycznia 2016 stało się jedynym IDE oficjalnie wspieranym przez firmę Google [11]. Dotychczas miejsce to zajmowało środowisko Eclipse wyposażone we wtyczkę Android Developer Tools. Do stworzenia Android Studio skorzystano z pomocy firmy JetBrains. Ich zaawansowane IDE o nazwie IntelliJ IDEA (Community Edition), które obsługuje m.in. języki Java, Groovy i XML, stało się bazą, którą przystosowano do potrzeb programistów Android. Automatyczna analiza kodu, mająca na celu poszukiwanie powiązań pomiędzy oznaczeniami użytymi w całej przestrzeni projektowej, skutkuje bardzo efektywnym wsparciem w trakcie pisania aplikacji. Wyświetlane na bieżąco podpowiedzi, możliwość uzupełniania kodu poprzez zastosowanie skrótów klawiszowych, wykrywanie błędów (literówki, brakujące znaki i operatory, argumenty niewłaściwej klasy) i wiele innych powodują, że Android Studio jest skutecznym, intuicyjnym i przyjemnym w użyciu edytorem. Do informacji programisty podawany jest także stopień optymalizacji kodu wraz z informacją, jakie zmiany należy wprowadzić, w celu jego zwiększenia [12, 13].

Do zautomatyzowanej budowy projektu Android Studio wykorzystuje dynamicznie rozwijające się narzędzie – Gradle, który stanowi poważną konkurencję dla dotychczas stosowanych Apache Maven i Apache Ant. Dzięki niemu możliwe jest napisanie aplikacji, która budowana jest w wielu

wariantach, np. dla różnych typów urządzeń (tablet, smartfon) lub w wersji darmowej i płatnej. Pozwala także zadeklarować zewnętrzne biblioteki, z których będziemy korzystać w trakcie tworzenia aplikacji [14].

Po uruchomieniu Android Studio i utworzeniu nowego projektu, należy skonfigurować swoją aplikację. Istotnym jest określenie minimalnej wersji systemu oraz poziomu składni (*API Level*), którą musi dysponować platforma, aby aplikacja działała poprawnie. Wybór ten jest powiązany z wersją SDK, a więc możliwościami aplikacji – im nowsza wersja tym są one większe, ponieważ każda kolejna jest kompatybilna z poprzednią. Jednak istnieje jeszcze drugie kryterium wyboru, dotyczy ono ilości urządzeń, które są wyposażone w system o wybranej wersji lub wyższy. Oznacza to, że decydując się na wersję systemu Android 7.1 Nougat (*API Level* 25) nasza aplikacja będzie miała dostęp do najnowszych narzędzi zaoferowanych przez producenta, jednak na rynku mniej niż 1% urządzeń będzie w stanie z niej skorzystać (jak wynika z danych z dnia 1.11.2016r. udostępnionych przez środowisko Android Studio). W wyborze pomocny jest wykres obrazujący jaki procent urządzeń aktualnie posiada daną wersję systemu oraz jakie zmiany ona wprowadza. W przestrzeni roboczej znajduje się pole edycji kodu oraz okno debuggera „logcat". Zakładka Android, domyślnie umieszczona w lewej części ekranu, to najważniejsza część przestrzeni roboczej, gdzie dostępne są pliki interpretowane przez Gradle oraz dotyczące bezpośrednio wyglądu i funkcji tworzonej aplikacji. Pliki XML przechowują informacje dotyczące wyglądu interfejsu użytkownika (*layout files)* aplikacji dla poszczególnych ekranów użytkownika. Mogą być one uzupełniane ręcznie, co wymaga od programisty doświadczenia, albo automatycznie, z wykorzystywaniem edytora graficznego dostarczonego w ramach Android Studio. Dodatkowo pliki XML wykorzystuje się do przechowywania w uporządkowany sposób danych, takich jak stałe matematyczne, czy treści komunikatów (dla łatwiejszego tłumaczenia aplikacji na inne wersje językowe – wystarczy zmodyfikować jeden plik XML). Specyficznym plikiem o rozszerzeniu *.xml* jest AndroidManifest. Zawarte są w nim informacje dla systemu niezbędne do uruchomienia aplikacji: identyfikator, komponenty, wymagane uprawnienia (dostęp do sieci GPS, nawiązanie komunikacji Bluetooth itp.) [15]. Pliki zawarte w folderze Java związane są z funkcjami aplikacji. Podstawowymi komponentami każdej aplikacji są *aktywności.* Dziedziczą one po klasie *AppCompatActivity* (lub po prostu *Activity*) metody, których przeciążenie pozwala określić, który wzór interfejsu zostanie wyświetlony, a także jak udostępnione w ten sposób elementy będą zachowywać się w wyniku interakcji z użytkownikiem. Metody te podlegają tzw. *cyklowi życia* (ang. *lifecycle*), czyli schematowi przedstawionemu na *Rysunku 1*, który określa chronologię, zgodnie z którą są one wywoływane od pierwszego wyświetlenia *aktywności* aż do zamknięcia aplikacji. Jego znajomość jest wymagana do napisania programu, który w sposób poprawny będzie komunikował się z użytkownikiem, nie spowolni pracy telefonu, czy zapobiegnie przypadkowej

utracie wprowadzonych danych. Podobnie scharakteryzować należy obiekty z rodziny klasy *Fragment.* Stanowią część *aktywności*, dla której udostępniają własny interfejs oraz metody związane z *cyklem życia*. Znajdują zatem zastosowanie, gdy pewne elementy interfejsu i ich funkcje nadają się do wielokrotnego użycia (wówczas jedna *aktywność* może być wypełnione przez kilka *fragmentów*) lub gdy interfejs, który należy wykorzystać może przybrać różne warianty w zależności od określonych przez programistę warunków (*aktywność* jest całkowicie wypełniana przez pojedynczy *fragment*) [16].

Szczególnie interesujące są pochodne klasy *Service (Usługa)* ze względu na umożliwienie aplikacji za ich sprawą pracy w tle. Ich *cykl życia*, w przeciwieństwie do *fragmentów* i *aktywności*, nie jest związany z aktualnie widocznym ekranem użytkownika. *Usługa* może zostać uruchomiona z poziomu *aktywności*. Zgodnie z treścią poradników, dobrą praktyką jest przydzielanie oddzielnego wątku na potrzeby jej zadań, zwłaszcza w sytuacjach, gdy dotyczy ona nawiązania i utrzymania połączenia, prowadzenia długotrwałych obliczeń, czy odtwarzania plików MP3. W przeciwnym wypadku zablokowany może zostać wątek obsługujący interfejs użytkownika i wzrasta ryzyko pojawienia się komunikatu o błędzie „Aplikacja nie odpowiada”.

Pisanie aplikacji zostało rozpoczęte od stworzenia listy stawianych jej wymagań oraz rozważenia różnych sposobów ich realizacji, co może uchronić przed czasochłonnym przebudowywaniem kodu w trakcie jego pisania. Zagadnieniami koniecznymi do dogłębnego przeanalizowania w niniejszej pracy było m.in. zaplanowanie sposobu w jaki telefon będzie komunikował się z przyciskiem zewnętrznym oraz jak przechowywać dane użytkownika.

1.7. Przechowywanie danych użytkownika

Aplikacje mobilne często wymagają od użytkownika podania danych, dzięki którym dokonana zostanie weryfikacja tożsamości lub dostosowana będzie funkcjonalność do preferencji użytkownika. W im większym stopniu dane pozyskiwane przez aplikację są poufne, tym większy wysiłek powinien włożyć jej twórca w bezpieczeństwo ich przechowywania. Dane mogą być zapisywane i później pozyskiwane ze źródła zewnętrznego, takiego jak baza danych z którą aplikacja łączy się poprzez sieć. Drugim rozwiązaniem jest operowanie na plikach znajdujących się w pamięci telefonu – w tym celu programiści mogą skorzystać np. z bazy *SQLite* lub z klasy *SharedPreferences*.

SQLite jest bazą danych dostarczoną wraz z systemem Android. Jej wykorzystanie opiera się na znajomości komend używanych przy obsłudze standardowych baz SQL i znajduje zastosowanie w przechowywaniu uporządkowanych struktur danych w dużych ilościach. *SQLite* domyślnie wykorzystuje przestrzeń pamięci zarezerwowaną dla aplikacji, wobec czego do danych nie mają

dostępu pozostałe aplikacje. Ponadto wykorzystanie bazy danych gwarantuje dużą elastyczność i szybki dostęp do poszukiwanych rekordów [17, 18]. Jednak do przechowywania niewielkich ilości danych o nieskomplikowanej strukturze lepszym rozwiązaniem jest skorzystanie z klasy *SharedPreferences*. Z założenia służy ona do zapisu opcji konfigurujących aplikację zgodnie z preferencjami użytkownika. W praktyce możliwe jest zapisywanie par klucz-wartość w pliku o rozszerzeniu *.xml*, który jest tworzony w obszarze pamięci przydzielonym dla aplikacji, do którego dostęp ma tylko pojedyncza *aktywność* lub, podobnie jak w przypadku bazy *SQLite*, cała aplikacja. Odwołanie do pliku następuje również poprzez podanie klucza, który zabezpiecza dane przed ich nadpisaniem lub wykradzeniem [19, 20].

Do pracy z klasą *SharedPreferences*, w celu zapisania danych innych niż typy proste (tzn. *integer, long, float, boolean* oraz *String*), można skorzystać z biblioteki *Gson* udostępnionej i rozwijanej przez programistów firmy Google od 2008 roku [21]. Służy ona do konwertowania obiektu stworzonego w języku Java do formatu *JSON (JavaScript Object Notation)*. W efekcie otrzymywany jest obiekt klasy *String*, z którego przy wykorzystaniu metod biblioteki *Gson* można z powrotem uzyskać postać sprzed konwersji. Możliwe jest zatem przechowywanie obiektów niemal dowolnych klas w formie tekstowej korzystając z klasy *SharedPreferences*. Dodatkowo, konwersja może usprawnić składowanie danych. Na przykład zamiast zapisywania n par klucz-wartość możliwe jest zapisanie listy, zawierającej n elementów, w formacie *JSON*. Wówczas odczytywanie danych z pliku *.xml* wymaga jednorazowej (zamiast n-krotnej) identyfikacji elementu poprzez klucz.

Dodatkowym sposobem na uchronienie informacji przed osobami trzecimi jest ich zaszyfrowanie przed zapisaniem, dzięki czemu bez klucza deszyfrującego będą one bezużyteczne [22]. Tworzona aplikacja nie zakłada konieczności pozyskiwania danych wymagających tak wysokiego stopnia ochrony.

1.8. Koncepcje połączeń z przyciskiem

Użyteczność tworzonej aplikacji w bardzo dużym stopniu zależeć będzie od sposobu, w jaki do smartfona zostanie dołączony dodatkowy przycisk. Analiza dostępnych rozwiązań pozwala podzielić metody połączeń na dwie podstawowe grupy: przewodowe i bezprzewodowe.

Połączenia przewodowe wykorzystują wejścia fizyczne - gniazda micro USB lub audio o średnicy 3,5mm (tzw. „mini jack”). W pierwszym przypadku bardzo przydatny jest adapter USB OTG (On-The-Go), który umożliwia podłączenie do smartfona urządzeń peryferyjnych (klawiatury, myszki, dyski zewnętrzne). Wadą tego rozwiązania jest konieczność napisania sterownika dla przycisku, ponadto nie wszystkie smartfony (zwłaszcza starsze) obsługują poprawnie podłączone w ten sposób urządzenia zewnętrzne, co wiąże się z uciążliwym procesem konfiguracji. Ciekawą

alternatywę stanowi wykorzystanie wejścia audio w sposób analogiczny do tego, jaki wykorzystuje przycisk zestawu słuchawkowego. Zasada działania takich przycisków polega na tym, że równolegle do wejścia mikrofonowego przyłączana jest rezystancja o wartościach (zgodnie ze specyfikacją systemu Android [23]) 0, 135, 270, 470 [Ohm] tworząc w ten sposób dzielnik napięcia z wbudowanym rezystorem 2,2 kOhm. Napięcie na tym dzielniku jest rejestrowane przez przetwornik analogowo-cyfrowy znajdujący się w smartfonie i w zależności od jego wartości wywoływana jest odpowiednia metoda. Domyślnie funkcjonalność taka jest wykorzystywana np. w aplikacjach typu odtwarzacz multimedialny (funkcje *„play”*, *„pause”*, *„next”*, *„vol+”*, *„vol-”*). Nieco inne zastosowanie zaprezentowali twórcy akcesoriów Selfie stick oraz Pressy.[24] Funkcjonalności urządzeń przez nich oferowanych, działające w oparciu o opisaną powyżej zasadę, są wykorzystywane do robienia zdjęć, uruchamiania wbudowanej latarki, czy dyktafonu. Połączenia wykorzystujące gniazdo audio do niestandardowych zastosowań spotykają się z krytyką, ze względu na ograniczanie części funkcji telefonu (wyłączenie wbudowanego głośnika czy blokada mikrofonu przy wciśnięciu przycisku) oraz wywoływanie częstych konfliktów z odtwarzaczami multimedialnymi [25]. Ogólną wadą połączeń przewodowych jest ograniczenie funkcjonalności telefonu wynikające z zajęcia gniazd oraz ewentualne zmniejszenie komfortu jego użytkowania związane z obecnością przewodów oraz przejściówek i rozgałęziaczy. Większością poruszonych problemów nie jest obarczona komunikacja bezprzewodowa. Jej realizacja na małe odległości jest możliwa w technologii IrDA, WiFi oraz Bluetooth.

