Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka

w specjalności Systemy Informacyjno-Decyzyjne

Aplikacja „Tu byłem” śledząca trasy użytkowników za pomocą znaczników NFC

Marcin Kiliański

Numer albumu 293119

promotor

dr inż. Mariusz Kamola

Warszawa 2021

Aplikacja „Tu byłem” śledząca trasy użytkowników za pomocą znaczników NFC

Streszczenie: Tematem pracy jest aplikacja na system Android służąca do weryfikacji poprawności przebytej przez użytkownika, wcześniej zaplanowanej trasy. Przykładami takiej trasy mogą być wyścig terenowy lub kontrola obchodu strażnika. Trasę wyznaczają tagi Near Field Communication (NFC), które w trakcie obchodu użytkowników są przez nich skanowane. Aplikację można podzielić na dwie części – część dostępną dla użytkownika oraz na część dedykowaną dla administratora (np. organizatora wyścigu). Ta pierwsza wyposażona jest przede wszystkim w funkcje skanowania i zapisywania tagów NFC, natomiast druga odpowiedzialna jest za zaplanowanie trasy oraz późniejszą weryfikację.

Słowa kluczowe: aplikacja mobilna, komunikacja bliskiego zasięgu, znacznik NFC, gromadzenie danych, weryfikacja danych.

„I was here” application that tracks users routes using NFC tags

Abstract: The subject of this thesis i san Android application used to verify the correctness of the planned route traveled by the users. Best examples of such a route are off-road race or a guard round. The route is marked by Near Field Communication (NFC) tags, which are scanned by the users during the race. The application can be divided into two parts – the part available only for administrator and the part for competitors. The first one contains modules for route planning and integrity verification, while the second one primarily contains functions of scanning and saving data from NFC tags.

Key words: mobile application, near field communication, NFC tag, data collection, verification of data.

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

WARSZAWA 10.05.2021 r.
miejscowość i data

MARCIN KILIAŃSKI
imię i nazwisko studenta

293119
numer albumu

INFORMATYKA
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

Marcin Kiliański
czytelny podpis studenta

Spis treści

1. Wstęp

Telefony komórkowe pełnią aktualnie nieodzowną część naszego życia. Szybki rozwój technologiczny sprawił, że możemy za ich pomocą robić większość rzeczy, o których jeszcze kilka lat temu ciężko było sobie wyobrazić. Urządzenia mobilne posiadają podzespoły zdolne do komunikacji z innymi urządzeniami mobilnymi, komputerami, routerami, dają również możliwość korzystania z takich możliwości jak Global Positioning System (GPS). Dzięki temu za pomocą telefonu możemy bez problemu komunikować się z innymi, opłacić rachunki w banku, sprawdzić swoją lokalizację czy zapłacić w sklepie za zakupy. Tak wiele możliwości wykorzystania urządzeń mobilnych oraz ich coraz to lepsze parametry techniczne pozwalają developerom na tworzenie coraz to bardziej zaawansowanych, sprawniejszych i użyteczniejszych aplikacji.

W życiu codziennym często spotykamy się z różnymi systemami i aplikacjami służącymi do określania naszej lokalizacji, wyznaczania optymalnych tras dojścia do celu, aż po aplikacje pozwalające śledzić przebyte trasy przez użytkownika. Praktycznie wszystkie dostępne aplikacje wymagają do poprawnego działania technologii GPS lub/oraz dostępu do Internetu. Jednak trzeba pamiętać, że nie zawsze mamy dostęp do wcześniej wspomnianych technologii, ponieważ nie wszystkie obszary są pokryte ich zasięgiem. Dodatkowo używanie w telefonie modułu lokalizacji, Wi-Fi, czy transmisji Global System for Mobile Communications (GSM) znacząco zwiększa zużycie baterii telefonu, co w niektórych warunkach może być bardzo ważne.

Na rynku ciężko znaleźć aplikację służącą do śledzenia lokalizacji użytkownika, działającą bez użycia wyżej wymienionych modułów telefonu, dlatego celem niniejszej pracy jest stworzenie aplikacji, która spełni te wymagania. Działanie aplikacji oparte będzie na tagach NFC, służących jako punkty kontrolne na trasie wyścigu bądź obchodu strażnika. Każdy z nich będzie przechowywał zaszyfrowaną wiadomość, którą użytkownicy będą musieli zeskanować. W celu zapobiegania naruszeniom integralności danych, każdy użytkownik jednocześnie będzie nadpisywał wiadomość w tagu za pomocą swojego identyfikatora tak, aby stworzyć sieć powiązanych ze sobą danych. Dane zebrane przez użytkowników w trakcie obchodu zostaną użyte w celu ustalenia ciągu wydarzeń, co umożliwi wykrycie potencjalnych celowych naruszeń integralności oraz pomoże uzupełnić ewentualne braki danych.

Na wstępie chciałbym również zdefiniować pojęcie wyścigu, które w niektórych sytuacjach mogłoby być mylące. W niniejszej pracy wyścig będzie rozumiany jako jeden z etapów organizacji zawodów sportowych (np. biegu terenowego), podczas którego to użytkownicy wyposażeni w urządzenia mobilne z zainstalowaną aplikacją, będą poruszali się od startu do mety po zaplanowanej wcześniej trasie. Trasa będzie wyznaczona przez punkty kontrolne, przy których uczestnicy będą musieli wykonać konkretne akcje, opisane w dalszych częściach pracy. Wyścig kończy się dotarciem do ostatniego punktu kontrolnego, którym jest meta. Po ukończeniu wyścigu przez wszystkich uczestników następują kolejne etapy organizacji zawodów.

2. Szczegółowa koncepcja aplikacji

Tematem niniejszej pracy jest stworzenie aplikacji mobilnej na system Android, która będzie umożliwiać zaplanowanie trasy oraz sprawdzenie, czy użytkownicy przeszli po niej w prawidłowy sposób i czy nie dopuścili się żadnych naruszeń integralności. Najważniejszą cechą aplikacji jest fakt, że jest aplikacją działającą całkowicie offline oraz nie korzysta z systemu GPS, który mógłby stać się słabym punktem aplikacji. Najważniejszymi wadami systemu GPS w telefonie są bez wątpienia wysokie zużycie baterii oraz możliwość wpłynięcia przez użytkownika na odczyt lokalizacji urządzenia (np. za pomocą innych aplikacji). Jedyną funkcjonalnością telefonu, jaką będzie wykorzystywać jest łączność Near Field Communication (NFC), która będzie umożliwiać nam przede wszystkim wymianę danych ze znacznikami NFC. Opis i specyfikacja technologii NFC jak i samych znaczników znajduje się w następnych rozdziałach.

Warunek opierania się wyłącznie na technologii NFC sprawia wiele problemów technicznych i koncepcyjnych. Pierwszym z nich jest potrzeba fizycznego zaplanowania trasy. Oznacza to tyle, że nie możemy trasy zaplanować tylko i wyłącznie na mapie bądź w aplikacji obsługującej GPS. Trasa będzie musiała mieć swoją reprezentację terenową. Do tego posłużą wcześniej wspomniane znaczniki NFC, które będą pełniły rolę swego rodzaju punktów kontrolnych. Każdy użytkownik powinien znać trasę (np. powinien posiadać mapę z zaznaczonymi tagami i kolejnością ich skanowania) oraz powinien być wyposażony w urządzenie z aplikacją. Ważne, aby na urządzeniu nie znajdowały się aplikacje umożliwiające nadpisywanie i skanowanie tagów, ponieważ mogło by to ułatwić dokonanie sabotażu (np. usunięcie danych ze znacznika, przez co kolejni zawodnicy nie będą w stanie odczytać z niego danych).

Kolejnym problemem jest sposób przechowywania danych, którymi aplikacja będzie operować, oraz sposób ich przetwarzania. Nie mamy tu możliwości wysłania danych bezpośrednio do serwera zaraz po odczytaniu informacji z tagu, więc dane będą musiały być przechowywane tylko i wyłącznie w urządzeniach użytkowników oraz w znacznikach. Co oczywiste nie chcemy jednak, aby użytkownik miał łatwy dostęp do tych danych, ponieważ mógłby wykorzystać je do zdobycia przewagi nad innymi. Co więcej, warto by również powiązać ze sobą w jakiś sposób kolejno odczytywane i zapisywane dane, tak aby możliwie jak najbardziej utrudnić wprowadzanie pozornie prawidłowych, lecz jednak fałszywych danych. Rozwiązaniem tych problemów jest łańcuch danych (ang. block chain), dzięki któremu w prosty sposób zaszyfrujemy dane, a informacje pobierane z kolejnych znaczników będą uzależnione od siebie. Dzięki temu użytkownik nie będzie w stanie ręcznie dodać brakujących bloków (ponieważ wtedy bloki te nie będą zależne) oraz nie będzie w stanie dodać pozornie prawidłowych danych, jeśli nie będzie znać zasady działania funkcji hashującej[1]. Szczegółowy opis użycia block chain zostanie opisany w kolejnych rozdziałach.

