POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Piotr Kuszewski
Nr albumu 16083

Biblioteka decyzyjna wspomagająca dystrybucję zadań wymagających obliczeniowo

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, czerwiec 2018

Streszczenie

Coraz lepsza wydajność urządzeń mobilnych oraz coraz większa dostępność aplikacji w sklepie Play sprawia, że aplikacje mobilne uruchamiane na smartfonach często zastępują klasyczne aplikacje komputerowe w zastosowaniach domowych. Programy te często wykonują skomplikowane operacje obliczeniowe, które mogą trwać bardzo długo na urządzeniach niższej klasy lub na starszych modelach. Ponadto, wykonywanie operacji lokalnie na telefonie prowadzi do szybszego rozładowywania baterii i wzrostu niezadowolenia użytkowników. Z drugiej strony, operacja nie zawsze może być wykonywana na serwerze, na przykład, gdy smartfon nie ma aktywnego połączenia z siecią Internet. Użytkownicy oczekują bezproblemowego działania aplikacji pobranych ze sklepu Play na każdym urządzeniu i w każdych warunkach. Zaproponowana biblioteka decyzyjna stara się rozwiązać ten problem w sposób najbardziej odpowiadający użytkownikom końcowym aplikacji. Jednocześnie jest łatwa w użyciu i integracji nawet z istniejącymi aplikacjami.

Słowa kluczowe

Informatyka, Android, biblioteka decyzyjna

Podziękowania

Chciałbym podziękować Promotorowi dr inż. Mariuszowi Trzaska za pomoc w tworzeniu niniejszej pracy. Dziękuję także mgr inż. Piotrowi Zybertowi oraz mgr inż. Michałowi Walędziakowi za użyczenie urządzeń testowych oraz możliwość przeprowadzenia eksperymentu. Dziękuję też wszystkim respondentom ankiet, zarówno użytkownikom jak i programistom za poświęcony czas oraz wkład merytoryczny w rozwój niniejszej pracy.

Spis treści

1. Wstęp

Aplikacje mobilne są obecne niemal w każdym aspekcie życia ludzkiego. Dzięki nim można zdecydowanie uprościć i przyspieszyć wiele codziennych czynności, takich jak zakupy, bankowość czy komunikacja z innymi. Przenośnie urządzenia typu „smartphone" zdobyły niesamowitą popularność, gdyż są uniwersalne i łatwo dostępne. Producenci takich urządzeń różnicują swoją ofertę, by dotrzeć do wszystkich zainteresowanych. Powoduje to dużą różnorodność urządzeń pod względem ich wydajności, a co za tym idzie również ceny. Ponadto, dynamiczny rozwój elektroniki sprawia, że telefony, które kiedyś były liderami w dziedzinie wydajności, po zaledwie kilku miesiącach stają się co najwyżej przeciętne.

Same aplikacje, z których tak chętnie korzystają użytkownicy, również się zmieniają. Wymagania stawiane przed programistami rosną, tak samo jak rosną wyzwania związane ze sprzętem. Użytkownicy oczekują, że aplikacja będzie działała szybko, bezbłędnie i bezproblemowo, jeżeli tylko można ją zainstalować. System Android pod tym względem jest bardzo elastyczny dla użytkownika, gdyż nawet kilkuletnie urządzenia posiadają pełne wsparcie systemu operacyjnego oraz jego twórców. Oznacza to, że użytkownicy niechętnie wymieniają swoje urządzenia na nowe, jednocześnie oczekując nienagannej płynności uruchomionych aplikacji.

Użytkownicy coraz częściej rezygnują z konwencjonalnego komputera klasy PC na rzecz urządzeń mobilnych, do wykonywania codziennych czynności takich jak:

- zakupy,
- tworzenie prostych dokumentów czy arkuszy kalkulacyjnych,
- zarządzanie zdjęciami,
- przeglądanie stron Internetowych,
- operacje bankowe.

Niektórzy producenci telefonów z systemem Android wyposażają swoje produkty w dedykowane stacje dokujące, które można podłączyć do telewizora lub innego monitora. Po włożeniu urządzenia mobilnego w taką stację, zmienia się ono w pełnoprawny komputer stacjonarny z oprogramowaniem Android. Interfejs użytkownika dostosowany jest wtedy do obsługi przy pomocy zewnętrznych peryferii, takich jak myszka czy klawiatura. Użytkownik odnosi wtedy wrażenie, że obcuje z komputerem klasy PC i takie też stawia oczekiwania przed urządzeniem i aplikacjami.

Powyższe czynniki wpływają na fakt, że aplikacje mobilne muszą realizować coraz trudniejsze zadania. Niektóre z nich wymagają ogromnej mocy obliczeniowej. Są to takie zastosowania jak:

- edycja i tworzenie filmów,
- konwersja plików graficznych i audiowizualnych,
- tworzenie trójwymiarowych modeli,
- renderowanie obrazu i dźwięku,
- analiza preferencji użytkownika (uczenie maszynowe),
- aplikacje naukowe.

Większość tych zastosowań kojarzy się z profesjonalnym wykorzystaniem urządzeń elektronicznych i wydaje się, że nie powinny mieć miejsca na urządzeniach przenośnych. Są to jedynie pozory, większość tych przypadków jest wykorzystywanych obecnie w aplikacjach do użytku codziennego, np. w aplikacjach społecznościowych, programach do wspomagania aranżacji wnętrz, agregatorach treści czy silnikach wyszukujących.

Nie bez znaczenia jest też coraz większa dostępność Internetu na urządzeniach mobilnych. Operatorzy komórkowi systematycznie obniżają ceny dostępu do sieci oraz limity przesyłanych danych. Większość smartfonów jest stale podłączona do sieci Internet. Zasięg sieci komórkowych pozwala na stały dostęp, a prędkość transmisji nowoczesnych sieci 4G oraz LTE pozwala na szybką komunikację urządzeń mobilnych z zewnętrznymi serwisami.

Programista tworzący aplikację musi podjąć trudną decyzję o miejscu wykonania obliczeń. Do wyboru ma urządzenie użytkownika, czyli obliczenia wykonywane lokalnie bądź chmurę obliczeniową lub serwer, czyli obliczenia wykonywane zdalnie. Decyzja nie jest łatwa ze względu na przedstawione powyżej czynniki, czyli duże zróżnicowanie wydajności urządzeń czy warunków, w których uruchamiana jest aplikacja.

Warto też zwrócić uwagę na koszt finansowy obydwu rozwiązań. W przypadku operacji wykonanych na urządzeniu, znacząco spada obciążenia serwera. Obliczenia zostają wtedy rozproszone na procesory użytkowników, co powoduje zmniejszenie zapotrzebowania na moc obliczeniową samego serwera. Dzięki temu można zdecydować się na mniejszą ilość procesorów czy pamięci RAM po stronie serwera, zmniejszając tym samym koszt utrzymania infrastruktury. Różnica ta wzrasta wraz z rosnącą liczbą użytkowników systemu.

2. Cel pracy

Celem niniejszej pracy jest identyfikacja czynników wpływających na decyzję o miejscu wykonywania wymagających obliczeń i algorytmów. Ponadto, w trakcie tworzenia pracy zostały przeprowadzone badania socjologiczne, których celem było poznanie potrzeb użytkowników aplikacji oraz potrzeb programistów tworzących owe aplikacje. By lepiej zrozumieć istotę problemu, w ramach tworzenia pracy zostały przeprowadzone serie eksperymentów mające na celu identyfikację czynników mogących wpływać na decyzję o miejscu wykonania obliczeń.

2.1. Istota problemu

Podczas tworzenia pracy została dogłębnie zidentyfikowana istota problemu. Przy pomocy ankiety internetowej zostało przeprowadzone badanie mające na celu potwierdzenie, czy dany problem jest realny i dotyka zarówno użytkowników końcowych aplikacji jak i samych programistów.

Analiza wyników ankiety wykazała, że potrzeby i priorytety użytkowników końcowych są odmienne od tych stosowanych przez programistów. Dla użytkowników najważniejszy jest komfort korzystania z urządzenia, podczas gdy programiści główną wagę przywiązują do szybkości działania i jakości samego efektu działania aplikacji.

Z wyników ankiety można także wnioskować, że wielu programistów zupełnie nie jest świadomych, że podejmowane przez nich decyzje podczas tworzenia aplikacji w tak dużym stopniu rzutują na jej odbiór wśród użytkowników końcowych. Zwykle decyzja podejmowana jest arbitralnie, bez głębszej analizy jej konsekwencji. Wielu programistów przyznaje również, że brak automatycznego systemu decyzyjnego wynika z ograniczonych możliwości finansowych oraz czasowych podczas tworzenia projektu. Dla wielu z nich problemem jest również stworzenie i utrzymywanie zdalnego środowiska wykonawczego, szczególnie w przypadku małych aplikacji o krótkim czasie życia. Pomimo tego, że kilka osób przyznało, że wykorzystywało proste systemy decyzyjne bazujące na zestawie reguł, nikt nie próbował stworzyć rozwiązania uniwersalnego, generycznego, mającego zastosowanie w przyszłych projektach.