Zgodnie ze standardem IrDA nośnikiem danych jest fala o długości z zakresu podczerwieni (850-900nm) cechująca się niedużą zdolnością do przenikania przez obiekty. Uzyskanie połączenia pomiędzy dwoma urządzeniami wymaga, aby znajdowały się one we wzajemnym polu widzenia. Emitowana wiązka stożkowa ma standardowo zasięg około metra i obejmuje kąt 30^{o}. Wymaga zatem od użytkownika właściwego ustawienia nadajnika, jednak charakteryzuje ją stosunkowo duża odporność na zakłócenia czy przechwycenie transmisji przez osoby trzecie. Tak powstała sieć łączy jedynie 2 urządzenia, nie wymagając dużej interakcji ze strony użytkownika. Warstwa protokołu dostępu do łącza (IrLAP) funkcjonuje domyślnie w trybie poszukiwania partnera do dialogu. Określa optymalne warunki przesyłu danych pomiędzy urządzeniami. Protokół IrCOMM pozwala na emulowanie portów równoległych oraz szeregowych [26-28].

Stworzenie bardziej obszernej sieci lokalnej jest możliwe przy wykorzystaniu technologii Bluetooth.

1.9. Technologia Bluetooth

Połączenia Bluetooth opierają się na architekturze określonej z języka angielskiego jako *Master/Slave* przy wykorzystaniu fal radiowych o częstotliwości około 2.4 GHz, należącej do

pasma ISM (Industrial, Scientific, Medical) [29]. *Master* oznacza urządzenie *nadrzędne*, które kontroluje zegar w celu synchronizacji komunikacji i jest inicjatorem połączenia z urządzeniem *podrzędnym* (*Slave*). Podstawą wspomnianej architektury jest pikosieć (*ang. piconet)*, na którą składa się pojedyncze urządzenie pełniące rolę *nadrzędną* oraz maksymalnie 7 aktywnych jednostek *podrzędnych*. Pikosieci mogą ze sobą oddziaływać i tworzyć większe struktury (ang. *scatternet)*, przez ustalenie wspólnej jednostki *podrzędnej* [27].

Zdolność do stworzenia sieci lokalnej jest jednym z powodów, dla którego technologię Bluetooth chętnie stosuje się do wymiany danych pomiędzy urządzeniami z grupy *inteligentnej elektroniki do noszenia*. Dodatkowym atutem jest dość swobodny kierunek propagacji nośnej w postaci fali radiowej oraz jej większa (w porównaniu do IrDA) zdolność do przenikania i odbijania się od przeszkód. Problem stanowić może interferencja z niepożądanymi falami z pasma zbliżonego do 2.4GHz, takimi jak szum generowany przez USB 3.0, w wyniku czego połączenie Bluetooth może zostać utracone [30]. Procedura transmisji danych polega na odpytywaniu kolejnych urządzeń *podrzędnych* przez jednostkę *nadrzędną* w celu pozyskania informacji na temat ich stanu, wobec czego rozpoczęcie przesyłu jest możliwe dopiero po uzyskaniu pozwolenia na nadawanie. Transmisja wykorzystuje metodę rozproszonego widma uzyskanego metodą zmiany częstotliwości nośnej (*ang. Frequency Hopping Spread Spectrum)*. Pasmo ISM jest podzielone na 79 kanałów o szerokości 1MHz każdy. Jednostka *nadrzędna* określa w sposób niedeterministyczny (pseudolosową) sekwencję zmian kanałów, w których będzie odbywać się wymiana danych pomiędzy nią a urządzeniami *podrzędnymi* znajdującymi się w *pikosieci*. Praca na zmiennej częstotliwości znacznie utrudnia podsłuchanie transmisji urządzeniom spoza *pikosieci* oraz stwarza możliwość poprawnego funkcjonowania wśród innych standardów, które współdzielą pasmo 2.4GHz (np. Wi-Fi) [31].

Na potrzeby opisu procesu nawiązywania łączności Bluetooth wprowadzono pojęcia *parowania (ang. pairing)* oraz *powiązania (ang. bonding)* urządzeń. Terminy te są rozróżniane ze względu na funkcjonalności, które wprowadzają. *Parowanie* to jednorazowe zdarzenie, którego głównym celem jest ustalenie klucza służącego do kodowania połączenia pomiędzy urządzeniami. Podczas niego wymieniane są także informacje dotyczące możliwości wprowadzania oraz odczytu danych przez urządzenia [32, 33]. Na ich podstawie podczas *parowania* zachodzi proces uwierzytelniania połączenia, którego przebieg zależy m.in. od wersji standardu Bluetooth, czy typu urządzeń. Często polega na wciśnięciu pojedynczego przycisku (rozwiązanie zazwyczaj stosowane przez urządzenia nie posiadające rozbudowanego interfejsu użytkownika) lub na podaniu kodu PIN (spotykane zazwyczaj wśród urządzeń zgodnych ze standardem v2.0 lub starszym) [33, 34]. *Parowanie* zakończone sukcesem pozwala *powiązać* urządzenia. Ponieważ klucze kodujące (szyfrujące?)

połączenie są zapisane w pamięciach urządzeń, możliwa jest między nimi komunikacja i kolejna próba nawiązania jej nie będzie wymagała ponownej wymiany wrażliwych danych ani interakcji ze strony użytkownika. *Powiązanie* sparowanych urządzeń może być wykonywane w sposób automatyczny w momencie, gdy znajdą się one we wzajemnym zasięgu, co znacząco poprawia wygodę oraz funkcjonalność wykorzystania technologii Bluetooth [32, 35]. Proces łączenia można podzielić kolejno na etapy:

- *Poszukiwania* (*ang. Inquiring)*, kiedy urządzenia nie są połączone i udostępniają sobie informację na temat nazwy i adresu MAC (*ang. Media Access Control)*;
- *Wywoływania* (*ang. Paging)*, kiedy nawiązywane jest połączenie przez urządzenie lub stwarza ono taką możliwość dla urządzeń znających jego adres MAC. Na tym poziomie może zostać wywołany proces *parowania*, następuje synchronizacja przeskoków między częstotliwościami dla urządzeń w roli *„Master”* i *„Slave”*;
- *Połączenia* (*ang. Connected)*, kiedy możliwa jest transmisja danych pomiędzy urządzeniami. W tym stanie jednostka może przejść w tryb energooszczędny, uśpienia bądź braku aktywności, przy zachowaniu pełnej synchronizacji z pozostałymi członkami *pikosieci* [36-38];

Ze względu na moc, którą dysponują i proporcjonalny do tej wartości zasięg komunikacji, urządzenia Bluetooth dzielą się na trzy klasy, których typowe parametry zostały zestawione w Tabeli 1.

–----------Tabela 1. Podział urządzeń Bluetooth na klasy.

Szybkość transmisji danych (przepustowość łącza) jest związana z kolejnymi wersjami standardu Bluetooth. Są one kompatybilne wstecznie i chociaż w lipcu 2016 roku organizacja Bluetooth SIG (Special Interest Group), zajmująca się rozwojem tej technologii, ogłosiła datę opublikowania standardu Bluetooth 5 na koniec grudnia 2016 roku, to najczęściej spotykane są urządzenia wykonane według standardu v2.x (z 2007 roku). Standard ten zwiększył transfer danych do około 2.1 Mb/s (co jest wielkością trzykrotnie większą w porównaniu do poprzedniej wersji), ułatwił proces *parowania* urządzeń (dla wersji 2.1, dodano np. rozszerzenie NFC, *ang. Near Field Communication*) zwiększając jednocześnie jego bezpieczeństwo [39]. Zwiększenie szybkości przesyłu danych było możliwe dzięki zastosowaniu techniki kluczowania fazy (ang. *Phase Shift Keying)* kodującej unikalne układy bitów na dyskretnych wartościach zmian fazy fali nośnej. Wpłynęło to pozytywnie na zmniejszenie poboru mocy, głównie ze względu na spadek czasu potrzebnego na przesłania danych, oferując jednocześnie zasięg połączenia do około 10m [40]. Rozwiązanie to jest wystarczające dla potrzeb większości urządzeń. Zwiększenie przepustowości łącza nawet do 40Mb/s umożliwił standard Bluetooth v3.x, natomiast zainteresowanie *„wearable*

electronics” znajduje swoje odbicie w standardzie v4.x, dla którego kosztem relatywnie niskiej szybkości transmisji danych (1 Mb/s) został zwiększony zasięg połączenia do 100m i znacznie ograniczono pobór energii, dzięki czemu urządzenia mogą działać nawet przez kilka miesięcy wykorzystując baterię pastylkową [39]. Ponieważ komunikacja Bluetooth powstała z myślą zastąpienia przewodów połączeniowych zgodnych ze standardem RS-232, wymiana danych jest oparta o emulację portu szeregowego w sposób opisany przez protokół RFCOMM. Dzięki temu wpasowała się ona w już istniejące rozwiązania i nie wymaga ich modyfikowania. Twórcy technologii Bluetooth zaadaptowali wiele rozwiązań, takich jak protokół wymiany danych, bezpośrednio ze standardu IrDA. Możliwe jest opracowanie rozwiązań łączących cechy obu standardów, czego przykładem może być koncepcja wykorzystania komunikacji IrDA przy wsparciu poszukiwania i wskazywania urządzenia, z którym użytkownik chce nawiązać połączenie Bluetooth [38].

Jednym z najbardziej powszechnych rozwiązań, umożliwiających komunikację w technologii Bluetooth pomiędzy urządzeniami jest wyposażenie każdego z nich w moduł Bluetooth połączony z mikrokontrolerem. Każdy z tych elementów posiada parę odprowadzeń RX (*ang.* receiver) oraz TX (*ang. transmitter*). Pin RX jest odpowiedzialny za otrzymywanie danych, natomiast pin TX za ich wysyłanie, wobec czego łączy się je w następujący sposób: pin RX należący do modułu Bluetooth z pinem TX mikrokontrolera i pozostałe dwa analogicznie. Dzięki nim zachodzi wymiana bitów z częstotliwością określoną przez wielkość nazywaną *„Baud Rate”* wyrażoną w liczbie bitów na sekundę (*bps*). Inaczej mówiąc, im większy *„Baud Rate”* tym krótszy czas trwania sygnału reprezentującego pojedynczy bit. Ważne jest, żeby zarówno mikrokontroler jak i moduł Bluetooth pracowały z tą samą częstotliwością transmisji (zazwyczaj jest to wartość 9600bps). Bity są połączone w tzw. *ramki* i oprócz kodowania właściwych danych sygnalizują początek i koniec transmisji/słowa (tzw. bity synchronizujące) oraz weryfikują poprawność przesyłania danych (tzw. bity parzystości, nie zawsze występują). W wymianie informacji pomiędzy urządzeniami pośredniczy zazwyczaj UART (*ang. Universal Asynchronous Receiver and Transmitter),* czyli układ dokonujący konwersji z interfejsu szeregowego na równoległy i odwrotnie. Transmisję równoległą wykorzystują układy cyfrowe, głównie w celu zwiększenia szybkości obliczeń - jeden takt zegara odpowiada przesłaniu całego słowa, a nie pojedynczego bitu (jak ma to miejsce w transmisji szeregowej) [41, 42].