Ostatnią kwestią, jaką należało poruszyć w pracy jest weryfikacja poprawności danych zebranych przez użytkowników. Brak natychmiastowego wysyłania informacji o użytkowniku i jego trasie do zewnętrznego serwera bardzo mocno zwiększa szansę na wystąpienie wszelakiego rodzaju błędów – od utraty danych do celowych, negatywnych działań użytkowników na swoją korzyść. Dzięki zastosowaniu block chain do zapisu danych gromadzonych przez użytkowników, wiele z tych problemów będzie można rozwiązać

[1] Inaczej funkcja skrótu – funkcja generująca dla dowolnych danych wiadomość o stałym rozmiarze, jest nieodwracalna.

odpowiednim algorytmem sprawdzającym poprawność danych, a próby naruszenia integralności będzie można wykryć i np. zdyskwalifikować odpowiednich uczestników. Weryfikację danych będziemy zaczynać od zebrania danych wszystkich użytkowników po ukończeniu trasy. Brak danych od któregoś z zawodników całkowicie uniemożliwi sprawdzenie poprawności danych, więc jest to bardzo ważny warunek. Dane zebrane od użytkowników będą zmodyfikowane przez funkcję hashującą, więc nie będzie możliwe zwykłe sprawdzenie powiązania ze sobą danych i wyszukania pewnych luk. Funkcja weryfikująca na podstawie początkowych danych z tagów oraz indywidualnych kluczy użytkowników będzie odtwarzać ciąg wydarzeń na trasie i porównywać wyniki z tymi, które uzyskali użytkowników na trasie. W przypadku, gdy wszyscy użytkownicy dostarczyli poprawne dane, funkcja powinna odtworzyć identyczne łańcuchy danych z tymi, które dostarczyli użytkownicy. W przeciwnym wypadku funkcja będzie potrafiła wyszukać ewentualne braki danych i określić, czy są one spowodowane złymi intencjami użytkowników czy po prostu wynikają z błędów w samej aplikacji. Szczegółowy opis algorytmu weryfikacji jest opisany w kolejnych rozdziałach.

3. Narzędzia i opis wykorzystanych technologii

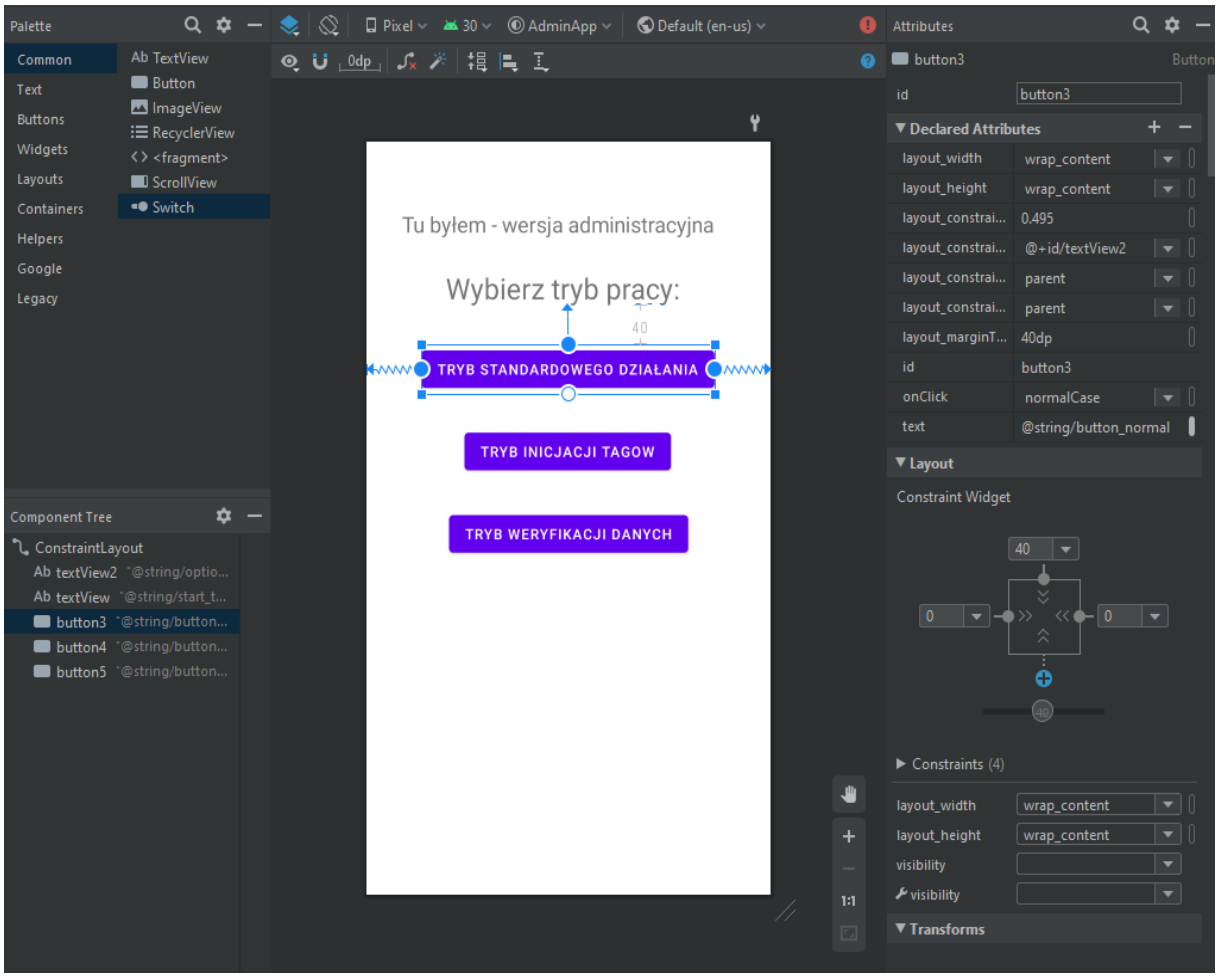
3.1 System Android

Android jest systemem operacyjnym dedykowanym wszelkim urządzeniom mobilnym. Początkowo był rozwijany przez firmę Android Inc., która następnie została przejęta przez Google. System ten jest aktualnie najpopularniejszym systemem mobilnym na świecie[2]. Jego popularność miała bardzo duży wpływ przy wyborze systemu docelowego, ponieważ łatwiej wejść w posiadanie urządzenia, które z niego korzysta, oraz sami użytkownicy powinni być lepiej zaznajomieni z jego obsługą, niż z obsługą np. sytemu iOS. Android dostarcza również wielu narzędzi przydatnych dla developerów takich jak na przykład Android Studio, w którym została stworzona aplikacja omawiana w tej pracy. Najnowszą oficjalną wersją systemu jest 11, wydana pod koniec 2020 roku. Android jest udostępniany na zasadzie licencji open-source, dzięki czemu każda z wersji systemu ma zapewnioną długą żywotność nawet po rezygnacji z oficjalnego wsparcia wersji, poprzez aktualizacje tworzone przez społeczność. Prócz tego, społeczność tworzy także swoje własne wersje systemu Android, zawierające modyfikacje, które nie zostały zawarte w oficjalnych wersjach.

3.2 Android Studio

Android Studio to oficjalne narzędzie dla developerów stworzone z myślą o tworzeniu aplikacji na systemy Android, oparte na środowisku IntelliJ wydanego przez firmę JetBrains[3]. Cechuje się wieloma wbudowanymi bibliotekami przeznaczonymi specjalnie dla urządzeń z systemem Android np. biblioteka wspierająca obsługę komunikacji NFC, czyli „android.nfc”. Podstawowymi językami programowania obsługiwanymi przez Android są Kotlin, Java oraz C++. Aplikacja opisana w niniejszej pracy opiera się na języku Java. Każde okno aplikacji jest osobną klasą rozszerzającą jedną z wielu klas podstawowych. Środowisko te oferuje opcję prostego tworzenia części wizualnej okien aplikacji metodą drag-and-drop (widoczny na rysunku nr 1). Sprawia to, że wizualną warstwę aplikacji jest w stanie stworzyć nawet osoba nie mająca wcześniej styczności z tworzeniem aplikacji. Tworzoną aplikację możemy na bieżąco testować za pomocą wbudowanego emulatora telefonu z systemem Android lub na fizycznym urządzeniu poprzez podłączenie go do komputera. Warto wspomnieć również, że program umożliwia łatwą publikację swoich aplikacji w Google Play Store[4].

[2] System Android posiada około 72% udziałów na rynku systemów operacyjnych. Źródło: https://gs.statcounter.com/ , dane za lipiec 2021r.
[3] Czeska firma zajmująca się dostarczaniem narzędzi do tworzenia oprogramowania dla developerów. Jej najpopularniejszym produktem jest środowisko IntelliJ.
[4] Internetowy sklep z aplikacjami i wszelkiego rodzaju multimediami zarządzany przez firmę Google.



Rysunek 1. Okno tworzenia warstwy wizualnej w Android Studio.