2.2. Efekt pracy

Efektem pracy jest zestaw reguł, który pozwala na podjęcie decyzji o miejscu wykonywania obliczeń w sposób optymalny dla użytkownika końcowego. Został on zawarty w strategiach decyzyjnych prototypu biblioteki przeznaczonej dla twórców aplikacji Android. Reguły decyzyjne biorą pod uwagę czynniki kluczowe dla użytkownika, takie jak:

- Stan naładowania baterii urządzenia,
- Średni oraz przewidywany czas wykonania operacji,
- Temperatura urządzenia,
- Obciążenie procesora,
- Ilość wymaganych do przesłania danych.

Prototyp biblioteki składa się z kilku podstawowych modułów, m.in.:

- procesor adnotacji,
- moduł podejmujący decyzję,
- menedżer strategii,
- moduł statystyczny,
- fasada bazy danych.

Dzięki zastosowaniu biblioteki programista oszczędza dużo czasu na pisaniu własnego rozwiązania automatyzującego podjęcie decyzji. Wielu programistów zupełnie rezygnuje z zastosowania jakiejkolwiek decyzyjności i podejmuje decyzję o miejscu wykonywania operacji w momencie pisania kodu, ignorując w większości warunki środowiskowe w jakich będzie wykonywana ich aplikacja.

Programista, który zastosuje bibliotekę w swoim kodzie, poprawi odbiór swojej aplikacji wśród użytkowników końcowych. Aplikacje korzystające z rozwiązania będą charakteryzowały się mniejszym zużyciem baterii, niższą temperaturą pracy urządzenia, zoptymalizowanym wykorzystaniem zasobów sieciowych i transferu danych.

2.3. Rozwiązanie przyjęte w pracy

W celu identyfikacji i potwierdzenia istoty problemu została przeprowadzona ankieta internetowa. Była ona skierowana do dwóch grup docelowych:

- użytkowników urządzeń z systemem Android,
- programistów aplikacji na system Android.

Odpowiedzi na pytania pozwoliły na głębsze poznanie problemów i identyfikację potrzeb obydwu tych grup użytkowników. Ankieta została przeprowadzona w języku polskim, za pośrednictwem serwisu surveymonkey.com [3]. Wybór serwisu motywowany był jego popularnością i przyjaznym interfejsem użytkownika, a także prostym tworzeniem ankiet. Wyniki ankiet zostały

zebrane do arkusza Excel w celu ich dalszej analizy. Na podstawie wyników zostały sporządzone wykresy, które w graficzny sposób obrazują odpowiedzi respondentów i pozwalają na łatwe wyciąganie wniosków.

W celu oceny przydatności różnych czynników mogących mieć wpływ na decyzję o miejscu wykonywania obliczeń została napisana prosta aplikacja mobilna na platformę Android wykonująca periodycznie wybrane operacje złożone obliczeniowo oraz zbierająca dane o aktualnym stanie urządzenia. Wykonywane operacje to:

1. Przycięcie i renderowanie filmu video w rozdzielczości Full HD,
2. Obliczanie ciągu Fibonacciego.

Aplikacja podczas pojedynczego eksperymentu wykonywała tylko jedną z powyższych operacji, periodycznie ze zmiennym okresem czasowym. W chwili poprzedzającej wykonanie zadania aplikacja przeprowadzała odczyt parametrów mogących mieć znaczenie podczas podejmowania decyzji o miejscu wykonania obliczeń:

1. Temperaturę urządzenia,
2. Aktualne obciążenie procesora,
3. Stan baterii,
4. Czas trwania operacji.

Sama biblioteka podejmująca decyzję została napisana w języku Java. Do napisania narzędzia zostały wykorzystane również dodatkowe biblioteki, m.in.:

1. Dagger – narzędzie do zarządzania i wstrzykiwania zależności,
2. RxJava – znany i popularny szkielet do programowania reaktywnego,
3. GreenDAO – biblioteka ORM,
4. Java Poet – biblioteka ułatwiająca generowanie plików źródłowych.

2.4. Rezultaty pracy

Podczas tworzenia niniejszej pracy zostały osiągnięte następujące rezultaty:

- Wynik analizy badania socjologicznego mającego na celu zbadanie potrzeb użytkowników i programistów aplikacji Android,
- Analiza przeprowadzonego eksperymentu, mająca na celu ujawnienie jak czynniki zewnętrzne wpływają na działanie aplikacji wykonującej operacje złożone obliczeniowo,

- Prototyp biblioteki decyzyjnej, której celem jest wygenerowanie odpowiedniego kodu pomocniczego na podstawie kodu aplikacji, monitorowanie i gromadzenie statystyk wywołań poszczególnych metod, podejmowanie decyzji o miejscu wykonywania obliczeń.

3. Badanie istoty problemu

Podczas przygotowań do napisania niniejszej pracy nie znaleziono biblioteki stanowiącej rozwiązanie badanego problemu. Najbardziej zbliżone tematycznie projekty służyły do dzielenia obliczeń pomiędzy wiele urządzeń mobilnych, lecz nie decydowały o miejscu wykonywania operacji. W celu zbadania rzeczywistej skali problemu, została przeprowadzona ankieta internetowa. Miała ona za zadanie potwierdzić, czy problem faktycznie istnieje oraz zbadać jego istotę.

3.1. Metoda badawcza

Jako metodę badawczą problemu wybrano ankietę internetową. Badanie ankietowe jest metodą badań społecznych polegającą na wypełnieniu, najczęściej samodzielnie przez badanego, specjalnych kwestionariuszy, najczęściej wysoko ustandaryzowanych w obecności lub bez obecności ankietera [1].

Przed przystąpieniem do tworzenia kwestionariusza należy dokładnie przeanalizować cel badania oraz jego tezę. Należy pamiętać, że badanie ma na celu udowodnienie lub obalenie tezy, toteż należy dołożyć wszelkich starań by ankieta pozostała bezstronna. Dotyczy to zarówno samej treści pytań jak i doboru odpowiedniej grupy docelowej. Przykładem niewłaściwego doboru grupy respondentów jest przeprowadzenie badania popularności kuchni wegańskiej w restauracji serwującej jedynie dania pochodzenia roślinnego. Klienci odwiedzający taką restaurację są zwykle silnie powiązani z weganizmem i będą faworyzować tezę, że kuchnia wegańska jest bardzo popularna. Dlatego takie badanie należy przeprowadzić w neutralnym miejscu, gdzie pojawiają się zarówno zwolennicy jak i przeciwnicy tezy w rozkładzie zbliżonym do rzeczywistego, np. na poczcie. Podobnie treść pytań nie może sugerować odpowiedzi potwierdzających zakładaną tezę. Przykładem błędnie sformułowanego pytania może być pytanie jednokrotnego wyboru o ulubione danie spośród podanych odpowiedzi, będących wyłącznie daniami wegańskimi, a następnie interpretacja odpowiedzi w sposób wskazujący na to, że wszyscy respondenci wybrali danie wegańskie jako swoje ulubione.

Dobór grupy docelowej respondentów zależy w dużym stopniu od sposobu dystrybucji ankiet. Wyróżnia się następujący podział ankiet [1]:

1. Środowiskowa (audytoryjna)
2. Pocztowa
3. Prasowa
4. Internetowa

Ankieta środowiskowa jest przeprowadzana fizycznie, na kartce papieru lub urządzeniu elektronicznym. Respondenci są zapraszani na konkretną godzinę do miejsca wypełniania ankiet, np. do ośrodka badawczego, gdzie fizycznie wypełniają kwestionariusz. Przy takim podejściu można precyzyjnie określić profil respondenta, gdyż na badanie zapraszane są tylko wybrane osoby, spełniające określone na potrzeby badania warunki. Innym sposobem przeprowadzenia ankiety środowiskowej jest wysłanie ankieterów do miejsc, gdzie mogą znaleźć się osoby odpowiadające profilowi respondenta, np. uczelnia wyższa, jeżeli grupą docelową są studenci. W takim wypadku nie można precyzyjnie określić grupy odbiorców, gdyż osoby odpowiadające na pytania są czysto przypadkowe, jednak ilość uzyskanych odpowiedzi jest zdecydowanie większa niż w przypadku zapraszania konkretnych osób.

Ankieta pocztowa jest przeprowadzana w sposób korespondencyjny. Wyselekcjonowani respondenci otrzymują kwestionariusz drogą pocztową, wypełniają go, a następnie odsyłają do ośrodka badawczego. Ta metoda pozwala dotrzeć do szerokiej grupy odbiorców, a jednocześnie daje możliwość dużej selekcji potencjalnych odbiorców. Należy jednak pamiętać, by zapewnić respondentom odpowiednią motywację do wypełnienia i odesłania ankiety, by zachować dobry stosunek ankiet otrzymanych do ankiet wysłanych. Sposobem na podniesienie poziomu odpowiedzi jest pokrycie kosztów wysyłki odpowiedzi lub oferowanie drobnych upominków w zamian za wypełnioną ankietę. W dobie cyfryzacji ankiety pocztowe są wypierane przez ankiety internetowe z powodu relatywnie wysokiego kosztu dystrybucji kwestionariuszy.

Ankieta prasowa polega na umieszczeniu kwestionariusza w gazecie lub czasopiśmie. Respondenci dobrowolnie wypełniają kwestionariusz i odsyłają go drogą pocztową do ośrodka badawczego. Ankieta prasowa znacząco obniża koszt dystrybucji kwestionariuszy oraz pozwala na dotarcie do szerokiego grona odbiorców. Należy jednak pamiętać o tym, że nie ma możliwości dokładnego określenia profilu respondenta, gdyż odbiorcą ankiety są czytelnicy danego czasopisma lub gazety. Ponadto, czytelnicy mają małą motywację by faktycznie wypełnić ankietę, stąd też główną wadą ankiety prasowej jest niski poziom odpowiedzi.