Możliwe jest kupno płytki bazowej z wbudowaną anteną RF, które pozwalają nawiązać łączność bezprzewodową w standardzie Bluetooth v4.0 Low Energy bez konieczności dołączania dodatkowego modułu Bluetooth.

2. Implementacja

2.1. Przycisk bezpieczeństwa

2.1.1. Opis zaproponowanego rozwiązania

Po zapoznaniu się z metodami umożliwiającymi połączenie przycisku ze smartfonem, została podjęta decyzja o wykorzystaniu do tego celu komunikacji Bluetooth. Z punktu widzenia użytkownika korzystanie z połączenia nie jest skomplikowane. Czynności związane z procedurą *parowania* nie są tak intuicyjne jak podłączenie przewodu do gniazda telefonu (np. USB), zwłaszcza dla osób starszych, które mają być głównymi beneficjentami przycisku. Należy jednak pamiętać, że jest to czynność jednorazowa, która może być wykonana przez osoby trzecie (inne niż bezpośredni użytkownik przycisku) podczas pierwszego uruchomienia urządzenia, natomiast kolejne połączenia mogą być nawiązywane w sposób automatyczny. W zamian, dzięki połączeniu Bluetooth, zyskać można dużo większy komfort użytkowania bez konieczności stosowania dodatkowych rozgałęziaczy i przewodów połączeniowych ograniczających zasięg i swobodę poruszania się użytkownika. Dużym atutem tego rodzaju komunikacji jest możliwość stworzenia *pikosieci,* w której rolę jednostki *nadrzędnej* przyjmie smartfon, wobec czego potencjalny rozwój projektu o dalsze moduły współpracujące z aplikacją nie powinien być nadmiernie skomplikowany. Komunikacja IrDA, pomimo że prostsza w użytkowaniu, w kontekście niniejszej projektu wprowadzałaby niepotrzebne skomplikowanie, głównie ze względu na kierunkowy charakter wiązki nośnej.

Prototyp mający służyć weryfikacji podstawowych założeń związanych z funkcjonowaniem przycisku jest zgodny z koncepcją zakładającą wykorzystanie mikrokontrolera i modułu Bluetooth. W celu jego stworzenia posłużyłem się płytką bazową Arduino Micro oraz modułem HC-05, które zostały połączone na płytce prototypowej. Po upewnieniu się, że zastosowane komponenty umożliwiają właściwe funkcjonowanie przycisku, prototyp zyskał finalną, użytkową postać.

2.1.2. Kryteria, którymi kierowano się przy wyborze elementów do budowy układu

Rozwiązanie wykorzystujące płytkę prototypową nie jest najbardziej efektywnym ze względu na rozmiar i użyteczność, jednak jego celem jest stworzenie możliwości obserwacji sygnałów oraz łatwego wprowadzania modyfikacji do układu. Nie wymaga ono łączenia elementów w sposób trwały, wobec czego jest to jedna z prostszych, powszechnie stosowanych metod budowania układów w celu weryfikacji ich poprawności.

Głównym powodem wyboru płytki bazowej z rodziny Arduino jest bardzo łatwy sposób programowania mikrokontrolera, na którym się ona opiera. Wbudowany *program rozruchowy*

(ang. bootloader) umożliwia wgrywanie kodu napisanego na komputerze w przeznaczonym do tego celu środowisku programistycznym udostępnionym przez producenta, zastępując tym samym zewnętrzny programator. Zajmuje on zazwyczaj około 2-4kB pamięci Flash mikrokontrolera, pozostawiając do dyspozycji użytkownika pozostałą część. Przesyłanie kodu (którego składnia przypomina język C) odbywa się przy użyciu przewodu łączącego gniazdo USB komputera z wejściem microUSB płytki. Arduino Micro charakteryzuje się dodatkowo niewielkim rozmiarem (wymiary poprzeczne 48x18[mm]) oraz wagą (13g) w porównaniu do większości modeli. Zawiera pojedynczy interfejs komunikacyjny UART (odprowadzeniom RX/TX odpowiadają piny 0 oraz 1), udostępnia 32kB pamięci Flash, a jego rdzeń stanowi mikrokontroler ATmega32U4 odpowiedzialny zarówno za obsługę połączenia nawiązywanego poprzez przewód USB jak i za wykonanie kodu, co odróżnia Arduino Micro od pozostałych płytek (z wyłączeniem Arduino Leonardo). Różnica ta ma wpływ na sposób korzystania z portu szeregowego oraz współpracę z komputerem [43, 44], co zostanie wyjaśnione przy okazji omawiania kodu mikrokontrolera, a także cenę modelu – zastosowanie pojedynczego procesora generuje mniejszy koszt produkcji, a więc i cenę, za jaką można nabyć Arduino [43]. Transmisja szeregowa odbywa się w standardzie TTL 5V, w związku z czym do optymalnego zasilania płytki wymagane jest źródło prądu stałego o napięciu od 7V do 12V, doprowadzone do regulatora napięcia poprzez piny V_{IN} oraz GND, lub 5V dostarczanych do gniazda microUSB. Zasilanie płytki napięciami innymi niż zalecane może prowadzić do jej niewłaściwego funkcjonowania i ewentualnego uszkodzenia [44]. Dopuszczalna wartość prądu płynącego przez piny I/O Arduino Micro wynosi w optymalnych warunkach 20mA, jednak w razie potrzeby może wzrosnąć do 40mA nie powodując trwałych uszkodzeń układu. Opisane wielkości napięć i prądów są typowe dla większości płytek bazowych Arduino lub różnią się nieznacznie, wobec czego nie stanowią istotnego kryterium porównawczego [45].

Dopuszczalne wartości prądu, jakie można otrzymać z wyjść Arduino Micro są wystarczające do pracy z modułem Bluetooth HC-05 wykonanym w technologii CMOS, umieszczonym na płytce wzorowanej na JY-MCU v1.02 zawierającej wyprowadzenia pinów ułatwiających montaż. Zarówno do zasilania jak i transmisji danych wykorzystywane są napięcia na poziomie 3.3V (zasilanie maksymalnie do 6V). Aby ułatwić połączenie z Arduino, większość płytek ma wbudowany regulator napięcia. Zgodnie ze specyfikacją, maksymalny prąd może zostać pobrany przez moduł HC-05 w trakcie *parowania* i wynosi on nie więcej niż 40mA (średnio 25mA), zaś w trybie normalnej pracy ogranicza się do około 8mA bez względu na to, czy przetwarzane są dane [46]. HC-05 jest wykonany zgodnie ze standardem Bluetooth v2.0+EDR (*ang. Enhanced Data Rate*), z czym wiąże się jego dość niskie zużycie energii oraz zasięg komunikacji do 10m. Zawiera interfejs komunikacyjny UART dopuszczający możliwość zmiany częstości transmisji bitów danych (parametr *„Baud Rate”)* poprzez użycie komend AT. Komendy umożliwiają również zmianę m.in.

nazwy urządzenia, kodu pin wymaganego w procesie *parowania* (domyślnie jest to 1234 lub 0000) oraz trybu pracy (jako jednostka *podrzędna* lub *nadrzędna* architektury *Master/Slave*) [47]. Uzyskana tym sposobem elastyczność oraz łatwość użytkowania, a także niska cena (około 3USD za sztukę) to powody, dla których HC-05 jest modułem popularnym, chętnie wykorzystywanym we współpracy z płytkami Arduino oraz aplikacjami na platformę Android, zawierającym bogatą dokumentację. Jego wymiary na płytce JY-MCU są zbliżone do rozmiaru Arduino Micro.

2.1.3. Prezentacja i opis działania prototypu

Połączenie komponentów układu na płytce prototypowej odbyło się zgodnie ze schematem przedstawionym na Obrazie 4. Pełny wykaz użytych elementów znajduje się w Załączniku 4 do pracy. Dzielnik napięciowy, na który składają się rezystory R2-R4, każdy o wartości 1kOhm (±5%), pełni rolę konwertera napięć logicznych. Obniża on napięcie otrzymywane na wyjściu TX mikrokontrolera, które wynosi 5V, zanim zostanie ono „odczytane” przez wejście RX modułu Bluetooth, dla którego wielkością zalecaną są 3.3V [47]. Ma to na celu zmniejszenie ryzyka uszkodzenia modułu HC-05. Możliwe jest poprawne funkcjonowanie urządzeń wykonanych w standardzie CMOS połączonych bezpośrednio z urządzeniami pracującymi zgodnie ze standardem TTL, bez konwersji poziomu sygnałów logicznych. Własność ta została wykorzystana w przypadku połączenia pinów TX modułu HC-05 oraz RX Arduino Micro.

----------schemat układu i zdjęcie prototypu (obrazy 4 i 5)

Zasada działania przycisku polega na tym, że wraz z jego naciśnięciem na pinie D2 Arduino Micro rejestrowany jest stan wysoki (*High*) napięcia. Kod mikrokontrolera interpretuje to zdarzenie jako sygnał do wysłania przez port szeregowy (przez pin TX interfejsu UART) prostego znaku z tablicy ASCII. Jest on przetwarzany przez moduł Bluetooth, który przekaże go w postaci sygnału radiowego. W stanie normalnym pin D2 odczytuje stan niski (*Low*) doprowadzony przez rezystor R5 (2.2kOhm) z wyjścia GND płytki Arduino.

Obecność czerwonej diody LED o średnicy 3mm jest związana z funkcją informacyjną - sygnalizuje ona wciśnięcie przycisku. Aby mieć pewność, że jej świecenie jest jednoznaczne z zarejestrowaniem przez mikrokontroler naciśnięcia przycisku, dioda jest zapalana za sprawą instrukcji wykonywanych dla tego samego warunku, co wysłanie informacji przez port szeregowy. Prąd płynący przez diodę jest ograniczany przez rezystor R1.

Schemat układu został wykonany w programie EAGLE, który wspomaga projektowanie obwodów elektrycznych. Wbudowany edytor płytek obwodów drukowanych (*ang. Printed Circuit Board, PCB*) umożliwił rozmieszczenie elementów oraz zaplanowanie przebiegu ścieżek w celu stworzenia nakładki (tzw. *„shield”*) na Arduino Micro. Rozwiązanie to pozwoliło ulokować w sposób warstwowy komponenty zajmujące najwięcej przestrzeni (Arduino Micro, moduł HC-05, bateria).

Przeniesienie układu z płytki prototypowej na obwód drukowany pozwoliło efektywnie zmniejszyć wielkość prototypu, którego rozmiar jest porównywalny z brelokiem lub małym pilotem. Przewlekany montaż elementów jest odporny na uszkodzenia mechaniczne oraz wstrząsy. Użytkownik posiada możliwość łatwej wymiany modułu Bluetooth, gdyby ten uległ awarii.