3.3 Near Field Communication

Near Field Communication (NFC – komunikacja krótkiego zasięgu) to standard komunikacji radiowej, cechujący działaniem na wysokich częstotliwościach i pozwalający na wymianę danych pomiędzy dwoma urządzeniami z bardzo małej odległości (do kilku centymetrów). Dzięki tak krótkiemu zasięgowi działania prawdopodobieństwo przechwycenia danych przez niechciane urządzenia jest bardzo niskie, dzięki czemu technologia ta dobrze sprawdza się np. przy obsłudze płatności mobilnych. Ważną zaletą NFC jest fakt, że technologia ta w odróżnieniu od na przykład Bluetooth czy Wi-Fi, nie potrzebuje wcześniejszego sparowania urządzeń w celu przesłania danych. Warto również wspomnieć o bardzo niskim poborze energii przez moduły obsługujące łączność NFC (poniżej 15 mA podczas odczytu danych oraz zerowy pobór mocy przez tagi przechowujące dane). Technologia NFC jest aktualnie rozwijana głównie przez serwis NFC Forum[5], powołany do życia w 2004 roku z inicjatywy trzech firm – Nokia, Sony i Phillips.

[5] Organizacja non-profit zrzeszająca ponad sto firm w celu promocji, rozwoju i standaryzowania technologii NFC.

Każde z urządzeń NFC może działać w jednym lub wielu z trzech różnych trybów. Pierwszym trybem działania jest peer-to-peer, który pozwala pomiędzy dwoma urządzeniami na wymianę danych takich jak tekst, zdjęcia czy danych o statusie danej aplikacji (pozwala to na przykład uruchomić wybraną aplikację w drugim urządzeniu). Kolejnym z trybów działania urządzeń NFC jest tryb symulacji kart płatniczych. Jest on obecnie wykorzystywany na szeroką skalę przy płatnościach mobilnych oraz systemach biletowych. Ostatnim z nich jest tryb odczytu/zapisu. Oznacza on tyle, że urządzenie w tym trybie jest w stanie odczytywać oraz nadpisywać dane zawarte w znacznikach NFC, które to są biernymi urządzeniami tj. same nie potrafią nawiązywać połączeń i służą jako kontenery prostych danych. Ten tryb działania będzie kluczowy dla tworzonej przeze mnie aplikacji. Tagi NFC będą pełniły tu rolę punktów kontrolnych, które użytkownicy systemu będą musieli odwiedzić i wykonać z nimi odpowiednie czynności. Dokładny proces zostanie przedstawiony w kolejnych rozdziałach.

Tabela 1. Tabela przedstawiająca porównanie wybranych parametrów technologii NFC oraz Bluetooth.

Rodzaj transmisji	NFC	Bluetooth
Szybkość transmisji	Do 424 kbit/s	Do 2,1 Mbit/s
Częstotliwość	13,56 MHz	2,4-2,5 GHz
Zasięg	Do 5 cm	Do 100 m
Prędkość nawiązania połączenia	< 0,1 s	Do 6 s
Typ sieci	Point-to-point	WPAN
Potrzeba zasilania tagu	Nie	Tak

3.4 Znaczniki NFC

Znacznik NFC (tag) – jest bardzo prostym urządzeniem, które umożliwia przechowywanie niewielkiej ilości danych (standardowo do około 1 kb). Zazwyczaj ma postać naklejki lub breloka o średnicy około 2-3 cm. Obecnie najczęściej używane są do uruchamiania na skanujących je urządzeniach różnego rodzaju poleceń, takich jak otwieranie stron internetowych, przekierowanie do konkretnych aplikacji w urządzeniu, nawiązywanie połączeń z dostępnymi sieciami Wi-Fi czy wysyłanie wiadomości SMS. Znaczniki NFC znajdują również zastosowanie przy wykonywaniu bardziej złożonych operacji, takich jak kontrola czasu pracy pracownika w firmie czy uzyskiwanie informacji o lokalizacji urządzenia.

Najważniejszą cechą tagów NFC jest bez wątpienia brak potrzeby ich zasilania, co sprawia że mogą być wykorzystywane w praktycznie każdych warunkach i nie potrzebują dodatkowego miejsca na baterię. Sprawia to, że tag nie jest w stanie sam nawiązywać połączenia z innymi urządzeniami i jest całkowicie zależny od urządzenia aktywnego, które podczas próby połączenia wytwarza pole elektromagnetyczne. Dzięki temu, że tag NFC zawiera w sobie obwód w kształcie cewki, indukowany jest w nim prąd, który następnie pobudza mikroprocesor i pozwala na wymianę danych pomiędzy urządzeniami. Używane przeze mnie

w czasie testów znacznik NFC NTAG216 (rysunek nr 2) mają postać naklejki, a ich średnica wynosi około 2 cm. Prosta budowa oraz zasady działania znaczników NFC zapewniają to, że są one bardzo tanie w produkcji, a cena przy zakupie większej ilości zwykle nie przekracza 1 zł za sztukę.

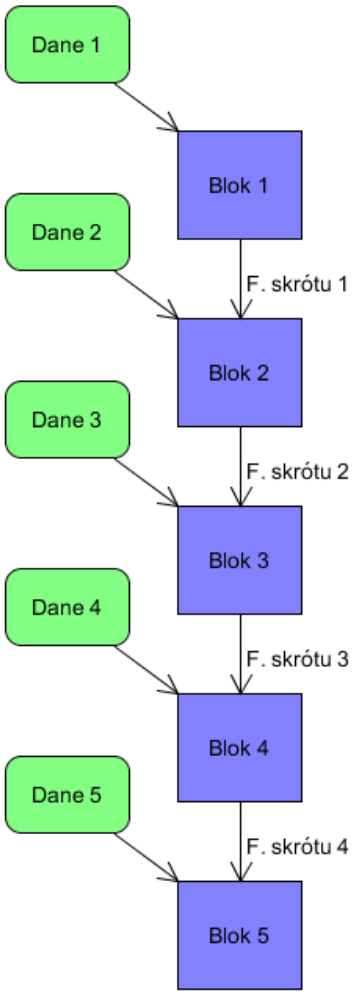
Rysunek 2. Tag NFC NTAG216 o pojemności 924 bajtów (zdjęcie własne).

3.5 Technologia rozproszonego rejestru

Technologia rozproszonego rejestru (ang. Distributed Ledger Technology, DLT) to sposób przechowywania danych w taki sposób, by były one mocno rozproszone po różnych bazach danych znajdujących się na różnych serwerach i w różnych lokalizacjach na świecie. Dzięki rozproszeniu danych, w przypadku uszkodzenia jednej z części bazy, nie stracimy wszystkich informacji, lecz co najwyżej ich część. Ten sposób przechowywania informacji cechuje się również częstą replikacją danych, co zwiększa zajmowaną przez bazę danych pamięć. Jednak dzięki temu rozwiązaniu w przypadku uszkodzenia jednego z nośników bazy danych, straty informacji będą jeszcze niższe, ponieważ utracone dane mogą znajdować się np. na innych serwerach. Oprócz tego, powielanie często używanych danych na różnych nośnikach pozytywnie wpływa na szybkość działania bazy, ograniczając wymianę informacji pomiędzy poszczególnymi serwerami.

W omawianej w niniejszej pracy aplikacji został użyty jeden z najpopularniejszych rodzajów DLT, czyli block chain. Jego najważniejszą zaletą, która przemawiała za użyciem go w aplikacji było ścisłe powiązanie ze sobą kolejnych bloków danych (schemat pokazany na rysunku nr 3). Pozwoli to częściowo zniwelować konsekwencje niepożądanych i nieprzewidzianych działań użytkowników, jak i błędów w przetwarzaniu samej aplikacji. Każdy blok powstaje z połączenia danych archiwalnych (np. skrót wiadomości z poprzedniego bloku), zapisywanej informacji oraz dodatkowych zmiennych mających za zadanie maksymalnie utrudnić rozszyfrowanie bloku. Takimi zmiennymi może być np. stempel czasowy czy liczba wykonanych operacji. Na koniec, tak utworzona wiadomość jest przetwarzana przez

odpowiednią funkcję kryptograficzną. Podczas tworzenia bloku nie można również zapomnieć o umieszczeniu w nim odnośnika do danych archiwalnych, czyli poprzedniego bloku. Block chain jest obecnie wykorzystywany na szeroką skalę przede wszystkim w dziedzinie kryptowalut[6].



Rysunek 3. Ogólny schemat budowy łańcucha danych (block chain).

3.6 Biblioteka android.nfc

Biblioteka android.nfc to zbiór użytecznych narzędzi służący do prostej obsługi wszystkich aspektów związanych z komunikacją NFC w systemie Android. Jest ona standardową biblioteką dostępną w środowisku Android Studio. Ma ona swoją reprezentację zarówno w języku Java, jak i w języku Kotlin. W opisywanej w niniejszej pracy aplikacji będzie służyć przede wszystkim do nawiązywania połączenia pomiędzy urządzeniem a znacznikami NFC oraz do odbioru i wysyłania do nich danych. W klasie zaimplementowane są również funkcje służące do tworzenia danych w formacie NDEF.

[6] Kryptowaluta to bazujący na kryptografii system księgowy, którego zadaniem jest przechowywanie informacji o stanie posiadania klienta w umownych jednostkach.