Ankieta internetowa jest obecnie najszerzej stosowanym sposobem ankietowania. Do jej zalet należy bardzo niski koszt dystrybucji kwestionariuszy oraz możliwość dotarcia do szerokiego grona odbiorców. Przy ankiecie internetowej można też w łatwy sposób zwiększyć motywację respondentów oferując upominki w formie elektronicznej, np. dostęp do e-booka czy kody zniżkowe na usługi i towary. Jednak ankieta internetowa oferuje najmniejszą kontrolę nad grupą docelową. Ankiety otwarte nie gwarantują, że respondenci wypełnią ankietę tylko jeden raz a wielokrotne wypełnianie tego samego kwestionariusza przez tego samego respondenta wypacza wynik badania. By temu zapobiec, stosuje się ankiety zamknięte, do których dostęp uzyskuje się przez podanie jednorazowego hasła. Hasło jest

generowane indywidualnie dla każdego respondenta i dystrybuowane przy pomocy poczty elektronicznej. Wadą takiego postępowania jest konieczność znajomości listy adresów odbiorców co znacząco zawęża potencjalną grupę respondentów. Pomimo swoich wad, ankiety internetowe są obecnie najszerzej stosowanym sposobem ankietowania, szczególnie w badaniach niewymagających ściśle określonego profilu odbiorcy.

Innym kryterium podziału jest jawność odpowiedzi. Według tego kryterium ankiety możemy podzielić na jawne i anonimowe. W przypadku ankiety jawnej wszyscy respondenci mają dostęp do odpowiedzi innych respondentów. Ankieta anonimowa zapewnia pełną poufność danych a respondenci mają dostęp jedynie do zbiorczych wyników ankiety, prezentowanych po zakończeniu badania. Informacja o poufności ankiety powinna znaleźć się w pierwszym elemencie kwestionariusza jakim jest list wprowadzający.

Kwestionariusz składa się z trzech elementów:

1. List wprowadzający
2. Zestaw pytań
3. Podziękowanie i instrukcja dalszego postępowania

W liście wprowadzającym powinna znaleźć się informacja identyfikująca instytucję lub osobę przeprowadzającą badanie, cel samego badania oraz powody, dla których respondent powinien odpowiedzieć na pytania. Dobre sformułowanie listu wprowadzającego jest kluczowe dla poziomu odpowiedzi ankiety. Zbyt długi list zniechęci respondenta do jego przeczytania, natomiast zupełny brak listu znacząco obniża wiarygodność instytucji przeprowadzającej badanie.

Treść właściwa kwestionariusza to ustandaryzowany zestaw pytań. Wyróżniamy kilka rodzajów pytań:

1. Pytania zamknięte jednokrotnego i wielokrotnego wyboru
2. Pytania otwarte
3. Pytania macierzowe
4. Pytania ze skalą szacunkową
5. Pytania zakresowe

Treść pytania powinna być sformułowana w sposób konkretny i jasny. Należy unikać wieloznacznych słów oraz nielogicznych sformułowań. Respondent musi dokładnie wiedzieć jaką informację chce uzyskać ankieter, szczególnie jeżeli ankieta nie jest wypełniana w obecności ankietera i nie ma możliwości doprecyzowania pytania. Dobrą praktyką jest wieloetapowy proces

przygotowywania kwestionariusza, w którym pytania są testowane na grupie kontrolnej bez faktycznego zbierania odpowiedzi. Zadaniem grupy testowej jest sprawdzenie czy pytania są jednoznaczne i zrozumiałe dla przyszłych, właściwych respondentów. Badania testowe niemal zawsze przeprowadza się w obecności ankietera, który obserwuje zachowania grupy testowej i na tej podstawie wyciąga wnioski dotyczące jakości pytań kwestionariusza.

Badanie ankietowe ma wiele niekwestionowanych zalet [2]:

1. Niski koszt badania społecznego,
2. Stosunkowo krótki czas gromadzenia danych,
3. Możliwość dotarcia do bardzo szerokiej grupy respondentów w krótkim czasie, szczególnie w przypadku ankiety internetowej,
4. Niska uciążliwość dla respondentów, co przekłada się na dobry stosunek uzyskanych odpowiedzi do wszystkich wysłanych ankiet.

Wadami badania ankietowego są:

1. Niski i trudny do oszacowania stopień wiarygodności z powodu stronniczości respondentów i możliwości konsultacji,
2. Brak możliwości weryfikacji wiarogodności udzielonych odpowiedzi, szczególnie w przypadku ankiety internetowej,
3. Niezrozumiałe pytania powodujące przekłamanie wyniku całego badania,
4. W przypadku ankiety anonimowej brak możliwości zestawienia odpowiedzi z profilem badanej osoby,
5. Brak możliwości precyzyjnej selekcji grupy respondentów w przypadku badania internetowego.

3.2. Wybór metody

W ramach pracy magisterskiej przeprowadzono dwie ankiety. Jedna została przeznaczona dla użytkowników smartfonów i jej celem było zbadanie w jaki sposób użytkownicy korzystają z urządzeń elektronicznych do zadań wymagających stosunkowo dużej mocy obliczeniowej urządzenia. Ponadto, ankieta miała sprawdzić jakie cechy smartfonu użytkownicy cenią sobie najbardziej i na czym im najbardziej zależy. Druga ankieta miała na celu zbadanie świadomości twórców aplikacji o istnieniu problemu oraz poznanie ich doświadczeń związanych ze stawianą tezą.

Analizując wady i zalety różnych metod ankietowania, do sprawdzenia tezy stawianej w pracy wykorzystano ankiety internetowe. Motywację tego wyboru stanowiły:

1. Brak zamkniętej listy respondentów,
2. Ograniczone środki finansowe,
3. Ograniczone możliwości dotarcia do szerokiego grona odbiorców,
4. Brak konieczności profilowania respondentów w ankiecie dla użytkowników,
5. Łatwy i pewny dostęp do respondentów w ankiecie dla programistów.

Ankieta została zaimplementowana przy pomocy serwisu internetowego surveymonkey.com.

3.3. Ankieta dla użytkowników

Grupą docelową ankiety dla użytkowników były osoby posiadające i korzystające na co dzień ze smartfonu w celach prywatnych. Ankieta składała się z 9 pytań. Taka liczba pytań była optymalna, gdyż dawała wystarczającą ilość danych do potwierdzenia lub obalenia tezy, jednocześnie nie była nużąca dla respondentów. Również liczba dostępnych odpowiedzi została ograniczona do minimum by skrócić czas potrzebny na zapoznanie się z pytaniem. Poza jednym pytaniem, liczba dostępnych odpowiedzi nie przekraczała pięciu. Ankieta została przeprowadzona w języku polskim.

Link do ankiety internetowej był dystrybuowany przy pomocy sieci społecznościowych oraz poczty elektronicznej. Magistrant umieścił ankietę na swoich profilach społecznościowych w serwisach wykop.pl oraz facebook.com oraz rozesłał ją poprzez pocztę elektroniczną do znajomych z prośbą o propagację. Takie działanie pozwoliło uzyskać dużą liczbę odpowiedzi, jednak uniemożliwił stworzenie profilu respondenta. Jedną pewną informacją o ankietowanych jest fakt, że korzystają oni z Internetu oraz że znają język polski, gdyż w takim języku został przygotowany kwestionariusz.

3.3.1 Cel badania

Celem badania jest poznanie potrzeb i problemów użytkowników podczas codziennego korzystania z urządzeń mobilnych. Ankieta miała za zadanie sprawdzić czy oferta aplikacji mobilnych jest wystarczająco bogata, by urządzenia mobilne mogły zastąpić konwencjonalne komputery PC. Pytania kwestionariusza sformułowane są tak, by sprawdzić, czy użytkownicy już teraz wykorzystują wyłącznie smartfony do zadań domowych. Ponadto, ankieta miała za zadanie potwierdzić lub obalić tezy znane magistrantowi z obserwacji własnej pracy zawodowej, m.in. maksymalny czas oczekiwania użytkownika na zakończenie długotrwałej operacji czy najbardziej irytujące czynniki wpływające na

frustrację użytkowników. Celem badania było także sprawdzenie czy powstanie automatycznej biblioteki decyzyjnej ma sens w kontekście rozpraszania obliczeń do serwisów zewnętrznych. Jeżeli użytkownicy nie mają na stałe podłączonego urządzenia do Internetu lub rygorystycznie kontrolują przesyłane dane przez aplikacje to przenoszenie obliczeń do chmury obliczeniowej zmniejszy ich komfort korzystania z urządzeń mobilnych a biblioteka będzie stale podejmowała tę samą decyzję o lokalnych wykonaniu obliczeń.

3.3.2 Wyniki ankiety

Ankieta składała się z ośmiu pytań zamkniętych jednokrotnego wyboru oraz z jednego pytania wielokrotnego wyboru z ograniczoną do 3 liczbą wybranych odpowiedzi. Otrzymano 3552 w pełni wypełnionych kwestionariuszy.