2.1.4. Kod Arduino

Kod do sterowania pracą mikrokontrolera został napisany w Arduino IDE i został zaprezentowany w Załączniku nr 1 niniejszej pracy. Po wgraniu na mikrokontroler przechowywany jest w obszarze pamięci Flash, która nie jest wymazywana wraz z odłączeniem zasilania. Jego pierwszy blok (przed częścią „S*etup*") definiuje/inicjuje wartości stałe i zmienne. Wyprowadzeniu D2 mikrokontrolera przypisana zostaje nazwa *buttonPin* w celu zwiększenia czytelności kodu. Określone zostają również wejściowe wartości zmiennych opisujących stan w jakim znajduje się obecnie (bądź znajdował w poprzednim cyklu) przycisk, natomiast ostatnie dwie zmienne (typu *unsigned long)* zostaną wykorzystane przy eliminacji zjawiska drgania styków przycisku (*ang. bouncing*). Następna struktura tworząca kod opatrzona jest nazwą „*Setup*" i tak jak blok definicji, jest wykonywana jednorazowo po uruchamianiu mikrokontrolera. Wykorzystana została do skonfigurowania stałej *buttonPin* jako pinu wejściowego Arduino oraz zainicjowania transmisji szeregowej z dwoma parametrami. Pierwszy narzuca szybkość transmisji (9600bps), natomiast drugi, stosowany opcjonalnie, służy do określenia postaci ramki danych - pozostawienie tego pola pustego oznacza przypisanie wartości domyślnej (8 bitów oraz bit stopu, bez kontroli parzystości) [50]. Ostatnią strukturą, która zawiera właściwą część programu, jest pętla nieskończona „*Loop"*. Instrukcje w niej zawarte, po uruchomieniu mikrokontrolera są wykonywane cyklicznie. Zmienna *reading* przechowuje stan w jakim aktualnie znajduje się pin D2 (*wysoki* oznacza, że przycisk jest wciśnięty). Jeśli jest on różny od stanu (domyślnie *niskiego*) z poprzedniego cyklu, do zmiennej *lastDebounceTime* wpisywany jest wynik funkcji *millis()*, podającej w milisekundach czas, który minął od uruchomienia programu. Dalsze instrukcje zostaną wykonane dopiero w momencie, gdy różnica pomiędzy czasem zarejestrowania zmiany, a aktualnym będzie większa niż 50ms, co wskazuje na ustalenie się sygnału i eliminuje wpływ zjawiska drgania styków. W zależności od poziomu sygnału ustalonego (*wysoki* lub *niski*) wykonywane są operacje odpowiednio zaświecania diody i wysyłania zmiennej *k* lub gaszenia diody. Inkrementacja wartości znaku *k*, wysyłanego przez port szeregowy, ułatwia weryfikowanie poprawności działania przycisku, w związku z czym problemu nie stanowi sytuacja, w której dochodzi do przekroczenia zakresu wartości (0-255), które można mu przypisać. Zważywszy na fakt, że jest to zmienna typu *char,* zajmuje ona najmniejszą możliwą ilość pamięci (jest zakodowana na 8 bitach). Zmienna, dla której bardzo istotny jest typ danych poprzez który jest reprezentowana,

to *lastDebounceDelay*. Jak podają twórcy Arduino IDE, aby móc przechować wynik działania funkcji *millis()* odpowiadający czasowi równemu ponad 47 dni, zmienna powinna być typu *long* bez znaku (32 bity).[51] Wielkość ta gwarantuje, że urządzenie będzie funkcjonować poprawnie, ponieważ przewidywany czas ciągłej pracy przycisku bez wyłączenia zasilania (a więc bez resetowania licznika funkcji *millis()*) zawiera się poniżej tej granicy. Ostatnim zagadnieniem, na które należy zwrócić uwagę, jest przesyłanie danych przez port szeregowy o nazwie *Serial1*. Większość płytek Arduino wykorzystuje w tym celu klasę *Serial* (wyjątkiem jest Arduino Mega posiadające 4 interfejsy UART). Dla procesora zastosowanego w Arduino Micro klasa *Serial* zarezerwowana jest wyłącznie dla komunikacji USB z komputerem, który jest portem wirtualnym, tworzonym na nowo wraz z każdorazowym uruchomieniem się *programu rozruchowego*. Oznacza to, że resetowanie płytki (poprzez przycisk na niej umieszczony lub wgrywanie nowego programu) wiąże się z zerwaniem i ponownym nawiązaniem połączenia. Ma to wpływ na zmienny numer portu przydzielany płytce przez komputer oraz na wgrywanie kodu i monitorowanie przepływu danych [43]. Transmisja przez piny RX/TX, w przypadku Arduino Micro, obsługiwana jest za sprawą klasy *Serial1*. Rozróżnienie nazw portów należy mieć na uwadze w trakcie pisania i analizy kodu.

2.1.5 Wstępny test działania prototypu

Do weryfikacji poprawności działania prototypu nawiązana została komunikacja pomiędzy komputerem, mającym możliwość transmisji danych przez Bluetooth, oraz modułem HC-05 prototypu w celu odczytania wiadomości przesłanej z mikrokontrolera. W tym celu wykorzystany został darmowy program o nazwie PuTTy. Ma on możliwość pracy w trybie monitora portu szeregowego COM na komputerze.

Po *sparowaniu* urządzeń procedura testu polegała na skonfigurowaniu ustawień programu, m.in. wprowadzeniu nazwy obserwowanego portu (np. COM18) oraz wartości parametru „*baud rate*” (9600). W utworzonym w ten sposób oknie terminalu, za każdym razem, gdy został naciśnięty przycisk umieszczony na płytce prototypowej, na ekranie wyświetlał się ciąg znaków wysyłanych poprzez port szeregowy Arduino Micro. Wynik testu spełnia oczekiwania, a jego rezultat został przedstawiony na Obrazie 6.

Urządzenie zostało zasilone z portu USB komputera o napięciu nominalnym 5V przy użyciu przewodu połączeniowego (USB-microUSB), wykorzystywanego do wgrywania programu na płytkę Arduino. W celu pomiaru pobieranego prądu oraz zużycia energii, pomiędzy nie został włączony szeregowo tester USB firmy KEWEISEI, model KWS-V20. Działa w warunkach pomiaru napięć z zakresu 4-20[V] oraz przepływu prądu do 3A, wyniki podając z dokładnością do 0.01.

----------Schemat układu usb(komputer)-tester-microUSB(arduino)----------

Przeprowadzone zostały 3 testy, każdy trwający 1h (pomiar czasu rejestrowany przez tester). Dotyczyły one kolejno pomiarów w sytuacjach, gdy zasilana jest:

- płytka Arduino Micro;
- płytka Arduino Micro z podłączonym modułem HC-05 (brak połączenia z telefonem);
- płytka Arduino Micro z podłączonym modułem HC-05 (po połączeniu ze smartfonem);

Wyniki pomiarów zostały zamieszczone w Tabeli 2. Stanowią podstawę do oszacowania wymaganych parametrów baterii, z której zasilony będzie układ, m.in. jej pojemności oraz wydajności prądowej. Naciśnięcie przycisku i zaświecenie diody powoduje wzrost chwilowego natężenia prądu o około 20mA. Czas świecenia jest krótki w odniesieniu do założonego całkowitego czasu pracy urządzenia, wobec czego zmiany te nie wywierają zauważalnego wpływu na sumaryczne zużycie prądu, choć należy je wziąć pod uwagę w kontekście wydajności prądowej źródła.

Test zasięgu komunikacji, pomiędzy urządzeniem a smartfonem, wykonano na otwartej przestrzeni oraz w pomieszczeniu zamkniętym. …

2.1.6. Zasilanie układu

Na podstawie przeprowadzonych testów oraz danych z dokumentacji technicznej Arduino Micro zaprojektowany został obwód zasilający prototyp, widoczny na Obrazie 5. Wykaz użytych do jego budowy elementów zawiera Załącznik 4 pracy. Jako źródło napięcia wybrana została niewielkich rozmiarów bateria CR123A (firmy Energizer), której napięcie nominalne wynosi 3V. Cechuje ją maksymalny prąd stały wynoszący 1.5A oraz pojemność rzędu 1500mAh (dla spadku napięcia do 2V) co pozwala urządzeniu na pracę przez ponad 78h (zakładając pobór prądu wynoszący 19mA). Aby podnieść poziom napięcia do wartości umożliwiającej pracę płytki Arduino (5V) wykorzystana została przetwornica „step-up” POLOLU-2564 pracująca w zakresie napięć od 0.5V do 5.5V z wydajnością 70-90%. W praktyce, zmierzone na jej wyjściu napięcie wynosiło około 5.8V. Jak podaje dokumentacja, płytki Arduino nie mają zazwyczaj wbudowanych regulatorów napięcia na wejściu gniazda microUSB (jedynie pomiędzy pinami V_{IN} i GND). Podanie na nie napięcia o wielkości odbiegającej od 5V może spowodować uszkodzenie układu scalonego. Aby temu zapobiec, wyjście przetwornicy zostało połączone ze stabilizatorem napięcia LM2940, który charakteryzuje się małym współczynnikiem „dropout voltage” wynoszącym 0.5V(0.1V) przy prądzie wyjściowym równym 1A(0.1A). Niemożliwe było zastosowania popularnego stabilizatora LM7805, dla którego wielkość ta wynosi około 2V. Otrzymane na wyjściu stabilizatora napięcie

wynosi 5.01V, co jest wartością bardzo zbliżoną do zmierzonego napięcia na gnieździe USB komputera, i jest podawane na piny zasilające żeńskiego gniazda USB. Umożliwia ono połączenie układu zasilającego i płytki Arduino za pomocą popularnego, używanego do tej pory przewodu USB-microUSB. Zaletą opisanego rozwiązania jest zastosowania źródła napięcia o niższej wartości (piny V_{IN} i GND wymagają podania napięcia z zakresu 7-12V), a także ewentualna możliwość skorzystania w sytuacji awaryjnej z tzw. banku energii (*ang. powerbank).*

----------*Obraz 5 : schemat i zdjęcie zasilania/płytki pcb*

2.2. Aplikacja dla Android OS

2.2.1. Wybór wersji platformy oraz urządzenie testujące działanie aplikacji

Aplikacja została napisana na urządzenia obsługujące platformę Android w wersji 4.4 *KitKat (*API Level 19) lub nowszą. Udostępnione przez Android Studio dane z dnia 13 grudnia 2016 wskazują na to, że aplikacje na tym poziomie zaawansowania mogą być instalowane na około 74% urządzeń dostępnych na rynku. Wprowadzenia wersji Android 4.4 w 2013 roku wiązało się z wykorzystaniem nowego środowiska uruchomieniowego (*ang. runtime environment*), wydajniejszego od używanego dotychczas pod względem zużycia energii. Wprowadzone zostało również pojęcie „aplikacji domyślnej" obsługującej usługę SMS, która jako jedyna ma możliwość wysyłania wiadomości oraz rejestrowania szczegółowych powiadomień na temat jej stanu. Uprawnienia pozostałych aplikacji są ograniczone, co jest związane ze zwiększeniem bezpieczeństwa użytkowania smartfonów. Wsparte zostały również klasy obsługujące alarmy oraz komunikację NFC. Pełny opis zmian, wprowadzonych wraz z aktualizacją, dostępny jest na stronie dewelopera [52]. Wcześniejsze wersje systemu udostępniają funkcje potrzebne do zrealizowania założeń aplikacji (współpraca z urządzeniami Bluetooth, w tym Low Energy – Android 4.3 *Jelly Bean*).

Do testowania aplikacji, ze względu na ograniczone możliwości emulatora dostarczonego wraz z Android Studio, użyty został smartfon marki Samsung, wersja Galaxy Grand Prime, model SM-G530FZ. Wyposażony jest m.in. w czterordzeniowy procesor Cortex-A53 o częstości taktowania 1.2GHz, 1GB pamięci RAM, adapter Bluetooth v4.0 oraz ekran o przekątnej równej 5 cali. Działa pod kontrolą mobilnego systemu operacyjnego Android 5.0.2 *Lollipop*.

2.2.2. Interfejs użytkownika

Pisanie aplikacji na platformę Android zostało rozpoczęte od zaplanowania wyglądu interfejsu użytkownika oraz przejść pomiędzy poszczególnymi ekranami. Głównym założeniem, przez wzgląd na docelowych użytkowników, było stworzenie aplikacji nieskomplikowanej w obsłudze, co

swoje odbicie znajduje w mało rozgałęzionym schemacie przejść, widocznym na Obrazie 5. Zawartość plików XML, na podstawie których generowane są ekrany użytkownika, znajduje się w Załączniku 2. Użytkownik jest przeprowadzany w sposób kontrolowany przez etap niezbędnej konfiguracji aplikacji. Ilość modyfikacji, które może on wprowadzić, została ograniczona do niezbędnego minimum, co wpływa na łatwość użytkowania aplikacji i zmniejszenie ryzyka wystąpienia błędu.

Po uruchomieniu aplikacji wyświetlony zostaje ekran wyboru funkcji, z której użytkownik może skorzystać. Na tym etapie przewidziana została dodatkowa część, związana z przypominaniem o zażyciu leków (przycisk „*MEDICAMENTS*”). Próba jej uruchomienia skutkuje wyświetleniem się komunikatu informującego o pracach trwających nad jej ukończeniem.