Biblioteka w wersji Java składa się przede wszystkim z czterech głównych, wymienionych niżej klas. Każda z nich została użyta w opisywanej aplikacji:

1. NfcManager – klasa zaimplementowana w bibliotece, służąca do przejęcia kontroli nad modułem NFC danego urządzenia. Jest klasą wyjściową dla wszystkich pozostałych klas.
2. NfcAdapter – klasa ta służy do utworzenia reprezentacji adaptera NFC w aplikacji. Bez tak utworzonej instancji adaptera nie jest możliwe wykonywanie jakiejkolwiek operacji z użyciem adaptera NFC takich jak na przykład pobranie informacji z tagu NFC czy ich zapisanie.
3. NdefMessage – klasa umożliwiająca reprezentację komunikatu danych w formacie NDEF, będącym standardowym formatem przechowywania danych używanym do komunikacji NFC. Komunikat NDEF służy jako kontener zawierający rekordy NDEF, który może być przesyłany pomiędzy urządzeniami. W pojedynczej wiadomości NDEF może znajdować się wiele rekordów NDEF.
4. NdefRecord – klasa reprezentująca właściwe dane przekazywane przez moduł NFC. Prócz samej przekazywanej wiadomości, zawiera również informacje o rodzaju przekazywanych danych.

4. Rozwiązanie

W niniejszym rozdziale zostaną opisane wymagania aplikacji, ich szczegółowe rozwiązania, algorytmy zawarte w aplikacji oraz przykładowe zestawy danych, na których aplikacja będzie działać.

4.1 Inicjalizacja znaczników NFC

Pierwszym krokiem, jaki należy wykonać w celu przeprowadzenia przykładowego wyścigu jest inicjalizacja znaczników, które będą służyły jako punkty kontrolne na trasie. Czynność ta będzie wykonywana z poziomu aplikacji administratora i zwykły użytkownik nie powinien mieć do niej dostępu. Kolejność inicjalizowania tagów będzie mieć kluczowe znaczenie, ponieważ będą one ze sobą kolejno powiązane poprzez umieszczanie funkcji skrótu aktualnie inicjalizowanego znacznika w kolejnym znaczniku. Inaczej mówiąc, kolejność inicjalizacji tagów będzie jednocześnie kolejnością, z jaką użytkownicy powinni zbierać dane z kolejnych punktów kontrolnych.

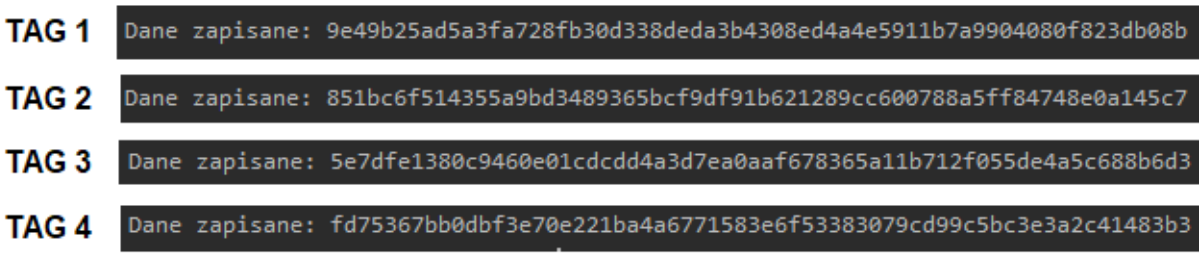
Wzór na wiadomość pozostawioną w n-tym tagu w czasie inicjalizacji przed operacją haszowania:

$$W_n = W_{n-1} + R,$$

gdzie R jest wartością pseudo losową lub wartością, która wcześniej znajdowała się w danym tagu.

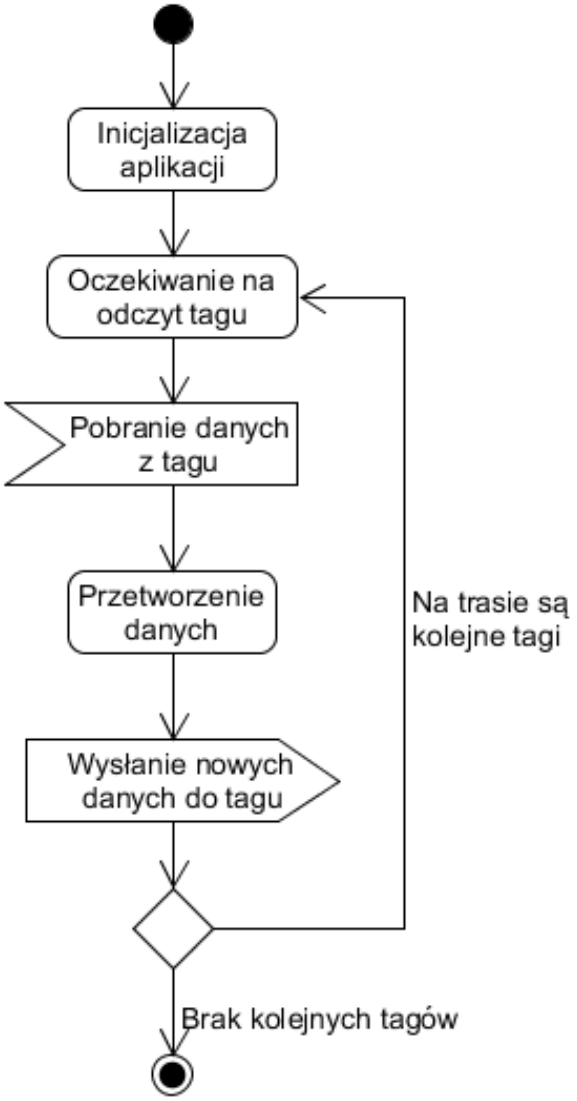
Wszystkie zapisywane do znaczników wartości początkowe muszą być również przechowywane przez aplikację na urządzeniu administratora. Będzie to niezbędne do przeprowadzenia weryfikacji poprawności danych dostarczonych przez użytkowników. Ostatnim krokiem na tym etapie organizacji wyścigu jest rozmieszczenie znaczników w odpowiednich miejscach oraz dostarczenie użytkownikom informacji o ich umiejscowieniu i kolejności, w jakiej powinny zostać zeskanowane.

W tym punkcie warto również zaznaczyć, że przed niektórymi działaniami użytkowników lub osób trzecich nie jesteśmy się w żaden sposób zabezpieczyć. Najsłabszym ogniwem systemu pod tym względem są bez wątpienia znaczniki NFC. Są one bardzo prostymi urządzeniami i nie posiadają praktycznie żadnych zabezpieczeń. Dzięki temu ktoś mógłby celowo wykasować lub zmienić dane zawarte w tagu, zmieniając przy tym „przerywając” przy tym łańcuch danych zbieranych przez użytkowników i skutecznie uniemożliwiając końcową weryfikacje ich poprawności. użytkownikom zgromadzenie odpowiednich danych i całkowicie uniemożliwić ich weryfikację. Ktoś mógłby taki znacznik również zniszczyć fizycznie lub zmienić jego położenie, przez co w aplikacjach części użytkowników nie zostałaby wygenerowana odpowiednia ilość bloków łańcucha danych. Jednak w tym przypadku można podjąć kroki mające na celu zniwelowanie problemu brakujących danych, pod warunkiem, że użytkownicy poinformowaliby o tym administratora. Odpowiedni algorytm znajdować będzie się w dalszej części rozdziału.



Rysunek 4. Przykładowe wiadomości wygenerowane przez funkcję tworzącą bloki danych

4.2 Zbieranie i zapisywanie danych w znacznikach przez użytkowników systemu



Rysunek 5. Schemat algorytmu pobierania i zapisywania danych przez użytkowników w trakcie wyścigu.

Gdy zainicjowane tagi zostaną już rozmieszczone na trasie, przechodzimy do drugiego etapu. Jest nim zasadnicza faza wyścigu (lub obchodu strażników), polegająca na pokonywaniu wyznaczonej trasy i skanowaniu odpowiednich znaczników przez użytkowników. Podczas

nawiązania połączenia pomiędzy urządzeniem użytkownika a tagiem NFC następuje złożona wymiana informacji. Zaczyna się ona od pobrania z tagu zaszyfrowanych danych. Następnie dane te trafiają do funkcji przetwarzającej, której zadaniem jest wygenerowanie nowej wiadomości i wygenerowanie nowego bloku, w którym wiadomość ta zostanie zapisana. Sam blok zostanie umieszczony na końcu łańcucha danych i będzie reprezentować odwiedzony przez użytkownika punkt kontrolny. Ostatnim etapem wymiany informacji ze znacznikiem jest wymazanie z niego starej wiadomości i umieszczenie w nim tej, którą aplikacja wygenerowała i umieściła w swoim łańcuchu danych. Wszystkie te operacje są wykonywane podczas jednego połączenia z tagiem tj. jednego przyłożenia telefonu. Jak już wspominałem we wcześniejszym rozdziale, całość trwa nie dłużej niż 0,1 sekundy i jest sygnalizowane przez wibrację urządzenia oraz wyświetlenie komunikatu o udanym połączeniu.