Pierwsze pytanie miało na celu sprawdzenie czy użytkownicy preferują używanie laptopa czy urządzenia mobilnego takiego jak smartfon czy tablet. Jako czynność wybrano obróbkę zdjęć i filmów, ponieważ jest to czynność popularna w dobie mediów społecznościowych i dzielenia się wspomnieniami ze znajomymi. Jednocześnie czynność ta wymaga odpowiedniej mocy obliczeniowej procesora by odbywała się w sposób niemal natychmiastowy. Testy wykonane na różnych urządzeniach mobilnych wskazują na długi czas oczekiwania na zakończenie obróbki na starszych urządzeniach lub z niższej półki. Wyniki pytania przedstawia Wykres 1.

Pytanie: Na jakim urządzeniu najczęściej obrabiasz zdjęcia i filmy?

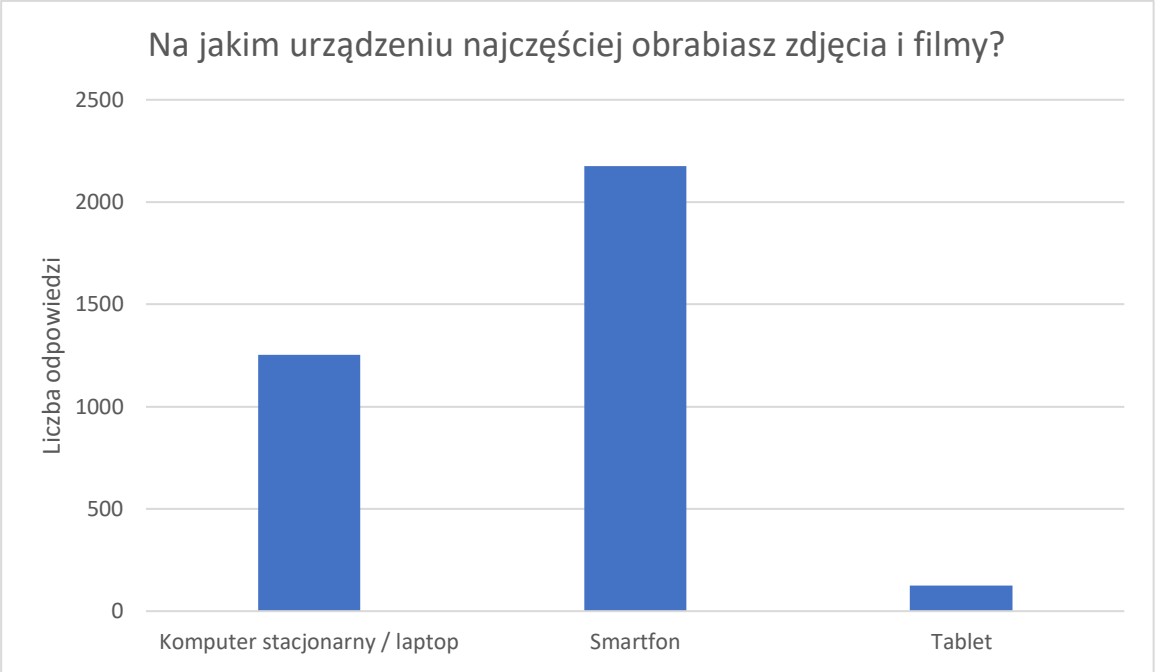
Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Komputer stacjonarny / laptop - 1252 / 35,25%,

B) Smartfon - 2176 / 61,26%,

C) Tablet - 124 / 3,49%.



Wykres 1: Wyniki pytania 1 ankiety dla użytkowników

Prawie dwie trzecie ankietowanych korzysta z urządzeń przenośnych do wykonywania obróbki zdjęć i filmów. Około 35% respondentów używa do tego celu komputera PC lub laptopa. Oznacza to, że pomimo dużo wygodniejszej obsługi przy pomocy myszki i klawiatury oraz znacznie mniejszej mocy obliczeniowej ludzie preferują korzystanie z urządzeń mobilnych. Należy także pamiętać, że obróbka zdjęć i filmów pochłania znaczną część energii elektrycznej, zarówno poprzez wymagające operacje dla procesora jak i stale włączony wyświetlacz. Pomimo wymienionych wyżej niedogodności, respondenci preferują używanie urządzeń mobilnych do operacji, wydawałoby się, zarezerwowanych dla komputerów PC.

Celem pytania drugiego było sprawdzenie jak ludzie podchodzą do przesyłania bardzo dużych ilości danych przez Internet, a także jak często ludzie potrzebują odpowiedniej mocy obliczeniowej do prostych operacjach na filmach. Publikacja filmu w portalu społecznościowym wiąże się z wykonaniem jego kompresji przed wysłaniem na serwer serwisu. Oprócz samej kompresji, wiele portali umożliwia edycję i montaż filmu przed wysłaniem, cięcie, dodawanie elementów. Takie operacje wiążą się zwykle z przekodowaniem całego filmu co jest szczególnie wymagające obliczeniowo. Wyniki pytania przedstawia Wykres 2.

Pytanie: Jak często zdarza Ci się dodawać filmy na portale społecznościowe?

Typ pytania: zakresowe.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

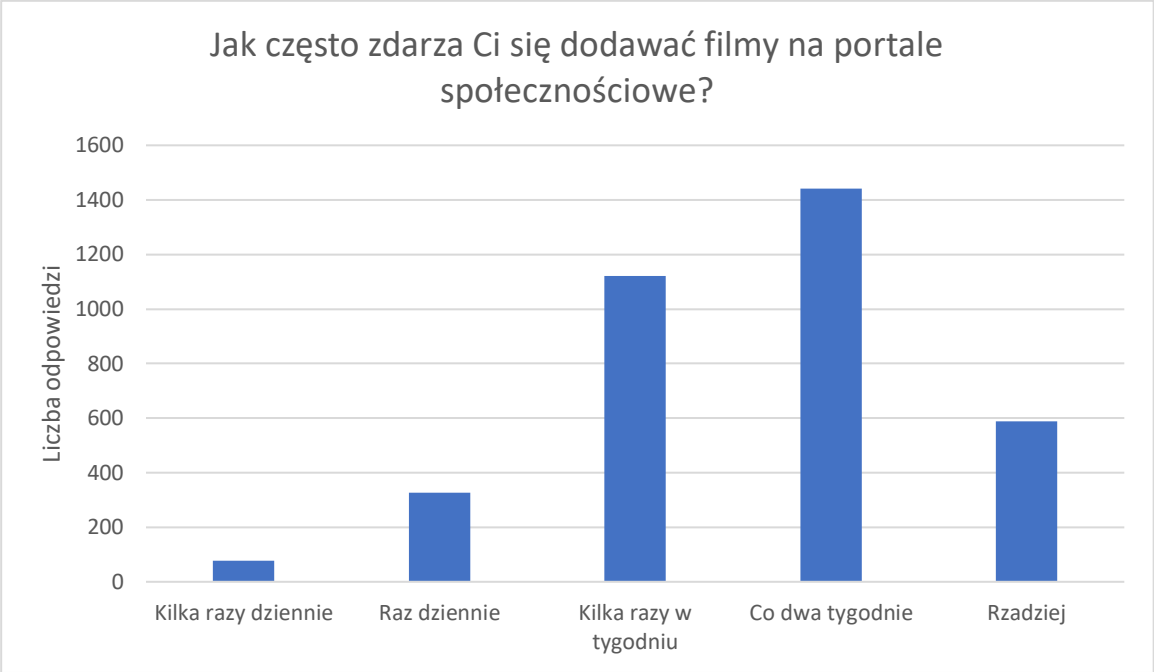
A) Kilka razy dziennie - 77 / 2,17%,

B) Raz dziennie - 326 / 9,18%,

C) Kilka razy w tygodniu - 1120 / 31,53%,

D) Co dwa tygodnie - 1441 / 40,57%,

E) Rzadziej - 588 / 16,55%.



Wykres 2: Wyniki pytania 2 ankiety dla użytkowników

Wyniki ankiety wskazują na brak oporów do przesyłania dużych plików przez Internet. Prawie 43% respondentów wykonuje taką czynność przynajmniej kilka razy w tygodniu. Oznacza to, że nie jest to czynność wykonywania sporadycznie a regularnie, ze stosunkowo dużą częstotliwością. Optymalizacja tej czynności spowoduje wzrost zadowolenia użytkowników.

Pytanie trzecie było najważniejsze w całym kwestionariuszu. Celem tego pytania było poznanie preferencji użytkowników w zakresie codziennego korzystania z urządzeń mobilnych. Identyfikacja potrzeb i problemów użytkowników jest kluczowa do zwiększenia komfortu korzystania ze smartfonów.

By nie ograniczać użytkowników ankieta dopuszczała maksymalnie 3 odpowiedzi na to pytanie. Wyniki pytania przedstawia Wykres 3.

Pytanie: Co Cię irytuje najbardziej podczas korzystania ze smartfonu?

Typ pytania: wielokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Krótki czas pracy na baterii - 3202 / 90,15%,

B) Zbyt duże zużycie danych - 582 / 16,39%,

C) Wysoka temperatura urządzenia - 311 / 8,76%,

D) Powolne działanie - 1102 / 31,02%,

E) Nieresponsywny interfejs - 89 / 2,51%,

F) Brak odpowiednich aplikacji - 22 / 0,62%,

G) Mały ekran - 199 / 5,6%,

H) Oczekiwanie na zakończenie operacji - 675 / 19%,

I) Mała ilość pamięci na dane - 294 / 8,28%.