Wybór opcji „*EMERGENCY*” przekierowuje do widoku, w którym możliwe jest skonfigurowanie połączenia Bluetooth. Użytkownik jest proszony o zgodę na uruchomienie transmisji, jeśli akcja ta nie została podjęta wcześniej. Przyciski „Paired” i „Discovery”, znajdujące się w górnej części ekranu, dają możliwość wyświetlenia listy adresów MAC i nazw urządzeń, z którymi przeprowadzony został już proces *parowania* (dane uzyskane z pamięci smartfona) lub wszystkich aktualnie znajdujących się w zasięgu transmisji. Wybór urządzenia następuje po jednokrotnym kliknięciu na jego nazwę. Informacje identyfikujące urządzenie, z którym po naciśnięciu przycisku „*CONFIRM*”, ma zostać nawiązane połączenie są wyświetlane w centralnej części ekranu. Uruchomienie aplikacji bez zgody na transmisję Bluetooth jest możliwe po wybraniu opcji „RUN WITHOUT BT”.

Umieszczenia wirtualnego guzika „ACTION” w górnej części kolejnego ekranu powoduje, że funkcjonalność aplikacji nie zostaje całkowicie ograniczona przez tę opcję. Guzik zastępuje w pewnym stopniu przycisk zewnętrzny, ponieważ jego naciśnięcie skutkuje wysłaniem wiadomości SMS zawierającej współrzędne geograficzne użytkownika. Pozostała część ekranu użytkownika służy do prezentacji danych, które zostaną wykorzystywane w razie użycia przycisku zewnętrznego. Pojedynczy zestaw danych tworzy obiekt klasy *EmergencyObject* i nazywany będzie *wzorem*. Na wzór składają się:

- nazwa, widoczna w pierwszym okienku tekstowym;
- numer telefonu, umieszczony w kolejnym polu tekstowym, pod który zostanie wysłana wiadomość SMS;
- treść wiadomości SMS, do której dołączone zostaną współrzędne geograficzne odczytane z modułu GPS. Wyświetlana jest w najbardziej obszernym, trzecim oknie;

Wraz z uruchomieniem aplikacji, okna są wypełniane danymi pochodzącymi ze *wzoru* znajdującego się na szczycie listy, która staje się widoczna po naciśnięciu przycisku „*CHANGE PATTERN*”.

Wspomniana lista prezentuje wszystkie *wzory* zapisane w pamięci smartfona. Ograniczając ich widoczną postać do nazwy i umieszczonego pod nią numeru telefonu, możliwe jest wyświetlenie do 6 elementów listy, zachowując zdolność do zidentyfikowania poszukiwanego *wzoru*. Wybór odbywa się poprzez krótkie, jednokrotne kliknięcie w wybraną pozycję (podobnie jak w przypadku wyboru urządzenia Bluetooth). Aby zapobiec przypadkowej zmianie *wzoru*, decyzję należy potwierdzić klikając „Confirm" w pojawiającym się oknie weryfikującym. Dłuższe przytrzymanie palca na elemencie listy wyświetli opcje umożliwiające jego edycję bądź usunięcie. Aplikacja nie pozwala użytkownikowi na całkowite wyczyszczenie listy, dzięki czemu nie powinno dojść do sytuacji, w której w pamięci urządzenia nie znajduje się ani jeden *wzór*, na podstawie którego wiadomość może zostać wysłana. Podobnie, podczas pierwszego uruchomienia aplikacji na urządzeniu lista powinna zawierać pozycję o nazwie „*Custom Pattern"* z wprowadzonym numerem 112. Jej usunięcie jest możliwe dopiero po dodaniu nowego *wzoru* poprzez ekran uruchamiany po naciśnięciu (znajdującego się pod obszarem zajmowanym przez listę) przycisku „*ADD NEW PATTERN*". Dla dodawania i edytowania *wzorów* wykorzystywany jest identyczny ekran użytkownika. Zawiera 3 pola edycji tekstu, gdzie należy wprowadzić kolejno nazwę *wzoru*, numer telefonu i treść wiadomości SMS. Ułożenie pól edycji jest analogiczne do rozmieszczenia okien tekstowych dla ekranu prezentującego dane wykorzystywane przez aplikację. Różnica pomiędzy *dodawaniem wzoru* a jego *edytowaniem* polega na tym, że w pierwszym przypadku pola edycji tekstu są puste (zawierają jedynie wskazówki), natomiast w drugim są one wypełnione automatycznie wartościami dotychczas zapisanymi. Naciśnięcie przycisku „*SAVE*" i potwierdzenie wyboru w przeznaczonym do tego oknie skutkuje odpowiednio zapisaniem lub nadpisaniem stworzonego *wzoru* w pamięci telefonu. Jest on umieszczany na szczycie listy, dzięki czemu przy ponownym uruchomieniu aplikacji zostanie wykorzystany jako najbardziej aktualny.

Starano się, aby polecenia kierowane do użytkownika były krótkie i rzeczowe, a elementy interfejsu, za pomocą których dokonuje on wyboru, były wyraźnie widoczne. Do wyświetlania napisów wykorzystano (poza wyjątkami) duże czcionki, aby zwiększyć czytelność tekstu. Ponieważ niektóre z ekranów mogą wydawać się podobne, zastosowano różnokolorowe tła mające ułatwić zorientowanie się w jakiej części aplikacji użytkownik się znajduje. Są one jednolite, aby łatwiej było wyróżnić użyteczne informacje. Dodatkowo, wiele działań ze strony użytkownika spotyka się z odpowiedzią aplikacji w postaci komunikatów wyświetlanych przez aplikację. Ich pojawienie się jest zazwyczaj związane z próbą podjęcia przez użytkownika niedozwolonej akcji i wyjaśniają powód, dla którego nie może zostać ona zrealizowana.

2.2.3. Nawiązywanie i obsługa komunikacji Bluetooth

Nawiązanie i obsługa komunikacji Bluetooth wiąże się z wykonaniem zespołu czynności, które

można podzielić na 4 grupy:

- zaprogramowanie ekranu użytkownika, pozwalającego na konfigurowanie połączenia;
- pozyskanie adresu urządzenia, z którym ma być nawiązane połączenie;
- wystawienie żądania o połączenie i *powiązanie* urządzeń;
- obsługa połączenia i zdarzeń;

Aplikacja wykorzystująca funkcje Bluetooth wymaga nadania jej uprawnień *BLUETOOTH* oraz *BLUETOOTH_ADMIN*, jeśli ma być możliwe nawiązywanie, konfigurowanie i wykorzystywanie połączenia.

Funkcje ekranu użytkownika, którego wygląd i zadania zostały omówione w poprzednim rozdziale, zaprogramowane zostały we *fragmencie* o nazwie *BluetoothListFragment*. Został on umieszczony w przestrzeni *aktywności BluetoothListActivity.* Funkcja *checkBtState()* sprawdza, czy smartfon wyposażony jest w domyślny adapter Bluetooth. Jeśli nie, użytkownik jest informowany o tym, że komunikacja Bluetooth z wykorzystaniem jego urządzeniu nie jest możliwa. W przeciwnej sytuacji dochodzi do dalszego sprawdzenia w jakim stanie adapter się znajduje. W przypadku, gdy jest wyłączony zostaje podjęta decyzja o uruchomieniu go, która wymaga potwierdzenia ze strony użytkownika za pośrednictwem okna dialogowego. Wynik operacji, w postaci liczby *całkowitej* warunkuje to, jaki komunikat zostanie wyświetlony na ekranie.

Interakcja z aplikacją za pośrednictwem ekranu użytkownika jest możliwa poprzez skorzystanie z przycisków, których użycie skutkuje wywołaniem przypisanych im funkcji. W celu skojarzenia zmiennej, reprezentującej przycisk lub dowolny element interfejsu w pliku *BluetoothListFragment.java,* z jej definicją w pliku *fragment_bluetooth_list.xml*, wykorzystano bibliotekę o nazwie *ButterKnife* autorstwa Jake'a Whartona. Została zainstalowana poprzez wprowadzenie w pliku *build.gradle(Module:app)* instrukcji/komendy:

—----------

Ostatnie cyfry określają wersję instalowanej biblioteki. *ButterKnife* pozwala zastąpić klasyczną metodę *findViewById()* w sposób pokazany poniżej*:*

—----------

Wpływa to korzystnie na czytelność, płynność pisania oraz elastyczność kodu. Przypisywanie nazw do numerów identyfikujących elementy interfejsu odbywa się w obszarze deklaracji zmiennych, a nie w przestrzeni metody *onCreateView()* (jak ma to zazwyczaj miejsce). Najnowsza wersja opatrzona jest numerem 8.4.0 [53] i różni się nieznacznie od zastosowanej w omawianej aplikacji (np. zastąpieniem komend *„Inject”* przez *„Bind”*), jednak różnice te nie wywierają istotnego wpływu na optymalizację kodu.

Do *powiązania* (i ewentualnego *parowania*) z urządzeniem zewnętrznym, smartfon musi dysponować jego adresem MAC. Jego pozyskiwanie rozpoczyna się w chwili naciśnięcia jednego z przycisków, które wyświetlają listę urządzeń sparowanych lub wykrytych w otoczeniu smartfona. Wywołana zostaje klasa *BtDeviceList* z parametrami właściwymi dla danego przycisku. Jeśli argumentem jest stała *GET_PAIRED*, lista zawiera sparowane już urządzenia i tworzona jest w oparciu o rekordy znajdujące się w pamięci smartfona. Jest to operacja mało kosztowna z punktu widzenia wykorzystanych zasobów, jednak nie dająca gwarancji, że wybrane urządzenie będących w zasięgu komunikacji ze smartfonem. Alternatywą jest bieżące wyszukiwanie urządzeń znajdujących się w zasięgu transmisji i zestawianie informacji na ich temat w formie listy, bez względu na to czy zostały one wcześniej sparowane (etap *Poszukiwania*, którego krótki opis zawiera rozdział 1.9.). W tym celu stworzony został obiekt klasy *BroadcastReceiver*, który nazywany będzie *odbiornikiem*. Jego zadaniem jest zarejestrowanie obecności i pozyskanie informacji na temat wykrytych urządzeń Bluetooth. Nadpisanie metody *onDestroy()* przez dodanie instrukcji *unregisterReceiver* zapewnia, że *odbiornik* stworzony w sposób dynamiczny (tzn. wewnątrz *aktywności BtDeviceList*) będzie prowadził nasłuchiwanie do momentu, w którym rzeczona *aktywność* zostanie zamknięta. Służy to ograniczeniu wykorzystania pamięci oraz baterii smartfona. Wyświetlane listy urządzeń Bluetooth (sparowanych lub wykrytych) pozostają puste w sytuacji, gdy komunikacja Bluetooth jest wyłączona. Wyszukiwanie urządzeń Bluetooth jest operacją pochłaniającą dużo mocy i spowalniającą działanie smartfona, co jest powodem, dla którego bezpośrednio po wybraniu urządzenia z listy, proces prowadzenia dalszych poszukiwań jest zatrzymywany. Informacje na temat wybranego urządzenia zapisane są w postaci pojedynczego ciągu znaków, wobec czego konieczne było skorzystanie z funkcji *substring()* celem podzielenia go na podciągi zawierające pożądane informacje – nazwę oraz adres MAC. Następnie dane są zwracane do *aktywności BluetoothListActivity.* Załączona zostaje również stała o nazwie *RESULT_OK* informująca, że proces ich pozyskania zakończył się sukcesem. W efekcie, aktualizacji ulegają informacje na temat urządzenia, z którym zostanie podjęta próba nawiązania połączenia Bluetooth.