4.3 Weryfikacja danych dostarczonych przez użytkowników

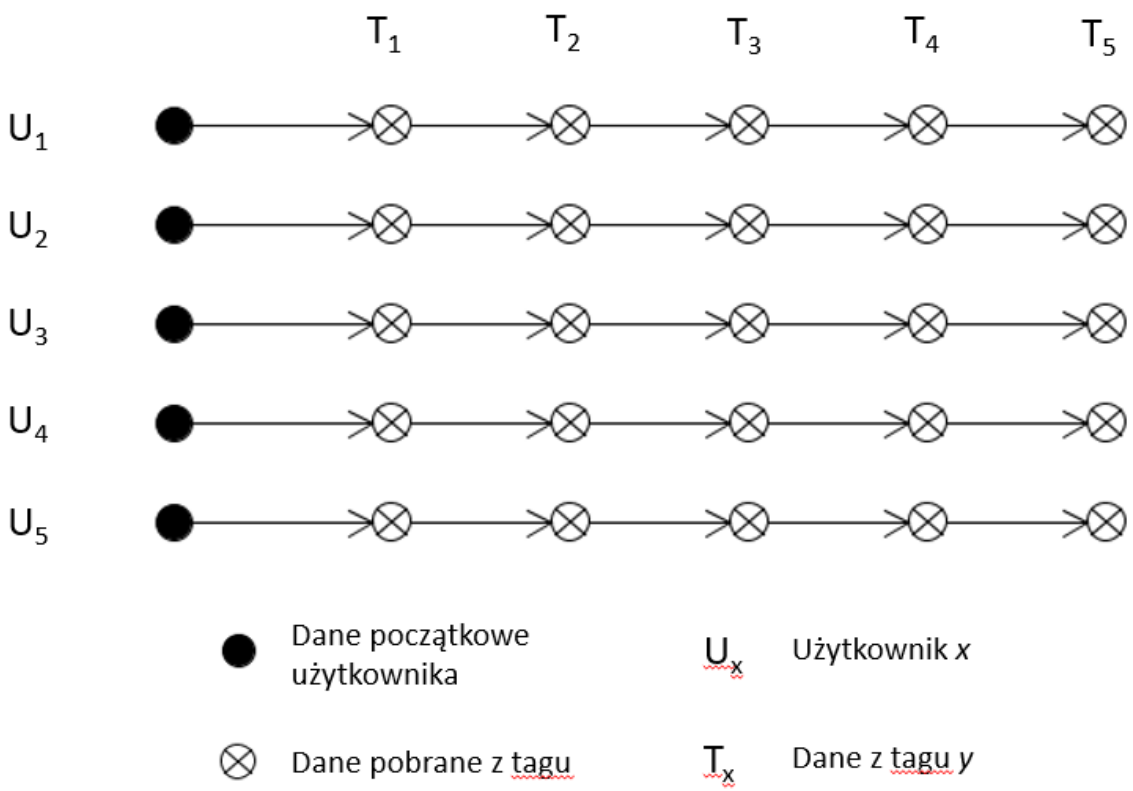
Etap ten rozpoczyna się od pobrania zebranych danych od wszystkich użytkowników. Na wstępie watro tu zaznaczyć, że nieobecność któregoś z zawodników na mecie wyścigu uniemożliwi dokładną weryfikację danych. W takim przypadku funkcja weryfikująca nie zadziała poprawnie i poinformuje administratora o nieciągłości danych. Dokładna interpretacja będzie już wtedy leżała tylko i wyłącznie po stronie administratora.

Właściwa weryfikacja rozpoczyna się poprzez utworzenie tablicy z danymi użytkowników oraz oznaczenia ewentualnych braków danych. Niekompletne łańcuchy danych będą oznaczone jako podejrzane i będą analizowane w nieco inny sposób. Dodatkowo należy utworzyć tablicę z informacją, czy dany użytkownik został już uznany za podejrzanego. Pozwoli to na ewentualne pomijanie zebranych przez niego danych (niepotrzebne rozpatrywanie kolejnych nieciągłości w łańcuchach).

4.4 Działanie algorytmu weryfikującego na różnych zbiorach danych dostarczonych przez użytkowników

Zależnie od dostarczonych łańcuchów danych, algorytm będzie musiał dostosować się do ewentualnych braków danych, wykrytych nieciągłości oraz naruszeń integralności. W przypadku wykrycia nieprawidłowości nie będziemy od razu oznaczać użytkownika jako podejrzanego, ponieważ nieprawidłowości w jego danych nie koniecznie muszą wynikać z jego złych intencji. Ze względu na przejrzystość działania opisywanego algorytmu każdy przypadek dostarczonych danych zostanie rozpatrzony osobno.

4.4.1 Dane kompletne bez naruszenia integralności



Rysunek 6. Przykład prawidłowych danych dostarczonych administratorowi przez użytkowników.

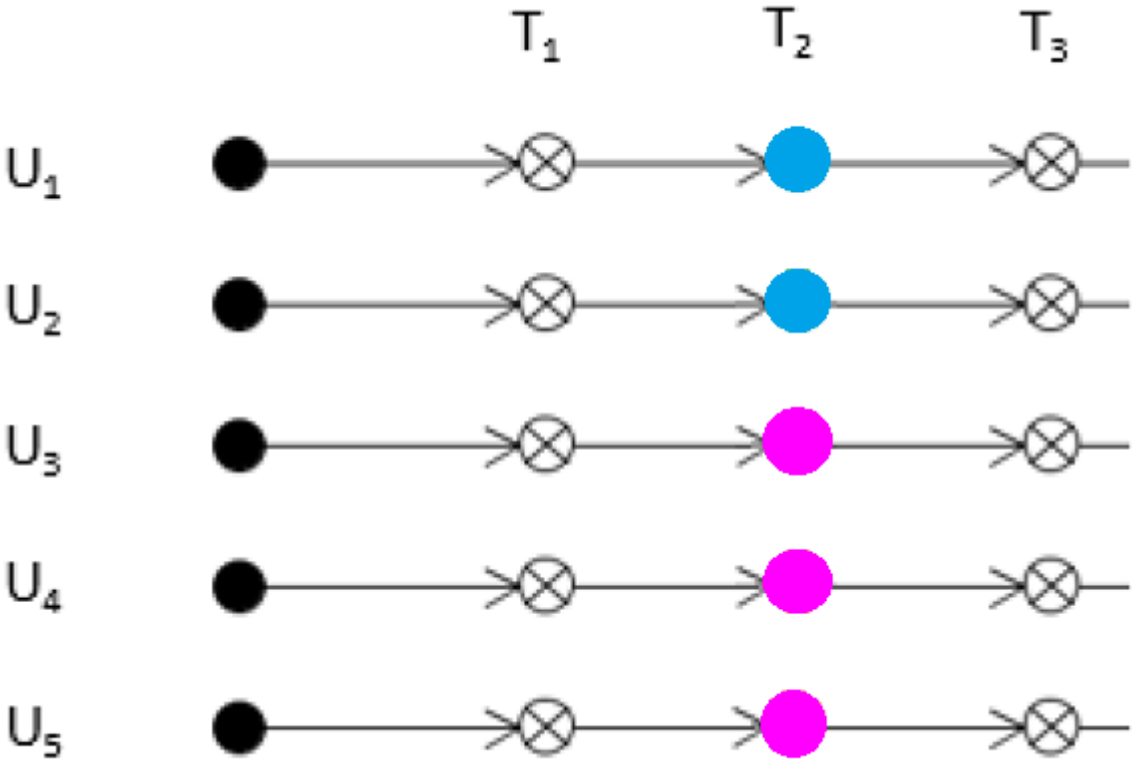
Jest to najprostszy możliwy schemat danych zebranych od użytkowników przez administratora. Według niego, każdy z uczestników dostarczy łańcuchy danych o prawidłowej długości oraz żaden z nich nie dopuści się celowego naruszenia integralności danych.

Proces weryfikacji będzie w tym przypadku odbywał się w następujący sposób:

1. Pobieramy wartość początkową dla tagu T_x.
2. Pobieramy dane z poprzedniego tagu T_{x-1} zeskanowanego przez użytkownika U_x (dla tagu T_1 będzie to blok początkowy indywidualny dla każdego z użytkowników).
3. Konwertujemy pozyskane tak dane za pomocą funkcji hashującej działającej w ten sam sposób jak u użytkowników, otrzymując wiadomość H_x.
4. Porównujemy wiadomość H_x z zapisaną w łańcuchu użytkownika dla tagu T_x. Jeśli są różne, wracamy do punktu nr 2 pobierając dane od kolejnego użytkownika U_{x+1}.
5. Jeśli dane się pokrywają, oznaczamy użytkownika jako sprawdzonego. Jeśli istnieją jeszcze nieoznaczeni użytkownicy, zebranymi przez aktualnie rozpatrywanego użytkownika danymi zastępujemy te, których do porównywania używaliśmy wcześniej (na początku była to wartość z T_x) i przechodzimy do punktu nr 2.
6. Jeśli dla tagu T_x wszyscy użytkownicy zostali oznaczeni jako sprawdzeni i nie był to ostatni tag, przechodzimy do punktu nr 1.
7. Jeśli był to ostatni tag, kończymy działanie algorytmu i wyświetlamy wyniki na ekranie urządzenia administratora.