Wykres 3: Wyniki pytania 3 ankiety dla użytkowników

Wyraźnie najbardziej irytującym czynnikiem jest krótki czas na baterii. Tę niedogodność wybrało ponad 90% respondentów. Oznacza to, że pomimo licznych możliwości jakie daje użytkownikom smartfon, bardzo ważną kwestią jest optymalizacja sposobu w jaki aplikacje realizują powierzone jej zadania. Argument ten jest bardzo ważny dla potwierdzenia tezy stawianej w ankiecie. Warto też zwrócić uwagę na stosunkowo niewielki procent respondentów wybierających zbyt duże zużycie danych komórkowych jako element irytujący. Oznacza to, że użytkownicy rzadko zwracają uwagę na ilość danych przesyłanych przez ich telefony. Warto tutaj dodać, że operatorzy komórkowi prześcigają się w oferowaniu coraz lepszych planów taryfowych w atrakcyjnych cenach, zatem można spodziewać się, że znaczenie tego argumentu będzie systematycznie spadać. Innym ciekawym faktem jest znikomy procent respondentów wybierających odpowiedź „brak odpowiednich aplikacji". To pokazuje, że urządzenia typu smartfon oferują szeroką gamę oprogramowania dla każdego. Biorąc pod uwagę łatwą dostępność tego typu urządzeń oraz rosnącą moc obliczeniową można stwierdzić, że w bliżej nieokreślonej przyszłości urządzenia te zupełnie wyprą komputery klasy PC w zastosowaniach domowych.

Czwarte pytanie miało na celu sprawdzenie jak często respondenci korzystają z konwencjonalnego komputera PC do zastosowań domowych. Wyniki pytania przedstawia Wykres 4.

Pytanie: Jak często korzystasz z komputera stacjonarnego / laptopa w celach prywatnych?

Typ pytania: zakresowe.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Kilka razy dziennie - 184 / 5,18%,

B) Raz dziennie - 732 / 20,61%,

C) Kilka razy w tygodniu - 1532 / 43,13%,

D) Co dwa tygodnie - 738 / 20,78%,

E) Rzadziej - 366 / 10,3%.

Wykres 4: Wyniki pytania 4 ankiety dla użytkowników

Wyniki wskazują, że komputer stacjonarny nadal jest popularnym urządzeniem, jednak tylko co czwarty ankietowany korzysta z niego codziennie. Oznacza to, że aplikacje mobilne nie wyparły jeszcze zupełnie komputerów PC do zastosowań domowych.

Celem pytania piątego było sprawdzenie czy implementacja biblioteki decyzyjnej rozdzielającej obliczenia pomiędzy urządzenie mobilne a serwis zdalny ma sens, gdyż bez dostępu do Internetu obliczenia będą mogły być wykonywane jedynie lokalnie na urządzeniu mobilnym. Wyniki pytania przedstawia Wykres 5.

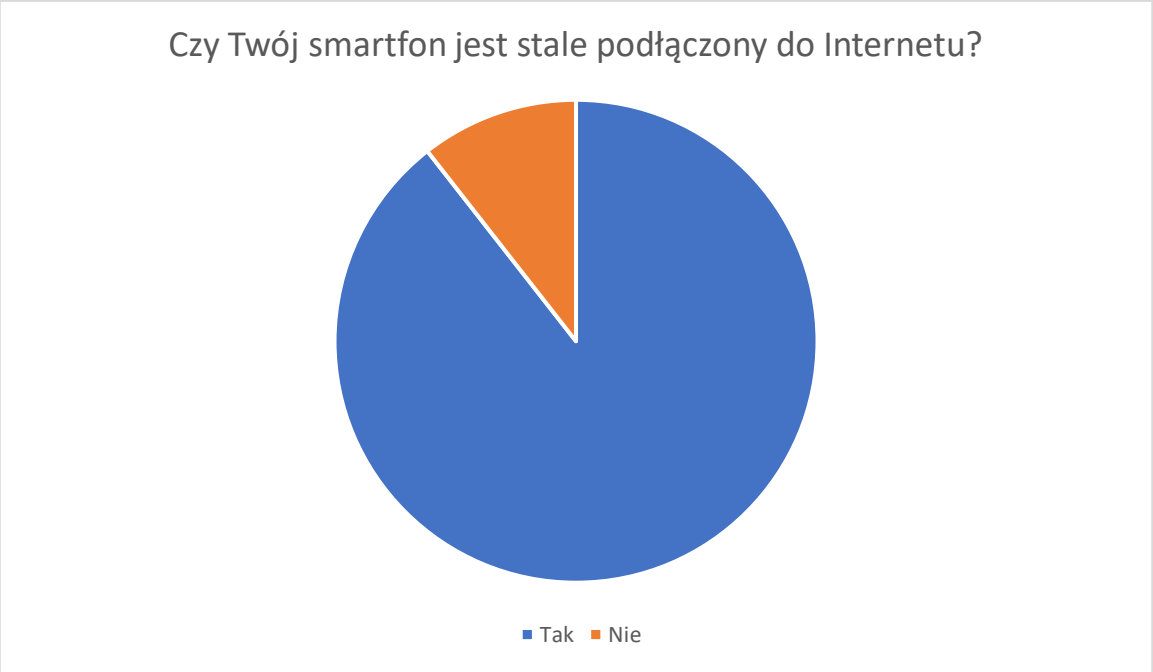
Pytanie: Czy Twój smartfon jest stale podłączony do Internetu?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Tak - 3177 / 89,44%,

B) Nie - 375 / 10,56%.



Wykres 5: Wyniki pytania 5 ankiety dla użytkowników

Niemal 90% ankietowanych ma telefon stale podłączony do Internetu. Znaczna część aplikacji mobilnych to aplikacje łączące się z chmurą danych. Użytkownicy korzystają z wielu różnych urządzeń i chcą być stale zsynchronizowani ze swoją pracą. Tak duży odsetek użytkowników stale podłączonych do Internetu sprawia, że tworzenie biblioteki decyzyjnej, której założeniem jest wysyłanie danych do chmury obliczeniowej jest zasadne, gdyż jej możliwości będą wykorzystane przez niemal 90% wszystkich użytkowników aplikacji.

Pytanie szóste miało na celu zbadać cierpliwość użytkowników. Z doświadczeń zawodowych magistranta wynika, że maksymalnym czasem oczekiwania na zakończenie operacji przed utratą cierpliwości jest około 10 sekund. Potwierdzają to dane dotyczące sposobu używania aplikacji mobilnych zebrane podczas pracy zawodowej magistranta, jednak podlegające tajemnicy pracodawcy. Z tego powodu pytanie trafiło do kwestionariusza by potwierdzić lub obalić wcześniejsze obserwacje. Wyniki pytania przedstawia Wykres 6.

Pytanie: Jak często zdarza Ci się wyłączać aplikację, gdy ta wykonuje jakąś operację, na którą musisz czekać dłużej niż 10 sekund?

Typ pytania: zakresowe.

Odpowiedzi i wyniki (liczba odpowiedzi / procent)::

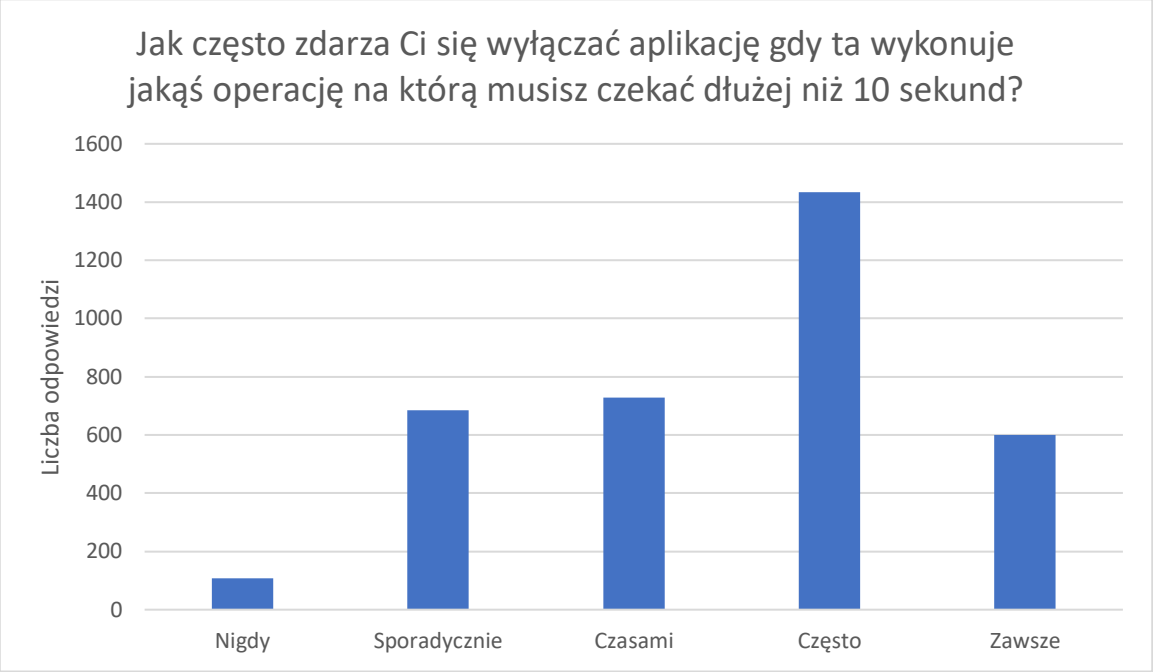
A) Nigdy - 107 / 3,01%,

B) Sporadycznie - 684 / 19,26%,

C) Czasami - 729 / 20,52%,

D) Często - 1433 / 40,34%,

E) Zawsze - 599 / 16,86%.