Zatwierdzenie części konfigurującej połączenie przyciskiem *CONFIRM* uruchamia nową *aktywność*, jednak nie jest ona bezpośrednio odpowiedzialna za stworzenie komunikacji Bluetooth. Jak zostało wspomniane we wstępie, ponieważ nawiązanie oraz obsługa transmisji Bluetooth nie powinny być zależne od *cyklu życia aktywności*, ich implementacja znajduje się w klasie typu *usługa*, o nazwie *BluetoothService*. Metoda *onCreate() usługi* wywoływana jest tylko raz dla pojedynczego uruchomienia aplikacji, dlatego ewentualne zerowanie zmiennych odbywa się wewnątrz metody *onStartCommand()*, której zawartość wykonywana jest za każdym razem, gdy *usługa* jest resetowana. Dla omawianej aplikacji sytuacja ta ma miejsce, po zmianie *wzoru*, na

podstawie którego uzupełniane są dane wiadomości SMS lub, gdy po uruchomieniu aplikacji w trybie pracy bez urządzenia zewnętrznego, użytkownik zechce po pewnym czasie połączyć je ze smartfonem. Przesłany adres MAC podlega wstępnej kontroli (dokładniejsza weryfikacja oraz sprawdzenie czy urządzenie znajduje się w zasięgu transmisji przeprowadzone zostaną w dalszej części programu). W oparciu o niego tworzony jest obiekt *deviceToConnectWith,* reprezentujący urządzenie Bluetooth, który posłuży do utworzenia *kanału komunikacyjnego* (*ang. Socket)* oraz zainicjowania transmisji pomiędzy fizycznymi urządzeniami. W tym celu posłużono się klasą lokalną *ConnectThread*, do zaimplementowania której wykorzystany został kod udostępniony przez producenta na oficjalnej stronie systemu Android [54]. Należy wprowadzić właściwy dla modułu HC-05 identyfikator (UUID), który dla portu szeregowego (SSP) wynosi 00001101-0000-1000-8000-00805F9B34FB [55] i jest weryfikowany w trakcie procesu łączenia smartfona z urządzeniem zewnętrznym. Jeśli wprowadzony adres MAC był poprawny, a urządzenie znajduje się w zasięgu, proces łączenia zostaje zakończony sukcesem. W przeciwnym wypadku, maksymalnie po 12 sekundach [54], proces kończy się niepowodzeniem, *kanał* jest zamykany, a użytkownik zostaje cofnięty do ekranu konfiguracji połączenia Bluetooth oraz poinformowany o zaistniałym błędzie.
Poprawnie stworzony *kanał* umożliwia zdefiniowanie wejściowego oraz wyjściowego strumienia danych w oparciu o klasę *ConnectedThread,* której szkielet znajduje się w dalszej części poradnika [54]. Metoda *read()* oczekuje, aż w strumieniu wejściowym znajdą się dane przekazane z modułu Bluetooth HC-05. Są one wczytywane w postaci tablicy bitów. W odpowiedzi na pojawienie się wiadomości, gdy włączony jest moduł GPS w smartfonie, uruchamiana jest funkcja aktualizująca współrzędne telefonu i wysyłająca wiadomość SMS. Rola obiektu *mHandler* w przekazywaniu wiadomości została wyjaśniona w rozdziale traktującym o wielowątkowości.

2.2.4. Wielowątkowość

Wykorzystanie wątku głównego w celu nawiązywania połączenia Bluetooth może być powodem zablokowania połączeń przychodzących [54]. Podobnie wczytywanie danych lub wykonywanie czasochłonnych obliczeń może uniemożliwić interakcję pomiędzy użytkownikiem a interfejsem i jeśli stan ten utrzyma się przez czas dłuższy niż 5 sekund, niezawodnie skutkować będzie wyświetleniem przez system komunikatu o prawdopodobnym zawieszeniu się aplikacji, dając użytkownikowi możliwość jej zamknięcia. Zgodnie z poradnikami [55, 56], w sytuacjach długotrwałego przetwarzania danych, aby uniknąć wyżej opisanych problemów, należy skorzystać z wielowątkowości. Klasy *ConnectThread* oraz *ConnectedThread* stanowią jawne rozszerzenie klasy *Thread*, co oznacza, że instrukcje nadpisujące metodę *run()* są wykonywane w osobno przydzielonych wątkach, asynchronicznie względem pozostałej części programu. W efekcie, zadania takie jak nawiązywanie połączenia i odczytywanie danych z portu szeregowego,

wykonywane są w tym samym czasie co obsługa interfejsu użytkownika bez ryzyka zawieszenia się aplikacji. Do przekazywania danych między wątkami służą obiekty klasy *Handler,* tzw. *uchwyty.* Odczytana z portu szeregowego wiadomość (znak *char* wysłany przez Arduino Micro) jest przekazywana za pomocą *uchwytu mHandler* do wątku głównego, obsługującego interfejs użytkownika (inaczej mówiąc: do wątku, w którym *uchwyt* został stworzony). Dopiero tam wywołane zostają funkcje stanowiące odpowiedź na naciśnięcie przycisku, bez zakłócania pracy wątku odpowiedzialnego za wczytywanie danych ze strumienia wejściowego.

Innym (niż jawne rozszerzenie klasy *Thread*) sposobem na przydzielenie zadaniom osobnego wątku, jest implementacja tzw. interfejsu *Runnable* i nadpisanie jego (jedynej) metody *run()* w opisany wcześniej sposób. Wpisane w ten sposób instrukcje mogą być traktowane jak zawartość nowo utworzonego wątku. W aplikacji rozwiązanie to zostało wykorzystane w celu zapewnienia płynnego działania aplikacji, niezakłócanego przez nawiązywanie komunikacji Bluetooth czy łączenie z serwisem Google Play.

2.2.5. Odczytywanie współrzędnych GPS. Usługi Google Play.

Wprowadzeniu systemu Android w wersji 7.0 *Nougat (API Level 24)* towarzyszyło zaprzestanie wspierania i rozwoju stosowanej dotychczas biblioteki *android.location*. Zawiera ona metody i rozwiązania służące do tworzenia aplikacji wykorzystujących funkcję GPS, które zostały określone mianem „przestarzałych” [58]. Zalecanym rozwiązaniem jest skorzystanie z usług Google Play, działających w tle systemu Android. Są one zainstalowane na każdym urządzeniu posiadającym system Android w wersji nowszej niż 2.2, a na ich usunięcie użytkownik decyduje się na własne ryzyko. Wykorzystanie właściwych bibliotek (tzw. „*client library”*) umożliwia tworzonej aplikacji komunikowanie się z usługami Google Play, które w efekcie wykonują pewne czynności w jej imieniu. Główna korzyść płynąca z takiego rozwiązania polega na tym, że usługi Google Play są na bieżąco ulepszane i aktualizowane, zatem aplikacje, które je implementują, korzystają z najnowszych rozwiązań niezależnie od zainstalowanej wersji systemu Android. Aktualizacje pobierane są za sprawą aplikacji Google Play Store [59]. Aby skorzystać z usług, należy w projekcie zainstalować wymagane dodatki (w czym użyteczny będzie instalator *SDK Manager)* oraz załadować wybrane biblioteki z listy znajdującej się na stronie dewelopera Google Play [60] (w sposób analogiczny do dodawania biblioteki *ButterKnife*). W celu aktualizowania współrzędnych położenia użytkownika, wykorzystywana jest biblioteka *„play-services-location”*. Należy również nadać aplikacji uprawnienie ACCESS_FINE_LOCATION, pozwalające na możliwie najdokładniejsze określenia lokalizacji wykorzystując wbudowany moduł GPS. Alternatywą jest określanie współrzędnych na podstawie sygnału operatora sieci komórkowej oraz okolicznych punktów dostępu WiFi, co jest rozwiązaniem korzystniejszym ze względu na zużycie

energii smartfona, jednak zwracającym mniej dokładne wartości.

Instrukcje, mające za zadanie określić położenia użytkownika, stanowią zawartość interfejsu *Runnable*, znajdującego się wewnątrz klasy *BluetoothService*. Rozpoczynają się od sprawdzenia, czy urządzenie, na którym znajduje się aplikacja, posiada dostęp do usług Google Play. Jeśli tak, aplikacja łączy się nimi i jest rejestrowana jako klient usługi *LocationServices*. Wywołanie funkcji *startLocationUpdates()*, która powoduje aktualizację położenia użytkownika zgodnie z warunkami określonymi wewnątrz funkcji *createLocationRequest()*, następuje w wyniku zarejestrowania naciśnięcia przycisku awaryjnego. Precyzowanie wielkości takich jak minimalny interwał czasowy czy dystans, dla którego aktualizowana ma być lokalizacja, nie jest kluczowe w kontekście tworzonej aplikacji, ponieważ pozyskanie współrzędnych ma być efektem naciśnięcia przycisku. Istotny jest argument metody *setPriority()*, który określa jak dokładne, a więc z jakich źródeł i jakim kosztem zużycia energii, współrzędne będą uzyskane. Wybierając *PRIORITY_HIGH_ACCURACY* wskazuje się się, że preferowanym źródłem informacji na temat lokalizacji będzie sygnał GPS, jednak w zależności od możliwości, wykorzystane mogą zostać także sygnały operatora sieci komórkowej oraz WiFi. Aby ograniczyć pochłanianie energii przez aplikację, pozyskiwanie współrzędnych odbywa się przez krótki okres czasu, tuż po naciśnięciu przycisku. W pozostałych przypadkach zaktualizowanie aktualizacja położenia następuje jedynie w wyniku działania innej aplikacji (np. Google Maps). W celu zrealizowania tego założenia wykorzystano odrębny interfejs *Runnable*, w którym umieszczone zostały instrukcje przerywające pobieranie współrzędnych oraz przypisujące uzyskane wartości do zmiennych reprezentujących szerokość oraz długość geograficzną (*ang. longitude/latitude)*. Wykonanie poleceń w wątku głównym następuje z opóźnieniem *TIME_FOR_LOCATION_UPDATE* (5000ms), wykorzystanym jako argument metody *postDelay() uchwytu mHandler.* Wielkość opóźnienia została wybrana eksperymentalnie jako kompromis pomiędzy ustaleniem się prawidłowych wartości zmiennych, a możliwie najkrótszym czasem ich aktualizowania. Uśpienie wątku, jako alternatywny sposób uzyskania tego samego efektu, wiązałoby się z niepożądanym zablokowaniem interfejsu aplikacji na 5 sekund. Jeśli pomimo uruchomionego modułu GPS, zaktualizowanie współrzędnych nie powiodło się (np. w z powodu aktywnego „Trybu Samolotowego” lub błędu) zmienne „*longitude”* i „*latitude”* przyjmują wartości równe 0 (efekt zerowania zmiennych, który odpowiada położeniu 0°00'00.0"N 0°00'00.0"E, czyli okolicom Zatoki Gwinejskiej). Wynik jednoznacznie wskazuje na błędne zlokalizowanie użytkownika aplikacji. Fragmenty opisanego kodu bazują na poradniku dostępnym na stronie [61, 62].

2.2.6. Wysyłanie wiadomości SMS

Przyjęte zostało założenie, że po upłynięciu czasu TIME_FOR_LOCATION_UPDATE wszystkie

dane (tzn. współrzędne GPS oraz informacje pochodzące z wybranego *wzoru*), potrzebne do wysłania wiadomości SMS o żądanej treści, są zaktualizowane i dostępne. Po zapisaniu współrzędnych położenia we właściwych zmiennych, wykorzystana zostaje funkcja *sendSMS()*. Tworzy ona domyślny obiekt, który służy do zarządzania komunikacją SMS. Argumenty metody *sendTextMessage()* określają treść wiadomości oraz numer, pod jaki zostanie wysyłana. Znajdują się one odpowiednio w zmiennych *emergencyMessage* oraz *emergencyNumber* (obie typu *String*). Ich źródłem jest klasa *EmergencyViewFragment*, która jest odświeżana za każdym razem, gdy zmieniany jest wykorzystywany *wzór* wiadomości. W efekcie wywołana zostaje funkcja *sendMessageAndNumberToService(),* co prowadzi do „zrestartowania" *usługi BluetoothService* (ponowne uruchomienie *onStartCommand()*) i zaktualizowania zawartości zmiennych *emergencyMessage* oraz *emergencyNumber.* Współrzędne geograficzne zostają wysłane w osobnej wiadomości SMS, której treść stanowi konkatenacja zmiennych *„latitude", „longitude"* oraz ciągu znaków typu *String.* W zależności od tego, czy w trakcie działania aplikacji funkcja GPS była włączona, różna jest treść wysyłanej wiadomości. Powodem, dla którego wysyłane są dwie osobne wiadomości SMS zamiast pojedynczej, jest losowe występowanie błędu, skutkującego zatrzymaniem działania aplikacji. Towarzyszący mu komunikat wskazuje na kwestie bezpieczeństwa aplikacji, jednak bezpośrednia przyczyna jego powstawania nie została poznana (pełna treść komunikatu znajduje się w komentarzach do kodu w Załączniku 2).