4.4.2 Dane kompletne z naruszeniem integralności

W przypadku wystąpienia naruszenia integralności danych przy skanowaniu konkretnego tagu, algorytm nie będzie w stanie stworzyć pojedynczej grupy oznaczonych użytkowników, ponieważ w pewnym miejscu algorytm nie będzie w stanie wyszukać kolejnego użytkownika z pasującymi danymi. W takiej sytuacji algorytm podejmie próbę połączenia nie oznaczonych jeszcze użytkowników w kolejną grupę, lub nawet kilka grup w przypadku wielu nieprawidłowości. Każdy uczestnik, którego dane będą znajdować się na początku każdej kolejnej grupy, będzie oznaczany jako podejrzany, a ostateczna interpretacja będzie leżała po stronie administratora.



Rysunek 7. Dane z wykrytym brakiem integralności dla drugiego tagu. Na kolory niebieski oraz różowy pokolorowano dwie grupy powstałe w wyniku działania algorytmu.

Przypadek na załączonym obrazku można przedstawić następującym algorytmem:

1. Pobieramy wartość początkową dla tagu T_x.
2. Pobieramy dane z poprzedniego tagu zeskanowanego przez użytkownika U_x (dla tagu T_1 będzie to blok początkowy indywidualny dla każdego z użytkowników).
3. Konwertujemy pozyskane tak dane za pomocą funkcji hashującej działającej w ten sam sposób jak u użytkowników.
4. Porównujemy otrzymaną wiadomość z tą zapisaną w łańcuchu użytkownika dla odpowiedniego tagu. Jeśli są różne, wracamy do punktu nr 2 pobierając dane od kolejnego użytkownika.
5. Jeśli dane się pokrywają, oznaczamy użytkownika jako sprawdzonego i dodajemy do odpowiedniej grupy. Zebranymi przez niego danymi zastępujemy te, których do

porównywania używaliśmy wcześniej (na początku była to wartość z T_x) i przechodzimy do punktu nr 2.
6. Jeśli dla danego tagu wszyscy użytkownicy zostali oznaczeni jako sprawdzeni i nie był to ostatni tag, przechodzimy do punktu nr 1.
7. Jeśli był to ostatni tag, kończymy działanie algorytmu i wyświetlamy wyniki na ekranie urządzenia administratora. W przypadku braku spełnienia jakiegokolwiek z warunków w punkcie 6 i 7, inkrementujemy oznaczenie grupy.
8. Pobieramy dane zapisane przez kolejnego użytkownika (np. pierwszy nieoznaczony grupą). Będą one stanowiły punkt wyjściowy tak jak wcześniej początkowe dane z tagu T_x.
9. Pobieramy dane z poprzedniego tagu zeskanowanego przez następnego niedopasowanego użytkownika.
10. Konwertujemy pozyskane tak dane za pomocą funkcji hashującej.
11. Porównujemy otrzymaną wiadomość z tą zapisaną w łańcuchu użytkownika dla odpowiedniego tagu. Jeśli są różne, próbujemy umieścić dane nie na końcu, lecz na początku łańcucha. W tym celu pobieramy dane archiwalne dla pierwszego użytkownika w grupie i konwertujemy je razem z danymi dla aktualnego tagu rozpatrywanego uczestnika. Jeśli się zgadzają, „doczepiamy” blok na początku łańcucha i zmieniamy użytkownika oznaczonego jako początkowego. Jeśli istnieją jeszcze nierozpatrzeni użytkownicy, przechodzimy do punktu nr 8.
12. W przypadku braku możliwości dokonania kolejnych dopasowań do danej grupy, oznaczamy użytkownika, którego dane znajdują się na początku nowego łańcucha jako podejrzanego i przechodzimy do punktu nr 7.
13. Jeśli dla danego tagu zostali rozpatrzeni wszyscy użytkownicy – przechodzimy do punktu nr 1.
14. Jeśli był to ostatni tag – kończymy działanie algorytmu.

4.4.3 Dane niekompletne

W przypadku stwierdzenia braku odpowiedniej ilości danych u jednego z użytkowników będziemy musieli określić, czy brak ten jest spowodowany próbą naruszenia integralności czy też błędem w zapisie informacji przez samą aplikację. Algorytm do kroku nr 10 będzie działał bardzo podobnie jak algorytm przedstawiony w punkcie 4.4.2 i nie ma potrzeby na ponowne przedstawianie tych punktów. Algorytm zmieni się dopiero na koniec, przy próbie odtworzenia brakujących danych.

Rysunek 8. Łańcuchy danych z niekompletnymi danymi dla użytkownika numer 3. Na kolory niebieski oraz różowy pokolorowano dwie grupy powstałe w wyniku działania algorytmu

Algorytm weryfikacji w przypadku niekompletnych danych:

1. Punkty 1-11 – identycznie jak w algorytmie nr 2.
12. W przypadku braku możliwości dokonania kolejnych dopasowań do danej grupy, oznaczamy użytkownika, którego dane znajdują się na początku nowego łańcucha jako podejrzanego i przechodzimy do punktu nr 7.
13. Jeśli dla danego tagu zostali rozpatrzeni wszyscy użytkownicy – przechodzimy do punktu nr 1.
14. Jeśli jakiś użytkownik (U_N) nie został oznaczony jako rozpatrzony, a dla danego tagu brakuje już kolejnych danych, przechodzimy do punktu nr 15.
15. Pobieramy dane dostarczone przez ostatniego użytkownika w kolejnej grupie G_x. Dodajemy do nich w odpowiedni sposób dane z poprzedniego tagu użytkownika U_N i konwertujemy za pomocą funkcji hashującej.
16. Za pomocą nowo wygenerowanej wiadomości, podejmujemy próbę połączenia grup użytkowników. W tym celu do nowo wygenerowanych danych dodajemy dane z poprzedniego tagu użytkownika, który znajduje się na początku łańcucha danych kolejnej grupy G_y i je konwertujemy. Jeśli tak otrzymana wiadomość będzie identyczna z danymi tego użytkownika dla aktualnie rozpatrywanego tagu, oznacza to, że użytkownik U_N nie próbował celowo naruszyć integralności danych i możemy wrócić do punktu nr 1, jednocześnie uzupełniając tablicę danych o nowo wygenerowany blok dla użytkownika U_N.
17. Jeśli połączenie się nie uda i mamy jeszcze nie rozpatrzone grupy, powtarzamy punkt nr 16 dla kolejnej grupy.
18. Jeśli rozpatrzymy wszystkie pozostałe grupy dla G_x, nie znajdziemy odpowiednich powiązań oraz mamy jeszcze pary grup, których nie próbowaliśmy połączyć, przechodzimy do punktu nr 3.
19. Jeśli rozpatrzymy już wszystkie grupy „każda z każdą” i nie znajdziemy odpowiedniego dopasowania, oznaczamy rozpatrywanego użytkownika jako podejrzanego i przechodzimy do punktu nr 6.

Podstawowa kwestią, jaką warto poruszyć w trakcie rozważania tego algorytmu, jest dopuszczalna ilość brakujących danych. O ile brak danych dla pojedynczego użytkownika w rozpatrywanym tagu nigdy nie będzie problemem, ponieważ w takim wypadku wyżej podany algorytm zawsze zadziała, tak problemy mogą pojawić się przy większej ilości luk w danych. Pierwszym problem jest fakt, że algorytm ten jest bardzo złożony i w przypadku dużej ilości podgrup i dużej ilości użytkowników z brakującymi danymi, jego wykonanie może zająć stosunkowo dużo czasu.

Drugim, o wiele poważniejszym problemem jest wymóg, by każda faktyczna podgrupa, jaką możemy utworzyć w zestawie danych, miała co najmniej jednego reprezentanta, do którego będziemy mogli dopasowywać dane. Oznacza to mniej więcej tyle, że w przypadku np. trzech przerw w łańcuchu danych dla wybranego tagu, wywołanych działaniem osób trzecich (przypadek podobny jak w podpunkcie 4.4.2), użytkownicy powinni zostać podzieleni na cztery podgrupy (ilość przerw powiększona o jeden), każda z co najmniej jednym reprezentantem. Jeśli nie uda się tego zrobić (np. dla danej podgrupy żaden z użytkowników nie dostarczył danych), nie będzie możliwe wykonanie pełnego podalgorytmu łączenia ze sobą grup użytkowników za pomocą generowanych danych pomocniczych (ponieważ po prostu nie będziemy znali wszystkich podgrup). W takim wypadku jedyną możliwym działaniem będzie oznaczenie wszystkich użytkowników, których nie udało się ostatecznie przydzielić do grup nawet za pomocą odtwarzania danych, jako podejrzanych i pozostawić ostateczną interpretację administratorowi.

5. Implementacja

W niniejszym rozdziale zostaną przedstawione najważniejsze aspekty implementacji aplikacji, z podziałem na konkretne klasy i funkcjonalności. Dla wybranych klas zostaną również przedstawione ich reprezentacje graficzne oraz/lub przykładowe wyniki działania owych klas.