Wykres 6: Wyniki pytania 6 ankiety dla użytkowników

Ponad połowa użytkowników często lub zawsze wyłącza aplikację, podczas gdy operacja, na którą oczekują, trwa dłużej niż 10 sekund. Jest to zrozumiałe biorąc pod uwagę sposób korzystania ze smartfona. Zwykle urządzenie mobilne jest trzymane przez użytkowników w ręku co powoduje dyskomfort, mając na uwagę fakt, że użytkownik w trakcie oczekiwania na zakończenie operacji nie może wykonać żadnej innej czynności. Nie bez znaczenia jest fakt, że podczas tak długiego wykonywania obliczeń znacząco wzrasta temperatura urządzenia, co dodatkowo potęguje efekt dyskomfortu. Wyniki pytania potwierdzają wcześniejsze obserwacje. Podczas implementacji aplikacji mobilnej kluczowym jest unikanie operacji dłuższych niż 10 sekund. Jeżeli taka operacja jest konieczna, niezbędne jest wykonywanie jej w tle by nie ograniczać swobody korzystania z urządzenia mobilnego.

Pytanie siódme miało na celu sprawdzenie czy użytkownikom może przeszkadzać zwiększona ilość przesyłanych danych przez sieci komórkowe. Wykorzystanie biblioteki decyzyjnie miałoby znaczący wpływ na ilość transferowanych bitów. W zależności od jej konfiguracji i przyjętej strategii można zarówno ograniczyć zużycie danych poprzez wykonywanie obliczeń na urządzeniu, jak i

znacząco zwiększyć przenosząc obliczenia do chmury obliczeniowej. Wyniki pytania przedstawia Wykres 7.

Pytanie: Czy monitorujesz ilość zużytych danych przez Twój smartfon?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Tak - 1570 / 44,2%,

B) Nie - 1982 / 55,8%.

Wykres 7: Wyniki pytania 7 ankiety dla użytkowników

Wyniki pytania wskazują na brak jednoznacznej odpowiedzi. Większość użytkowników wskazała, że zupełnie nie monitoruje zużytych danych przez ich smartfon. Ponad 44% ankietowanych kontroluje transfer danych, jednak nie oznacza to, że nie byliby skłonni zwiększyć ilości przesyłanych danych na rzecz szybszego wykonywania operacji obliczeniowych czy też zwiększenia czasu pracy na baterii.

Celem pytania ósmego było sprawdzenie czy oferta aplikacji mobilnych jest wystarczająco bogata by smartfon mógł zastąpić komputer klasy PC w zastosowaniach domowych. Według portalu Internetowego Statista [4] w sklepie Google Play, głównym kanale dystrybucji aplikacji na platformę

Android, jest już ponad 3 600 000 różnych aplikacji. Liczba ta stale rośnie. Osobną kwestią jest jakość samych aplikacji oraz ich różnorodność. Wyniki pytania przedstawia Wykres 8.

Pytanie: Czy kiedykolwiek zdarzyła Ci się sytuacja, że nie znalazłeś / nie znalazłaś potrzebnej Ci aplikacji na smartfon?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Nie, zawsze się znalazła - 2867 / 80,72%.

B) Znalazłem/Znalazłam, ale była płatna - 75 / 2,11%.

C) Tak, nie było ani darmowej, ani płatnej wersji - 610 / 17,17%.

Wykres 8: Wyniki pytania 8 ankiety dla użytkowników

Wyniki pytania wskazują, że jedynie lekko ponad 17% użytkowników nie znalazło odpowiedniej dla siebie aplikacji. Prawie 83% użytkowników nie miało problemu ze znalezieniem aplikacji spełniającej ich oczekiwania. Tylko 2% respondentów nie znalazło darmowego odpowiednika. Wyniki pytania potwierdzają, że oferta sklepu Play jest nie tylko szeroka w sensie ilości aplikacji, ale także ich użyteczności i jakości.

Pytanie 9 miało na celu sprawdzić w jako sposób użytkownicy korzystają ze smartfonu. Zdecydowana większość urządzeń z systemem Android umożliwia podłączenie telefonu do

zewnętrznego ekranu. Ponadto do takich urządzeń można podłączyć mysz i klawiaturę poprzez interfejs Bluetooth lub kabel USB. Taki zestaw gwarantuję wysoką wygodę użytkowania, porównywalną z konwencjonalnym komputerem klasy PC z dokładnością do mniejszej mocy obliczeniowej. Część smartfonów jest sprzedawana z dedykowaną stacją dokującą pozwalającą na szybką zmianę urządzenia przenośnego w stację multimedialną bądź komputer PC, w zależności od potrzeb i konfiguracji. Nowsze urządzenia pozwalają na bezprzewodowe klonowanie ekranu przy pomocy sieci Wi-Fi co ułatwia podłączenie ekranu zewnętrznego. Wyniki pytania przedstawia Wykres 9.

Pytanie: Czy korzystasz z telefonu / tabletu podłączonego do telewizora lub monitora? (Kablem, bezprzewodowo lub przez stację dokującą)

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Tak - 217 / 6,11%,

B) Nie - 3335 / 93,89 %.

Wykres 9: Wyniki pytania 9 ankiety dla użytkowników

Wyniki pytania 9 są jednoznaczne: użytkownicy nie preferują podłączania urządzenia do zewnętrznego ekranu i korzystają z wyświetlacza urządzenia. W kontekście biblioteki decyzyjnej oznacza to konieczność wzięcia pod uwagę zwiększonego poboru energii elektrycznej z akumulatora

urządzenia. Dla programistów aplikacji mobilnych jest to wyraźny sygnał, że nie ma potrzeby projektowania osobnego interfejsu przeznaczonego dla myszy i klawiatury.

3.3.3 Podsumowanie ankiety dla użytkowników

W badaniu wzięło udział 4228 respondentów z czego 3552 odpowiedziała na wszystkie pytania. Liczba ta jest bardzo wysoka w stosunku do oczekiwanej. Ankieta nie miała dodatkowego motywatora w postaci upominków czy kwot pieniężnych. Użytkownicy z własnej woli, na prośbę magistranta, wypełnili ankietę. Nie bez znaczenia był sposób dystrybucji ankiety. Dzięki sieci Internet ankieta mogła dotrzeć do szerokiego grona odbiorców w łatwy i tani sposób.

Statystyki z serwisu surveymonkey.com dotyczące tego konkretnego kwestionariusza wskazują na bardzo duży procent respondentów wypełniających ankietę w całości w stosunku do wszystkich wypełniających. Ponad 84% wszystkich biorących udział w badaniu odpowiedziało na wszystkie pytania ankiety. Kwestionariusz zawierał tylko 9 pytań zamkniętych, bez możliwości wpisywania tekstu. Średni czas wypełniania ankiety to 4 minuty i 38 sekund, zaś najszybszy respondent uporał się z ankietą w czasie 2 minut i 14 sekund. Tak krótki średni czas wypełniania ankiety przyczynił się do dużego poziomu odpowiedzi.

3.3.4 Wnioski z ankiety dla użytkowników

Część wniosków została już opisana przy podsumowaniu poszczególnych pytań. W tym podpodrozdziale wnioski zostaną usystematyzowane.

Z badania wynika, że użytkownicy coraz częściej wykorzystują urządzenia mobilne zamiast konwencjonalnych komputerów PC do podstawowych czynności domowych, np. obróbki zdjęć lub filmów. Nie mają też problemu ze znalezieniem odpowiedniej dla siebie aplikacji, spełniającej zakładane potrzeby. Większość tych aplikacji jest darmowa co czyni je idealnymi do użytku domowego. Respondenci często korzystają z dobrodziejstw sieci społecznościowych i nie mają oporów by wykorzystywać urządzenia mobilne do prostej edycji i montażu filmów przed podzieleniem się nimi ze znajomymi. Ponadto, zdecydowana większość ankietowanych jest na stałe podłączona do Internetu. Z ankiety wynika, że mniej niż połowa użytkowników faktycznie monitoruje ilość transferowanych danych, aczkolwiek nie oznacza to, że nie będą oni skłonni zwiększyć swojego zużycia danych odczuwają inne pozytywne aspekty działania biblioteki, takie jak dłuży czas pracy urządzenia bez ładowania. Z czynników szczególnie irytujących użytkowników podczas korzystania ze smartfonów można wymienić zbyt krótki czas pracy na baterii i powolne działanie urządzenia. Biblioteka decyzyjna ma za zadanie poprawić oba te aspekty, w zależności od urządzenia, na którym uruchomiona jest aplikacja.

3.4. Ankieta dla programistów

Grupą docelową ankiety byli programiści aplikacji Android, którzy mają zawodowe doświadczenie w pisaniu aplikacji mobilnych. Kwestionariusz składał się z 8 pytań, w tym jedno pytanie było otwarte. Pomimo tego liczba odpowiedzi była zaskakująco wysoka i sięgnęła 217 respondentów. Podobnie jak w przypadku ankiety dla użytkowników, liczba dostępnych odpowiedzi była ograniczona do minimum i nie przekraczała sześciu. Ankieta została przeprowadzona w języku polskim.