Tworzonej aplikacji nie są nadane uprawnienia umożliwiające niezależne wysyłanie wiadomości SMS, w związku z czym w procesie tym pośredniczy aplikacja domyślna [63]. Wysłana wiadomość umieszczana jest w skrzynce nadawczej, której podgląd powinna umożliwiać aplikacja domyślna. Jeśli operacja nie mogła zostać ukończona (np. ze względu na tymczasowy brak zasięgu), zdarzenie to zostanie obsłużone w sposób typowy dla ustawionej aplikacji pośredniczącej (najczęściej jest to ponowne nadanie wiadomości, gdy pojawi się taka możliwość).

2.2.7. Funkcjonowanie aplikacji bez przycisku zewnętrznego

Aplikacja stwarza możliwość aktualizowania informacji na temat położenia użytkownika i wysłania wiadomości SMS poprzez interakcję na poziomie interfejsu użytkownika (a nie fizycznego przycisku). Efekt ten uzyskano dzięki przypisaniu funkcji, uruchamianych przez przycisk zewnętrzny, do przycisku *„ACTION"*, który stanowi część ekranu użytkownika *aktywności EmergencyDetailActivity. Usługa BluetoothService* pełni podwójną rolę - oprócz obsługiwania połączenia Bluetooth, udostępnia swoje publiczne metody i funkcje. Korzystać z nich mogą *aktywności* implementujące metodę *bindService()* [64]. Funkcjonalność, będąca skutkiem powiązania, jest wykorzystywana jedynie, gdy ekran *aktywności* jest widoczny. Z tego powodu metody *bindService()* i *unbindService()* zostały umieszczone odpowiednio wewnątrz funkcji

onStart() i *onStop().* Stworzenie obiektu *bluetoothService*, poprzez który następuje odwołanie do metod *usługi,* wymagało zdefiniowania klasy *MyBinder* oraz interfejsu *ServiceConnection,* w oparciu o treść poradnika [65]. Fakt bezpośredniej interakcji użytkownika ze smartfonem został wykorzystany, w celu poproszenia go o ewentualne włączenie funkcji GPS. Prośba wyświetlana jest w sytuacji, gdy moduł GPS jest wyłączony i ma formę okna dialogowego, zawierającego informacje na temat wykorzystywanej usługi lokalizacyjnej Google Play [66]. W zależności od podjętej przez użytkownika decyzji, wysłana wiadomość SMS zawiera współrzędne bieżącej lub ostatnio zarejestrowanej lokalizacji. Zapytanie nie jest wyświetlane w przypadku użycia przycisku zewnętrznego, ponieważ użytkownik nie ma możliwości odpowiedzenia na nie. Uruchomienie funkcji GPS (Bluetooth, WiFi itp.) przez aplikację w sposób automatyczny (tzn. bez wyświetlenia okna dialogowego lub bez przekierowania użytkownika do ustawień smartfona) jest w systemie Android niemożliwe z przyczyn bezpieczeństwa. Obowiązkowe aktywowanie funkcji GPS podczas uruchamiania aplikacji (w momencie konfigurowania połączenia Bluetooth) może się wiązać ze sceptycznym nastawieniem potencjalnych użytkowników, wobec czego wybór ten pozostanie opcjonalny.

Wybór opcji „*RUN WITHOUT BT DEVICE”* w ekranie konfigurującym połączenie BT uruchamia aplikację w trybie pracy nie wymagającym obecności przycisku zewnętrznego. Przesłany do klasy *BluetoothService* adres MAC (zawierający treść „null”) wskazuje na to, że użytkownik nie życzy sobie nawiązywać połączenie Bluetooth z jakimkolwiek urządzeniem peryferyjnym. Wówczas rola *usługi* ogranicza się jedynie do udostępniania metod publicznych.

2.2.8. Przechowywanie danych użytkownika

Zapisywanie danych pozwala spersonalizować aplikację i ułatwia dostosowanie jej do potrzeb użytkownika. Aplikacja będzie współpracować zazwyczaj z tym samym urządzeniem zewnętrznym o niezmiennym adresie MAC, który zostaje zapisany w wyniku poprawnie nawiązanego połączenia Bluetooth. Podczas kolejnego uruchamiania aplikacji adres ten jest wczytywany jako domyślny – wystarczy zatwierdzić go wybierając *CONFIRM*. W efekcie procesy konfigurowania połączenia Bluetooth został uproszczony i przyspieszony.

Do zapisywania i wczytywania danych została użyta klasa SharedPreferences. Uzyskanie dostępu do pliku o wskazanej nazwie (ewentualnie wcześniejsze utworzenie go) jest możliwe dzięki wykorzystaniu metody *getSharedPreferences()* [67]. Wywoływana jest w odpowiedzi na wiadomość *SUCCESS_CONNECT uchwytu mHandler* (wewnątrz klasy *BluetoothService).* Później następuje zapisanie adresu MAC w 3 etapach:

- otwarcie edytora pliku (metoda *edit()*);

- wprowadzenie danych (metoda *putString(), putBoolean(), putInt()* itd.) na zasadzie podania pary klucz-wartość;
- zapisanie zmian (metoda *commit()*);

Do wczytania elementu z pliku wykorzystywana jest odpowiednio jedna z metod *getString()*/*getBoolean()*/*getInt()* itp. Znalazła ona zastosowanie np. podczas tworzenia ekranu umożliwiającego konfigurowanie połączenia Bluetooth (funkcja *loadLastMacAddress()* klasy *BluetoothListFragment*). Zwraca wartość identyfikowaną poprzez klucz (argument pierwszy), lecz jeśli jej się to nie uda, zwracana jest wartość domyślna (argument drugi). Klasa *SharedPreferences* jest dużo prostsza w użyciu niż baza danych *SQLite*, która wymaga zaimplementowania dodatkowych klas i w praktyce może zachowywać się w sposób zależny od urządzenia, na którym aplikacja się znajduje.

Utrudnienie w korzystaniu z *SharedPreferences* wynika z możliwość zapisywania jedynie danych typu prostego. Obiekty bardziej skomplikowane podlegają serializacji, np. poprzez użycie metody *toJson()* (*należącej do biblioteki Gson*, załadowanej tak samo jak biblioteka *ButterKnife)*. Przekształcony w ten sposób obiekt może zostać zapisany w pliku utworzonym przez klasę *SharedPreferences*. Przykładem obiektu, wymagającego przekształcenia do formatu JSON, jest lista *wzorów „patternList”* (zdefiniowana wewnątrz klasy *EmergencyViewFragment*). Przywrócenie jej do użytkowej postaci (po wczytaniu z pliku) wymaga skorzystania z metody *fromJson()*, dla której jednym z argumentów jest typ danych - dla podanego przykładu „*List<EmergencyObject>”*. Zapisanie listy oszczędza użytkownikowi uzupełniania danych dotyczących wiadomości SMS po każdorazowym uruchomieniu aplikacji.

2.2.9. Działanie aplikacji

Działanie aplikacji zostało przetestowane w sposób szczegółowy na telefonie Samsung Galaxy XCover, model SM-G388F, z zainstalowanym systemem Android w wersji 5.1.1. Testy pozwoliły wykryć i wyeliminować problemy związane z niekontrolowanym resetowaniem połączenia Bluetooth podczas przesyłania danych pochodzących ze zaktualizowanych *wzorów* do *usługi BluetoothService*, a także niewłaściwą obsługę połączenia podczas pierwszego uruchomienia aplikacji na smartfonie (gdy plik wykorzystywany przez klasę *SharedPreferences* był jeszcze pusty). Jest to powód, dla którego niemal wszystkie klasy, mające związek z *usługą BluetoothService*, przesyłają adres MAC o treści *„empty”*, co pozwala uniknąć resetowania raz utworzonego połączenia Bluetooth.

Testom podlegało działanie funkcji przycisku bezpieczeństwa w sytuacjach, gdy ekran użytkownika był widoczny (aplikacja aktywna), gdy używana były inne aplikacje (np. odtwarzacz multimedialny

VLC, przeglądarka Chrome) oraz, gdy był on zablokowany. W każdej z nich naciśnięcie przycisku bezpieczeństwa skutkowało przesłaniem wiadomości SMS ze zaktualizowanymi współrzędnymi położenia, której przykład widoczny jest na Obrazie 7.

W trakcie testu trwającego 18h, w warunkach normalnej pracy smartfona bez konieczności jego ładowania, przetestowana została trwałość połączenia Bluetooth oraz zużycie baterii przez aplikację. Połączenie uległo zerwaniu 1 raz, czego powodem była konieczność zwolnienia zasobów przez system, skutkująca tzw. „zabijaniem" niektórych procesów działających w tle. Problem w tej sytuacji stanowi fakt, że smartfon nie inicjuje automatycznie połączenia z modułem HC-05 i wykorzystywana przez aplikację biblioteka *BluetoothDevice* nie udostępnia metod, które stwarzałyby taką możliwość. Bateria smartfona została w czasie testu rozładowana niemal całkowicie, do czego, jak informują narzędzia systemowe, testowana aplikacja przyczyniła się w 9%, a funkcje wyświetlacza oraz przeglądarki internetowej w 56%.

Aplikacja przetestowana została również pod kątem reagowania na dane przesyłane (z wykorzystaniem portu szeregowego) z innych źródeł niż przycisk bezpieczeństwa. Podczas jej działania w tle, przeprowadzono transmisję danych przez Bluetooth pomiędzy laptopem a smartfonem. Zakończyła się ona sukcesem i nie wpłynęła w zauważalny sposób na funkcjonowanie aplikacji.

3. Podsumowanie

Założenia pracy, polegające na zbudowaniu prostego przycisku bezpieczeństwa oraz napisaniu aplikacji mobilnej korzystającej z jego funkcjonalności, zostały spełnione.

Przycisk ma niewielkich rozmiarów urządzeniem, którego rdzeń stanowi płytka bazowa Arduino Micro, a komponenty zostały umieszczone na płytce obwodu drukowanego. Wyposażenie przycisku w moduł HC-05 umożliwiło komunikację bezprzewodową ze smartfonem, zgodną ze standardem Bluetooth 2.0+EDR. Rozwiązanie to pozwala na swobodne przemieszczanie przycisku w promieniu około 10m od smartfona, przy zachowaniu jego pełnej funkcjonalności, co znacznie wpływa na wygodę użytkowania względem innych, rozważanych metod połączenia.

Zasilanie z baterii CR123 o napięciu 3V umożliwia nieprzerwaną pracę przycisku przez około 78h, a wykorzystanie do tego celu portów microUSB oraz USB pozwala w razie potrzeby na zastąpienie baterii przez bank energii (*ang. power bank)*.

Aplikacja została napisana w taki sposób, by reagować na dane wysyłane z portu szeregowego Arduino Micro. Rozwiązanie to nie wywołuje konfliktu z innymi aplikacjami podczas użycia przycisku, co stanowi różnice w porównaniu do pilotów o charakterze klawiatury multimedialnej, które są źródłem sygnałów błędnie interpretowanych przez smartfon (np. zdalne wyzwalacze migawki aparatu powodujące zwiększenie głośności dźwięku).

Aplikacja ma charakter programu działającego w tle, co oznacza, że możliwe jest jednoczesne korzystanie z niej oraz innych funkcji smartfona. Dzięki zastosowaniu tzw. „wielowątkowości" użycie aplikacji nie powoduje tymczasowym zawieszeniem pozostałych operacji wykonywanych przez smartfon.

Interfejs umożliwia użytkownikami skonfigurowanie niezbędnych ustawień aplikacji – połączenia ze wskazanym urządzeniem Bluetooth oraz wprowadzenia danych wysyłanych na skutek użycia przycisku bezpieczeństwa w wiadomości SMS. Funkcja ta jest również wywoływana z poziomu interfejsu użytkownika. Zapisywanie danych w pamięci smartfona oraz ich wczytanie przy ponownym uruchomieniu aplikacji stanowi dodatkowe ułatwienie procesu konfiguracji.

Elementy interfejsu mają nieskomplikowaną, intuicyjną postać. Stosunkowo duże odstępy pomiędzy przyciskami, zastosowanie czcionek dużego rozmiaru oraz wykorzystanie okien dialogowych, potwierdzających wybory dokonywane przez użytkownika to czynniki mające na celu zwiększenie komfortu używania aplikacji i zapobiegnięcie wprowadzeniu przypadkowych zmian. Dodatkowe wsparcie stanowi wyświetlanie krótkich komunikatów informujących użytkownika o stanie aplikacji oraz o efekcie podjętych przez niego czynności.