5.1 Block chain

Opis implementacji aplikacji rozpocznę od opisu klasy odpowiedzialnej z tworzenie bloków dla łańcuchów danych, ponieważ większość z funkcjonalności aplikacji korzysta właśnie z niej. Zaimplementowana klasa jest uproszczoną wersją standardowej wersji block chain, ponieważ nie zawiera między innymi stempla czasowego czy tak zwanej zmiennej „nonce”. Standardowy blok w aplikacji składa się jedynie z danych pobranych dla nowo tworzonego bloku, skrótu wiadomości wygenerowanego dla poprzedniego bloku w łańcuchu oraz z wygenerowanego za ich pomocą skrótu wiadomości dla aktualnie tworzonego bloku.

```
public class blockChain {
    private String hash;
    private String previousHash = null;
    private String data;

    public blockChain(String data, String previousHash) {
        this.data = data;
        this.previousHash = previousHash;
        this.hash = calculateBlockHash();
    }
```

Rysunek 9. Elementy klasy blockChain wraz z konstruktorem.

Warty pokazania jest również sam sposób tworzenia skrótu wiadomości, w tym przypadku ma on miejsce w funkcji calculateBlockHash(). Jest to uproszczona wersja standardowej implementacji takiej funkcji. Na początku funkcji z dostarczonych danych i wiadomości z poprzedniego bloku w łańcuchu tworzymy jedną wiadomość, z której zostanie utworzony nowy hash dla aktualnie tworzonego bloku. Za pomocą klasy MessageDigest i dostarczanych przez nią narzędzi możemy w prosty sposób dokonać transformacji wiadomości za pomocą wybranego algorytmu tworzenia skrótu wiadomości (w tym wypadku SHA-256). Na koniec otrzymany hash jest konwertowany do systemu szesnastkowego w celu zwiększenia przejrzystości danych i ich skrócenia.

```java
public String calculateBlockHash() {
    String dataToHash = previousHash + data;
    MessageDigest digest = null;
    byte[] hash = null;
    try {
        digest = MessageDigest.getInstance("SHA-256");//hash
        hash = digest.digest(dataToHash.getBytes(UTF_8));
    } catch(Exception exc) {
        throw new RuntimeException(exc);
    }
    StringBuffer buffor = new StringBuffer(); //hex buffor
    for (byte tempByte : hash) {
        buffor.append(String.format("%02x", tempByte));//convert to hex
    }
    return buffor.toString();
}
```

Rysunek 10. Funkcja wykonująca operację pozyskania skrótu wiadomości na podstawie otrzymanych danych.

5.2 Odczyt i zapis wiadomości w tagu

5.2.1 Przygotowanie aplikacji oraz tagu do nawiązania połączenia

Odczyt oraz zapis informacji w znaczniku NFC jest kluczową funkcjonalnością całej aplikacji. Bez tej funkcjonalności użytkownicy nie byliby zdolni do zeskanowania czy zapisania jakichkolwiek danych z tagów. Klasa odpowiedzialna za tą funkcję jest wywoływana automatycznie po nawiązaniu przez urządzenie połączenia z tagiem NFC. Dzieje się tak dzięki dwóm dodatkowym zapisom w pliku AndroidManifest.xml[7] (rysunek nr 11). Pierwszy z nich zezwala aplikacji na korzystanie z modułu NFC, natomiast drugi odpowiedzialny jest za wywołanie wskazanej klasy po wykryciu tagu (w tym przypadku jest to NFCManager.java).

```xml
<uses-permission android:name="android.permission.NFC" />
```

```xml
<activity android:name=".NFCManager">
    <intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

Rysunek 11. Wpis umożliwiający aplikacji korzystanie z modułu NFC

[7] Plik opisujący podstawowe informacje o aplikacji takie jak wymagania sprzętowe czy wymagane przez nią uprawnienia.

Gdy już zostanie nawiązane połączenie z tagiem NFC, możemy przejść do właściwej klasy obsługującej wymianę danych z tagami, a mianowicie NFCManager.java. Na początku, klasa ma za zadanie utworzyć reprezentację tagu, której będziemy używać do dalszej komunikacji z tagiem (w tym przypadku tylko zapisywanie danych). Następnie wymuszamy pierwszeństwo dla aplikacji w obsłudze znacznika z którym nawiązaliśmy połączenie, tak aby w trakcie wymiany informacji żadna z aplikacji nie przeszkodziła w tym procesie. W przypadku posiadania na urządzeniu kilku aplikacji obsługujących łączność NFC, w trakcie połączenia na ekranie urządzenia może pojawić się komunikat o preferowanej aplikacji do wykonania tej czynności. Mogłoby to prowadzić do wielu błędów w działaniu samej klasy, jak i zwiększenia ryzyka błędów ludzkich takich jak zbyt szybkie odsunięcie urządzenia od znacznika NFC i w rezultacie możliwą utratę danych. Takie zabezpieczenie jest standardem w aplikacjach korzystających z NFC, dlatego zostało ono w dużej mierze zaczerpnięte z dokumentacji systemu Android.

```
protected void onResume() {
    super.onResume();
    Intent nfcIntent = getIntent();
    Tag tag = (Tag) nfcIntent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
    if(tag != null) { //pobranie zawartosci tagu, przygotowanie danych do odczytu
        nfcIntent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, nfcIntent,  flags: 0);
        IntentFilter[] intentFiltersArray = new IntentFilter[]{};
        String[][] techList = new String[][]{
                {android.nfc.tech.Ndef.class.getName()},
                {android.nfc.tech.NdefFormatable.class.getName()}
        };
        NfcAdapter nfcAdpt = NfcAdapter.getDefaultAdapter(this);
        nfcAdpt.enableForegroundDispatch( activity: this, pendingIntent, intentFiltersArray, techList);
```

Rysunek 12. Implementacja przygotowania tagu do dalszych operacji.

5.2.2 Inicjalizacja tagów NFC

```
    System.out.print("TRYB 2 ");
    pom++;
    blockChain newBlock = new blockChain(pom.toString());
    NdefMessage ndef = createTextMessage(newBlock.getHash());
    writeTag(tag, ndef);
    System.out.print("Dane zapisane: ");
    System.out.println(newBlock.getHash());
}
Toast.makeText(getApplicationContext(),  text: "Zeskanowano tag!", Toast.LENGTH_SHORT).show();
super.finish();
```

Rysunek 13. Proces inicjalizacji danych po przechwyceniu znacznika przez aplikację.

Inicjalizacja danych w tagach NFC jest procesem rozpoczynającym przeprowadzenie wyścigu lub kontrolowanie pracy strażnika. Na tym etapie nie jest jeszcze konieczne tworzenie łańcucha danych, a jedynie pojedynczych bloków, które ułatwią stworzenie wiadomości zaszyfrowanych i obsługiwanych przez aplikację. W widocznej na powyższym obrazie funkcji widzimy prócz stworzenia samego bloku oraz wypisania danych, ułatwiających śledzenie

działań aplikacji w środowisku Android Studio, przekierowania do dwóch funkcji: **createTextMessage** oraz **writeTag**, które zostaną omówione w dalszej części podrozdziału. Warta uwagi jest również zmienna **pom**, która służy jako ziarno do generowania kolejnych wiadomości umieszczanych w inicjalizowanych tagach. Zmienna ta jest wyznaczana przez administratora na początku inicjalizacji, a następnie modyfikowana dla każdego kolejnego zapisu (w tym przypadku prosta inkrementacja).

Funkcja **createTextMessage** ma za zadanie przekonwertować wiadomość, którą chcemy umieścić w tagu do formatu NDEF. W poprzednich rozdziałach wspominałem, że jest to standardowy format danych przechowywanych przez znaczniki. Takie przekształcenie jest operacją konieczną, ponieważ adapter NFC nie przyjmuje wartości ze standardowych bibliotek Java takich jak string. Przy tworzeniu rekordu, stałe TNF_WELL_KNOWN oraz RTD_TEXT przekazują informację o pożądanym formacie danych, w jakim chcemy zapisać wiadomość w znaczniku. W tym przypadku będzie to zwykły plain/text. Od wyboru typu danych zależy również to, która z aplikacji zainstalowanych na urządzeniu „przechwyci" tag po nawiązaniu z nim połączenia, tak więc jego prawidłowe sformułowanie jest kluczowe dla działania aplikacji (pożądany przez aplikację typ danych określany jest w pliku AndroidManifest.xml). Po przygotowaniu wiadomości w ten sposób, wywoływana jest funkcja writeTag, która stanowi ostatnią fazę wymiany informacji z tagiem NFC. Ma ona za zadanie ponownie nawiązać kontakt z tagiem (**Ndef::connect()**), zapisać wiadomość w tagu (**Ndef::writeNdefMessage()**) oraz ostatecznie zamknąć połączenie ze znacznikiem (**Ndef::close()**). Podane funkcje pochodzą z biblioteki android.nfc.