Podobnie jak w przypadku ankiety dla użytkowników, link do ankiety internetowej był dystrybuowany przy pomocy sieci Internet. Głównym medium dystrybucji była poczta elektroniczna. Magistrant rozesłał kwestionariusz do znajomych programistów z prośbą o propagację wyłącznie wśród osób z doświadczeniem programistycznym. Ankieta została umieszczona także na portalach społecznościowych powiązanych z tematyką programowania na platformę Android. W przypadku ankiety dla programistów niemożliwe było czy respondent faktycznie miał doświadczenie zawodowe w programowaniu. Jednakże brak jakiejkolwiek nagrody za wypełnienie ankiety zmniejszył ryzyko fałszywych odpowiedzi.

3.4.1 Cel badania

Celem ankiety dla programistów było zbadanie świadomości programistów istnienia problemu oraz zainteresowania ewentualnym rozwiązaniem. Ponadto, ankieta miała za zadanie ustalić profil tworzonych aplikacji celem sprawdzenia, czy proponowana w ramach pracy magisterskiej biblioteka znajdzie zastosowanie w codziennej pracy. Badanie miało na celu ustalenie jakie rodzaje operacji złożonych obliczeniowo są najczęściej wykorzystywane przez programistów. Dzięki takiej informacji można ustalić, czy jest możliwe podzielenie tych operacji i wykonywanie ich zamiennie lokalnie na urządzeniu oraz zdalnie na serwerze.

3.4.2 Wyniki ankiety

Ankieta składała się z sześciu pytań jednokrotnego wyboru, jednego pytania wielokrotnego wyboru oraz jednego pytania otwartego. Odpowiedź na pytanie otwarte była opcjonalna, by nie zniechęcać potencjalnych respondentów koniecznością udzielania pisemnej wypowiedzi.

Celem pierwszego pytania było ustalenie czy aplikacje tworzone przez programistów korzystają z zewnętrznych serwisów. Bez serwisu zewnętrznego wykorzystanie biblioteki decyzyjnej nie miałoby sensu. Sprawdzenie jak często programiści wykorzystują połączenia zdalne do implementacji części funkcjonalności pozwoliło na ocenę popularności modelu aplikacji z przeniesioną częścią logiki biznesowej na serwer zewnętrzny. Wyniki przedstawia Wykres 10.

Pytanie: Jak często Twoja aplikacja łączy się z zewnętrznym serwisem?

Typ pytania: zakresowe.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

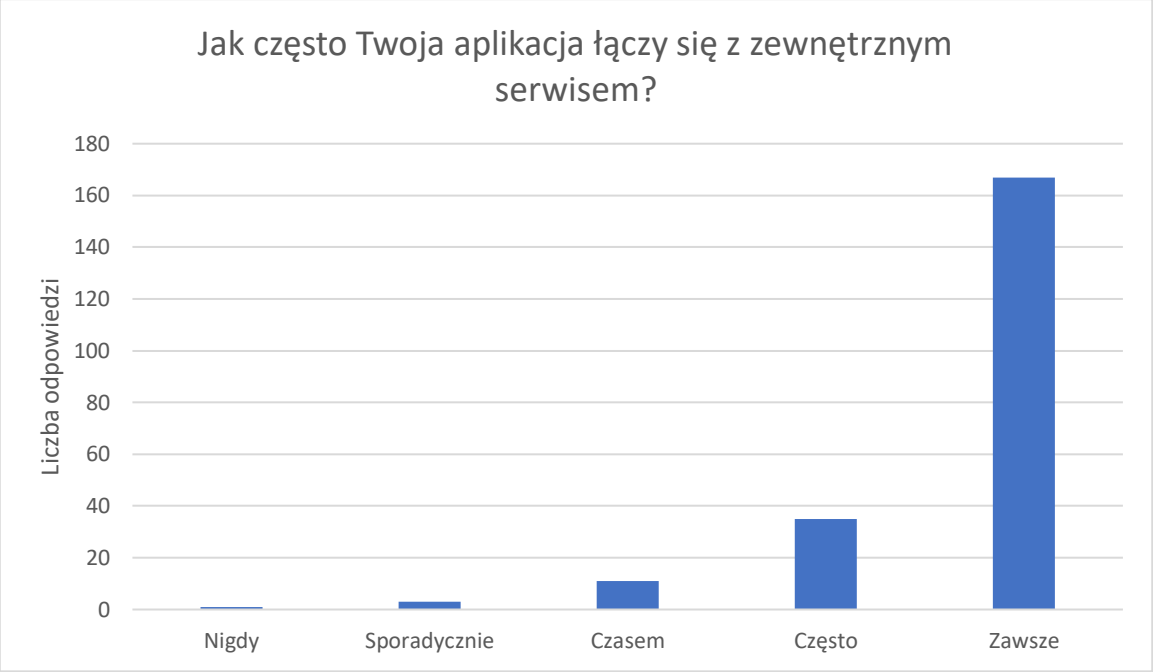
A) Nigdy - 1 / 0,46%,

B) Sporadycznie - 3 / 1,38%,

C) Czasem - 11 / 5,07%,

D) Często - 35 / 16,13%,

E) Zawsze - 167 / 76,96%.



Wykres 10: Wyniki pytania 1 ankiety dla programistów

Wyniki ankiety potwierdzają popularność modelu aplikacji z rozproszoną logiką biznesową na zewnętrznym serwisie. Ponad 75% badanych deklaruje, że każda stworzona przez nich aplikacja korzysta z połączenia z serwisem zdalnym. Ponad 93% respondentów korzysta z takiego rozwiązania co najmniej często. Jest to szczególnie ważne w kontekście omawianej biblioteki decyzyjnej. Obecność zewnętrznego serwisu w projekcie znacząco ułatwia dodawanie do niego nowych funkcjonalności. W warunkach komercyjnych bardzo trudną do podjęcia decyzją projektową jest fakt wdrożenia zewnętrznego serwisu. Zazwyczaj wiąże się to z koniecznością zaangażowania kolejnej osoby do zespołu projektowego, zwiększeniem ilości potrzebnego czasu na stworzenie rozwiązania a co za tym idzie znacznego wzrostu kosztu produkcji oprogramowania. Generuje to również dodatkowe koszty

związane z konfiguracją i obsługą środowisk testowych i produkcyjnych dla serwisu zewnętrznego. Wynik ankiety wskazuje, że taka decyzja często jest podejmowana na korzyść włączenia serwisu zewnętrznego do architektury projektu. Należy zauważyć, że w przypadku, gdy serwis zewnętrzny już istnieje, zdecydowanie łatwiej jest dopisać do niego dodatkowe metody. Wynik ankiety potwierdza, że wprowadzenie biblioteki nawet do istniejących aplikacji nie będzie wiązało się z dużą ingerencją w architekturę rozwiązania, a w przypadku nowych aplikacji nie będzie się wiązało z wprowadzeniem dodatkowych kosztów na etapie projektowania.

Pytanie drugie miało na celu sprawdzenie jak często programiści tworząc aplikacje mobilne korzystają z algorytmów mogących znacząco obciążać procesor urządzenia a tym samym negatywnie wpływać na czas życia baterii. W kontekście proponowanej biblioteki ma to duże znaczenie, gdyż pozwala oszacować czy taka biblioteka jest w ogóle potrzebna. Wyniki przedstawia Wykres 11.

Pytanie: Jak często implementujesz operacje wymagające obliczeniowo? (czas ich wykonania jest na tyle znaczący, że użytkownik czeka na jej zakończenie)

Typ pytania: zakresowe.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

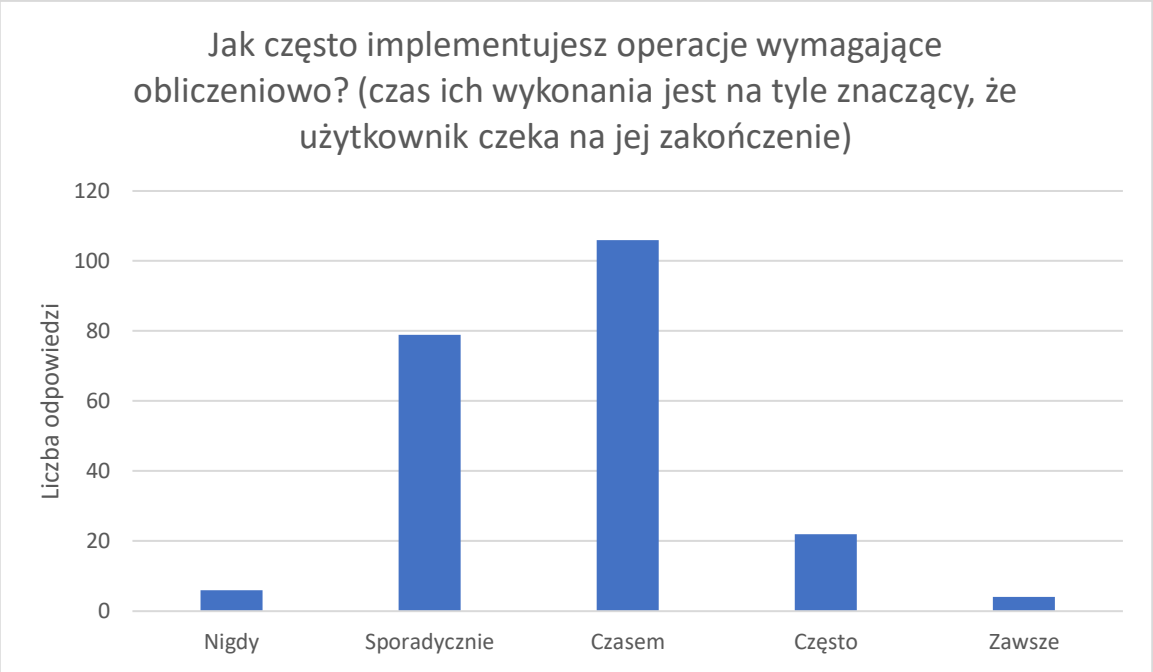
A) Nigdy - 6 / 2,76%,

B) Sporadycznie - 79 / 36,41%,

C) Czasem - 106 / 48,85%,

D) Często - 22 / 10,14%,

E) Zawsze - 4 / 1,84%.