Wiadomość SMS, wysyłana po naciśnięciu przycisku bezpieczeństwa, zawiera treść przygotowaną wcześniej przez użytkownika oraz jego możliwie najbardziej aktualne oraz dokładne współrzędne

położenia. Do pozyskania ich wykorzystywany jest moduł GPS znajdujący się w smartfonie. Ze względu na relatywnie krótki czas jego wykorzystania, który wynosi 60 sekund od chwili naciśnięcia przycisku, powodowany przez aplikację wzrost zużycia energii jest nieznaczny. W normalnych warunkach pracy smartfona, zużycie baterii smartfona przez aplikację w ciągu dnia wyniosło %, co nie powinno powodować konieczności jego dodatkowego ładowania.

Problem dotyczący automatycznego nawiązywania połączenia, gdy urządzenia znajdą się we wzajemnym zasięgu, nie została rozwiązany. Powodem jest brak odpowiedniego wsparcia ze strony wykorzystywanej biblioteki *BluetoothDevice* twórców systemu Android. Rozwiązaniem nie wymagającym zmiany standardu Bluetooth (a więc wymiany modułu HC-05 na nowszy), jest rejestrowanie zdarzenia, gdy połączenie Bluetooth z urządzeniem zostanie zerwane (*ACTION_ACL_DISCONNECTED* [68]) i jego cykliczne wyszukiwanie do momentu, aż znajdzie się ono w zasięgu komunikacji, co jest operacją kosztowną ze względu na użycie zasobów smartfona. Istnieje wiele sytuacji (np. wyłączenie przycisku przez użytkownika bez zatrzymania działania aplikacji), gdy rozwiązanie to może wywołać niepożądane skutki działania aplikacji (m.in. szybkie rozładowanie baterii telefonu).

4. Perspektywy rozwoju

Najważniejszym etapem dalszego rozwoju przycisku bezpieczeństwa jest wymiana modułu HC-05 na nowszy, np. HM-10/11, funkcjonujący w standardzie Bluetooth v4.0 Low Energy. Oprócz zmniejszenia ilości pobieranego ładunku z baterii zasilającej urządzenie, możliwe byłoby zmodyfikowanie kodu obsługującego komunikację Bluetooth w taki sposób, aby bazował on na klasie *BluetoothGatt* [69]. Dzięki temu możliwa byłaby współpraca aplikacji z urządzeniami opartymi na *Bluetooth Low Energy* oraz efektywne wykorzystanie ich możliwości. Dotyczy to również automatycznego inicjowania połączenia ze sparowanymi modułami, gdy znajdą się one w zasięgu komunikacji (funkcja *connectGatt* [70]), który efektywnie zwiększyłby się również do około 30m.

Dodatkową korzyść mogłoby przynieść zastąpienie płytki bazowej Arduino przez mikrokontroler z rodziny AVR (np. ATtiny). Zostałby zmniejszony rozmiar przycisku bezpieczeństwa oraz prawdopodobnie prąd potrzebny do zasilania układu. Alternatywnym, prostszym rozwiązaniem byłoby zastosowanie płytki rozwojowej zawierającej mikrokontroler i wbudowany moduł umożliwiający komunikację Bluetooth (przykładowo: [71, 72]).

Planowane jest także wzbogacenie aplikacji o dodatkowe funkcje. Pierwsza dotyczy nawiązanie połączenia telefonicznego w trybie głośnomówiącym, z wprowadzonym wcześniej przez użytkownika numerem telefonu, na skutek przytrzymania przycisku awaryjnego przez dłuższy czas (np. 2 sekundy). Kolejna, umożliwiająca wprowadzenie informacji na temat przyjmowanych przez użytkownika lekach oraz częstotliwości ich dawkowania, stanowiłaby pasywną część aplikacji. Stworzyłaby możliwość przypomnienia o zażyciu farmaceutyków przy użyciu funkcji alarmu. Dodatkowa informacja, związana z tym zdarzeniem, mogłaby trafiać do opiekuna osoby chorej. Obie wyżej opisane funkcje mogłyby znaleźć zastosowanie np. w domowych systemach opieki nad chorym. Smartfony wyposażone są także w akcelerometry, które umożliwiają detekcję drgań. Fakt ten mógłby być wykorzystany, aby nadać smartfonowi rolę czujnika wystąpienia ataku epileptycznego. Jego zarejestrowanie wiązałby się z odpowiedzią aplikacji, identyczną do naciśnięcia przycisku bezpieczeństwa (ataki nie są poprzedzone żadnymi symptomami, przez co funkcjonalność przycisku jest w ich kontekście bezużyteczna).

Planowane jest także poprawienie estetyki i funkcjonalności przycisku poprzez umieszczenie go w obudowie wykonanej w technice druku 3D.

Z myślą o użytkownikach posługujących się językiem niż angielski, zawartość pliku *String.xml* należałoby przetłumaczyć na polski, niemiecki i francuski, aby stworzyć możliwość wyboru różnych wersji językowych.

roid.content.Context,%20boolean,%20android.bluetooth.BluetoothGattCallback)

http://profandroid.com/network/bluetooth/bluetooth-auto-connect.html

https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html

Bibliografia:

1. http://pclab.pl/news69812.html
2. https://www.accenture.com/pl-pl/insight-fjord-trends-2016
3. http://www.pcworld.idg.com.au/article/430254/why_japanese_smartphones_never_went_global/?
4. http://searchnetworking.techtarget.com/definition/i-Mode
5. http://www.osnews.com/story/27416/The_second_operating_system_hiding_in_every_mobile_phone
6. https://prezi.com/ujefpyr7cvtk/real-time-operating-system-and-smartphone/
7. http://www.gartner.com/newsroom/id/3415117
8. http://www.openhandsetalliance.com/oha_members.html 27.11.16
9. http://www.tiobe.com/tiobe-index/ 27.11.16
10. https://developer.android.com/studio/run/emulator.html
11. http://android.com.pl/news/40618-android-studio-ide-2016/
12. http://android-developers.blogspot.in/2013/05/android-studio-ide-built-for-android.html
13. https://www.jetbrains.com/idea/features/
14. http://www.vogella.com/tutorials/AndroidBuild/article.html#gradle-for-building-android-applications
15. https://developer.android.com/guide/topics/manifest/manifest-intro.html
16. https://developer.android.com/guide/components/fragments.html
17. https://developer.android.com/training/basics/data-storage/databases.html
18. http://www.android4devs.pl/2011/07/sqlite-androidzie-kompletny-poradnik-dla-poczatkujacych/
19. https://developer.android.com/training/basics/data-storage/shared-preferences.html
20. http://www.android4devs.pl/2011/08/sharedpreferences-zapisywanie-danych-aplikacji/
21. https://github.com/google/gson
22. https://source.android.com/security/encryption/
23. https://source.android.com/devices/accessories/headset/specification.html
24. http://get.pressybutton.com/
25. http://www.androidpolice.com/2014/08/26/pressy-review-i-had-no-clue-something-could-suck-this-badly/
26. https://db.zmitac.aei.polsl.pl/BZ/archive/wl_irda.pdf
27. http://mit.weii.tu.koszalin.pl/MIT1/Modele%20inzynierii%20teleinformatyki%201_12%20Teodorowicz%20Ryngwelski.pdf
28. http://www.pg.gda.pl/~zbczaja/pdf/mse/irda.pdf
29. http://isap.sejm.gov.pl/DetailsServlet?id=WDU20140001843
30. http://www.intel.com/content/www/us/en/io/universal-serial-bus/usb3-frequency-interference-paper.html 17.11.16
31. http://kti.eti.pg.gda.pl/ktilab/bluetooth/Bluetooth%20-%20Instrukcja%20laboratoryjna%20Teoria.pdf
32. https://piratecomm.wordpress.com/2014/01/19/ble-pairing-vs-bonding/ 24.11.16
33. http://blog.bluetooth.com/bluetooth-pairing-part-1-pairing-feature-exchange/?_ga=1.192186537.2022653936.1476829960 25.11.16
34. http://www.differencebetween.net/technology/difference-between-bluetooth-2-0-and-bluetooth-2-1/ 23.11.16
35. http://www.ellisys.com/technology/een_bt07.pdf
36. https://learn.sparkfun.com/tutorials/bluetooth-basics/how-bluetooth-works 20.11.16
37. http://ccm.net/contents/69-bluetooth-how-it-works 25.11.16
38. https://faculty.cs.byu.edu/~knutson/publications/IrDA_Assisted_BT_Discovery.pdf 26.11.16
39. http://bluetoothreport.com/bluetooth-versions-comparison-whats-the-difference-between-the-versions/ 26.11.16

40. http://www.newswireless.net/index.cfm/article/629 10.11.16
41. https://learn.sparkfun.com/tutorials/serial-communication 30.11.16
42. http://forbot.pl/blog/artykuly/programowanie/kurs-arduino-3-uart-komunikacja-z-pc-zmienne-id3836 01.12.16
43. https://www.arduino.cc/en/Guide/ArduinoLeonardoMicro#toc9 02.12.16
44. https://www.arduino.cc/en/Main/ArduinoBoardMicro 03.12.16
45. https://www.arduino.cc/en/Products/Compare 03.12.16
46. http://cdn.makezine.com/uploads/2014/03/hc_hc-05-user-instructions-bluetooth.pdf 04.12.16
47. http://www.thaieasyelec.com/downloads/EFDV390/EFDV390_Datasheet.pdf 04.12.16
48. https://www.element14.com/community/docs/DOC-82933/l/the-intel-genuinoarduino-101-eagle-library
49. https://github.com/ErichStyger/mcuoneclipse
50. http://www.plociennik.info/index.php/transmisja-szeregowa 07.12.16
51. https://www.arduino.cc/en/Reference/Millis 07.12.16
52. https://developer.android.com/about/versions/android-4.4.html 13.12.16
53. http://jakewharton.github.io/butterknife/ 18.12.16
54. https://developer.android.com/guide/topics/connectivity/bluetooth.html 21.12.16
55. https://www.bluetooth.com/specifications/assigned-numbers/service-discovery 21.12.16
56. http://javastart.pl/static/zaawansowane-programowanie/watki-wprowadzenie-i-przyklad/ 22.12.16
57. https://developer.android.com/reference/java/lang/Thread.html 22.12.16
58. https://developer.android.com/reference/android/location/package-summary.html 23.12.16
59. https://developers.google.com/android/guides/overview 23.12.16
60. https://developers.google.com/android/guides/setup 23.12.16
61. https://developer.android.com/training/location/index.html 23.12.16
62. https://developers.google.com/android/reference/com/google/android/gms/location/package-summary 23.12.16
63. https://android-developers.googleblog.com/2013/10/getting-your-sms-apps-ready-for-kitkat.html 25.12.16
64. https://developer.android.com/guide/components/services.html 26.12.16
65. https://developer.android.com/guide/components/bound-services.html 26.12.16
66. https://developers.google.com/android/reference/com/google/android/gms/location/SettingsApi 27.12.16
67. https://developer.android.com/training/basics/data-storage/shared-preferences.html 29.12.16
68. https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html 3.01.16
69. https://developer.android.com/guide/topics/connectivity/bluetooth-le.html 3.01.16
70. https://developer.android.com/reference/android/bluetooth/BluetoothGatt.html 3.01.16
71. http://bleduino.cc/ 3.01.16
72. http://redbearlab.com/blenano/ 3.01.16

Brudnopis:

// bluetooth włączany podczas wcisnięcia guzika?

//użyteczność pod koniec, jak wyjaśni się chociaż sprawa jak to będzie wyglądać i z czego korzystać

//finalna postać? (powiąż to z użytecznością i wymaganiami stawianymi/kryteriami wyboru)

//wymagania na podstawie których wybór gdzieś przy realizacji

// rozwiązanie dla przepełnienia unsigned long przez millis()?

// delete też weryfikować dodatkowym oknem

wyk1

http://www.gartner.com/newsroom/id/3415117

//https://en.wikipedia.org/wiki/Mobile_operating_system#/media/File:World_Wide_Smartphone_Sales.png

budowanie – budowanie to proces, który obejmuje m.in. kompilacje (patrz poniżej) – ogólnie rzecz ujmując budowanie to proces przekształcania wszystkich kodów źródłowych i innych plików w projekcie (module) w artefakt – postać gotową do użycia w innym projekcie/module (lub w serwerze aplikacji)

//HM-10 opisać dopiero