```
byte[] lang = Locale.getDefault().getLanguage().getBytes( charsetName: "UTF-8");
byte[] text = content.getBytes( charsetName: "UTF-8");
int langSize = lang.length;
int textLength = text.length;
ByteArrayOutputStream payload = new ByteArrayOutputStream( size: 1 + langSize + textLength);
payload.write(text,  off: 0, textLength);
NdefRecord record = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
        NdefRecord.RTD_TEXT, new byte[0],
        payload.toByteArray());
return new NdefMessage(new NdefRecord[]{record});
```

Rysunek 14. Funkcja createNdefMessage mająca za zadania przygotować wiadomość do umieszczenia w tagu.

5.2.3 Wymiana danych z tagami podczas wyścigu

Odczyt oraz zapis danych do znaczników jest bardzo ważną funkcjonalnością z punktu widzenia całej aplikacji. Jej wywołanie może nastąpić jedynie w przypadku, gdy aplikacja została ustawiona w tryb pracy wyścigu czy obchodu strażnika. W momencie nawiązania połączenia z tagiem zawierającym konkretny typ danych przez aplikację, następuje przygotowanie instancji tagu i adaptera do wymiany informacji (proces opisany w podpunkcie 5.2.1).

Gdy znacznik jest już przygotowany do komunikacji, możemy pobrać z niego rekord danych. Odbywa się to w kilku krokach, widocznych na rysunku nr 15. Pierwszym z nich jest pobranie

wiadomości przekazanych przez intencję (Intent[8]), odnoszącej się do nawiązania połączenia z tagiem NFC. Następnie musimy przekonwertować rekordy takiej wiadomości na format danych NdefMessage. Z racji tego, że tagi przechowują tylko jeden rekord zawierający skrót wiadomości, to wystarczy pobrać tylko rekord o indeksie 0.

```
nfcAdpt.enableForegroundDispatch( activity: this, pendingIntent, intentFiltersArray, techList);
Parcelable[] rawMessages = nfcIntent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
NdefMessage[] messages = new NdefMessage[rawMessages.length];
messages[0] = (NdefMessage) rawMessages[0];
NdefRecord recordFromTag = messages[0].getRecords()[0];
String dataFromTag = new String(recordFromTag.getPayload(), StandardCharsets.UTF_8);
```

Rysunek 15. Proces pobrania rekordu o indeksie 0 z adaptera NFC.

Dla pobranych w ten sposób danych należy następnie wygenerować nowy blok łańcucha danych, który zostanie umieszczony na jego końcu. Oprócz pobranej wiadomości konstruktorowi bloku należy również przekazać skrót wiadomości uzyskanej dla poprzedniego bloku w łańcuchu. W przypadku skanowania pierwszego tagu, dla którego nie możemy uzyskać wartości skrótu wiadomości poprzedniego bloku, na podstawie pobranych danych tworzymy losową wiadomość, która będzie służyła jako indywidualny klucz dla użytkownika. Proces przygotowania wiadomości do wysłania i samo jej wysłanie jest taki sam jak zostało to opisane w podpunkcie 5.2.2.

5.2.4. Weryfikacja danych dostarczonych przez użytkowników.

Sprawdzenie poprawności danych jest najważniejszą i najbardziej złożoną funkcją aplikacji. Wynikiem jej działania powinny być logi wyświetlane na urządzeniu administratora, informujące o wszelkich naruszeniach integralności. Dodatkowo będą one zawierały informację o kolejności skanowania konkretnego tagu przez użytkowników, co ma ułatwić administratorowi arbitralne rozwiązywanie kwestii spornych. W tym podpunkcie zostanie opisana implementacja algorytmu dla danych bez naruszeń integralności.

Weryfikacja danych rozpoczyna się od przekazania do urządzenia administratora łańcuchów danych utworzonych w urządzeniach użytkowników podczas trwania wyścigu. Podstawową metodą przekazania tych danych jest nawiązanie połączenia NFC pomiędzy urządzeniami. Niestety od wersji systemu Android 10, funkcja Android Beam, która obsługuje wymianę NFC pomiędzy dwoma urządzeniami, została usunięta i zastąpiona innymi technologiami. W takim przypadku należy rozpatrzeć inne sposoby na przekazanie danych administratorowi. Najprostszym sposobem na wymianę danych jest użycie jednego z tagów jako pośrednika do wymiany danych pomiędzy urządzeniami.

Po zebraniu danych i wpisaniu ich do macierzy przechowującej wszystkie łańcuchy danych, funkcja przechodzi do wykonania algorytmu. Dla każdego z tagów funkcja pobiera wartość początkową w tagu zainicjalizowaną przez administratora (pętla **for** na rysunku nr 16), a następnie próbuje odnaleźć rozpatrywaną wartość w odpowiednim bloku łańcucha danych

[8] Intencja to mechanizm systemu Android umożliwiający obsługę rozkazów użytkownika aplikacji, uruchamianie konkretnych aplikacji i klas oraz umożliwiający komunikację między nimi.

kolejnych użytkowników(pętla **while**). Jeśli się to uda, aplikacja oznacza wybranego użytkownika jako sprawdzonego dla danego tagu, a następnie pobiera wartość hash w jego bloku, czyli wartość którą to on zapisał w tagu i powtarzamy proces szukania tej wiadomości w łańcuchach danych. W międzyczasie zapisujemy kolejność skanowania konkretnego tagu przez użytkowników. Czynność tę wykonujemy dopóki wszyscy użytkownicy nie zostaną oznaczeni jako sprawdzeni. W przypadku gdy się to nie uda, aplikacja wyświetli na ekranie informację o nieprawidłowościach w danych.

```
for(int i = 1; i < tagsInitiateValues.size(); i++){ //przejscie po kolejnych tagach
    String actualTagValue = tagsInitiateValues.get(i);
    Integer [] userCheck;
    userCheck = new Integer[users];
    List<String> order = new ArrayList<>();
    int checked = 0;
    Integer k = 0;
    int group = 1; // grupy uwierzytelniania, poterzebne w przypadku braku danych
    while(k < users) { //pierwsze przejscie po tagach, zaczynamy od tagu poczatkowego, nie musimy sprawdzac
        if (matrix.get(k).get(i).getData() == actualTagValue) {
            userCheck[k] = group;
            order.add(k.toString()); //zapis kolejnosci
            actualTagValue = matrix.get(k).get(i).getHash();
            k = 0;
            checked++;
        } else
            k++;
    }
    usersOrder.add(order);
}
```

Rysunek 16. Proces badania integralności i wyznaczania kolejności dla kolejnych tagów.

Po zakończeniu działania funkcji na ekranie administratora zostaną wypisane informacje o przebiegu weryfikacji danych. Jest to ostatni etap przeprowadzania wyścigu za pomocą aplikacji.

6. Podsumowanie

Aplikacja spełnia najważniejsze wymagania postawione na etapie projektowania. Przede wszystkim jest w stanie działać bez użycia klasycznych technologii pozwalających badać lokalizację urządzenia takich jak GPS. Cały proces gromadzenia danych odbywa się jedynie za pośrednictwem technologii komunikacji NFC. Z odczytanych z tagów wiadomości program potrafi utworzyć łańcuchy danych, które umożliwią dokładne sprawdzenie integralności zebranych danych. Aplikacja jest w stanie działać w kilku różnych trybach pozwalających na obsługę poszczególnych etapów realizacji wyścigu terenowego bądź obchodu strażnika.

Największym wyzwaniem implementacyjnym w omawianej aplikacji była bez wątpienia funkcja badająca integralność danych. Niestety nie udało się zaimplementować jej w całości. Dla prawidłowych danych dostarczonych przez użytkowników potrafi ona odtworzyć cały przebieg wyścigu, wliczając w to ustalenie kolejności skanowania konkretnego znacznika przez zawodników. W przypadku wykrycia braku integralności danych, aplikacja poinformuje administratora o wykrytych nieprawidłowościach, lecz nie będzie potrafiła określić, czy wynikają one z błędu zapisu danych przez aplikację czy z celowej próby naruszenia integralności przez użytkownika (algorytmy opisany w punktach 4.4.2 oraz 4.2.3).

Problemy pojawiły się też ze strony wykorzystywanych technologii. Najważniejszym z nich jest wycofanie funkcji przesyłania danych peer-to-peer za pomocą NFC w najnowszych wersjach systemu Android i zastąpienie ich innymi technologiami. Powoduje to brak możliwości dostarczenia łańcucha danych administratorowi przez posiadaczy urządzeń z systemem Android 10 lub nowszym. Wymusiło to zmianę początkowej koncepcji i wykorzystanie pośrednika wymiany danych pomiędzy urządzeniami w postaci dodatkowego znacznika NFC, który może być zeskanowany przez każde urządzenie.

Jeśli chodzi o dalszy rozwój aplikacji, na chwilę obecną priorytetem jest implementacja obsługi badania integralności dla wszystkich możliwych przypadków naruszenia integralności opisanych w podpunkcie 4.4. Udoskonalenia wymaga również interfejs graficzny aplikacji, który na ten moment jest bardzo uproszczony.

Bibliografia

1. J.Bloch, *Effective Java – Third Edition*. Addison-Wesley Professional, 2018, ISBN: 9780134685991
2. https://nfc-forum.org/our-work/specification-releases/specifications/nfc-forum-technical-specifications/
3. https://developer.android.com/guide/topics/connectivity/nfc
4. https://developer.android.com/training/beam-files
5. https://developer.android.com/reference/android/nfc/NfcAdapter
6. https://developer.android.com/topic/architecture
7. https://developer.android.com/guide/topics/ui
8. https://developer.android.com/reference/android/nfc/NdefRecord
9. https://developer.ibm.com/patterns/create-an-android-app-with-blockchain-integration/

Spis rysunków

Spis tabel