Wykres 11: Wyniki pytania 2 ankiety dla programistów

Z ankiety wynika, że implementowanie złożonych operacji nie jest powszechną praktyką dla aplikacji mobilnych. Niemal połowa ankietowanych tylko czasami włącza do swoich aplikacji bardziej skomplikowane obliczenia. Ponad jedna trzecia badanych robi to zaledwie sporadycznie. Potencjalnym wykorzystaniem biblioteki decyzyjnej mogłoby być zainteresowanych jedynie około 12% badanych.

Pytanie trzecie było pytaniem otwartym i miało na celu identyfikację najczęściej implementowanych operacji złożonych obliczeniowo. Celem pytania było sprawdzenie czy dla takich operacji zastosowanie biblioteki decyzyjnej znacząco wpłynie na odbiór aplikacji przez użytkownika końcowego.

Pytanie: Jakie operacje wymagające obliczeniowo implementujesz w swoich aplikacjach.

Rodzaj pytania: otwarte.

Odpowiedzi zgrupowane (liczba odpowiedzi):

- Kompresja danych przed wysłaniem przez sieć (18),
- Dekompresja danych pobranych z sieci (7),
- Operacje na plikach video: montaż i edycja (37),
- Uczenie maszynowe (12).

Respondenci najczęściej wskazywali operacje na plikach wideo jako czynność złożoną obliczeniowo. Jest to bardzo dobry przykład dla biblioteki, gdyż taką operację można bardzo łatwo zdublować na zewnętrznym serwisie. Należy jednak pamiętać, że pliki wideo są zwykle duże pod względem rozmiaru i ich przesłanie przez sieć może być również kosztowne, zarówno energetycznie jak i finansowo. Drugą wskazaną przez programistów operacją jest kompresja i dekompresja plików. W przypadku tego typu operacji biblioteka decyzyjna ma znikome zastosowanie z uwagi na cel kompresji i dekompresji. Zwykle te operacje wykonywane są przed wysłaniem lub po odebraniu danych przez sieć w celu minimalizacji liczby transferowanych bajtów. Zastosowanie do tego biblioteki wypaczało by kompletnie sens samej operacji, gdyż zewnętrzny serwis mógłby serwować dane bez kompresji. Jedynym skrajnym zastosowaniem byłoby wykorzystanie biblioteki do dekompresji danych pobieranych z niezależnego serwisu trzeciego który nie oferuje możliwości przesłania danych nieskompresowanych. W takim przypadku, uruchamiając aplikację na urządzeniu o bardzo niskiej mocy obliczeniowej można zastosować dekompresję danych na zewnętrznym serwisie i przesłać zdekompresowane dane z powrotem na urządzenie mobilne.

Pytanie czwarte miało sprawdzić, czy programistom aplikacji zdarzyło się zmienić architekturę aplikacji już po jej wdrożeniu. Aplikacje mobilne są zwykle produktami o bardzo krótkim czasie życia. Wynika to ze stale zmieniających się trendów w dziedzinie projektowania aplikacji, a także ze zmieniających się funkcjonalności samych systemów operacyjnych. Podczas gdy większość rozwiązań serwerowych zachowuje względną świeżość przez długie lata tak aplikacje mobilne wymagają regularnego odświeżania. W wielu przypadkach zmiana jest konieczna niedługo po publikacji oprogramowania i udostępnienia go użytkownikom. To opinie odbiorców końcowych są kluczowe do podjęcia takiej decyzji. Pytanie miało na celu potwierdzić lub obalić fakt zmiany architektury aplikacji już po jej publikacji w zakresie miejsca przeprowadzania obliczeń związanych z logiką biznesową. Wyniki przedstawia Wykres 12.

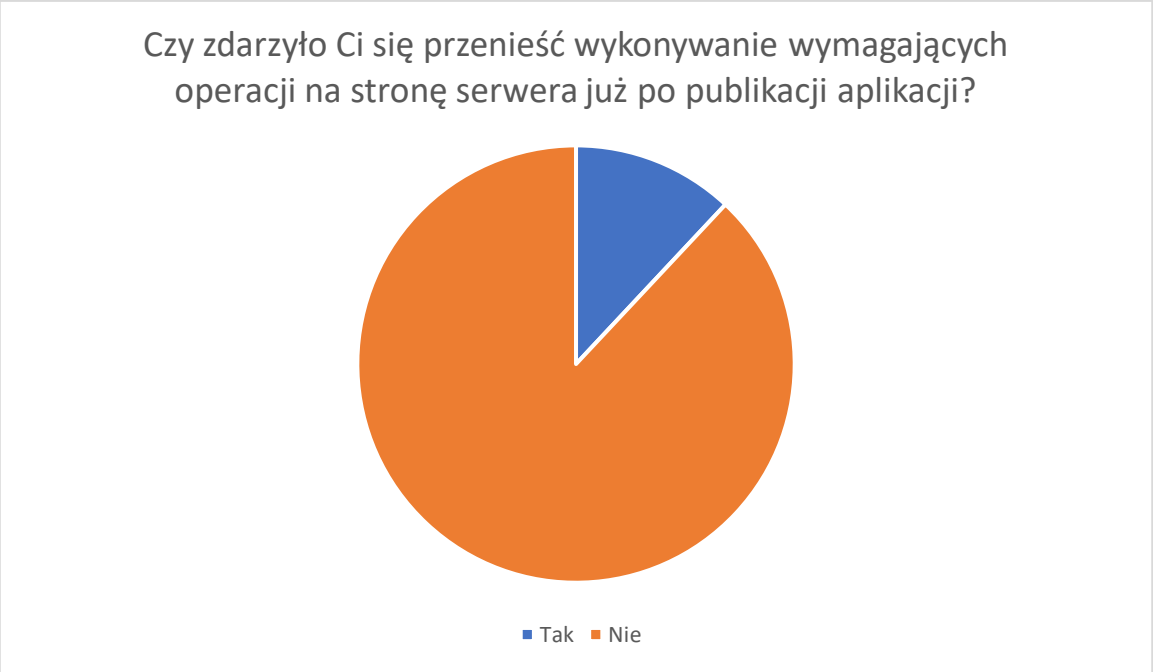
Pytanie: Czy zdarzyło Ci się przenieść wykonywanie wymagających operacji na stronę serwera już po publikacji aplikacji?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Tak - 26 / 11,98%,

B) Nie - 191 / 88,02%.



Wykres 12: Wyniki pytania 4 ankiety dla programistów

Pomimo przytłaczającej przewagi odpowiedzi negatywnej na zadawane pytanie, należy zauważyć, że zdarzają się przypadki przenoszenia istotnej części logiki biznesowej na stronę serwerową a tym samym całkowitą zmianę architektury aplikacji. Powodów takiego działania może być kilka. Począwszy od negatywnych opinii użytkowników, poprzez potrzebę zwiększenia elastyczności złożonych algorytmów przez możliwość szybkiej ich podmiany bez konieczności aktualizacji aplikacji mobilnych, po zmianę założeń architektonicznych mających na celu maksymalne rozproszenie rozwiązania i traktowanie aplikacji mobilnej jako prostego klienta wizualnego dla logiki biznesowej utrzymywanej na serwerze sieciowym. W kontekście proponowanej biblioteki decyzyjnej jest to o tyle ważne, że przy założeniu łatwości jej zastosowania wielu programistów mogłoby się zdecydować na jej wykorzystanie również w istniejących projektach, będących w fazie utrzymaniowej.

Kolejne pytanie wprost sprawdza czy programiści aplikacji stosowali podwójną implementację tej samej operacji – jedna na urządzeniu mobilnym a druga na serwerze zdalnym. Celem pytania było sprawdzenie czy doświadczenia zawodowe magistranta są odosobnione, czy też spotykają również innych programistów aplikacji Android. Stosowanie podwójnej implementacji jest kosztowne, gdyż należy zapewnić pełną kompatybilność obydwu algorytmów. Nie zawsze istnieje możliwość zastosowania dokładnie tego samego kodu po stronie aplikacji mobilnej i serwerowej, gdyż zastosowane w poszczególnych środowiskach technologie różnią się od siebie, chociażby wspieraną wersją języka

Java czy implementacją maszyny wirtualnej Java. Pytanie miało na celu zbadanie czy istniały jakiekolwiek przypadki stosowania podwójnej implementacji. Wyniki przedstawia Wykres 13.

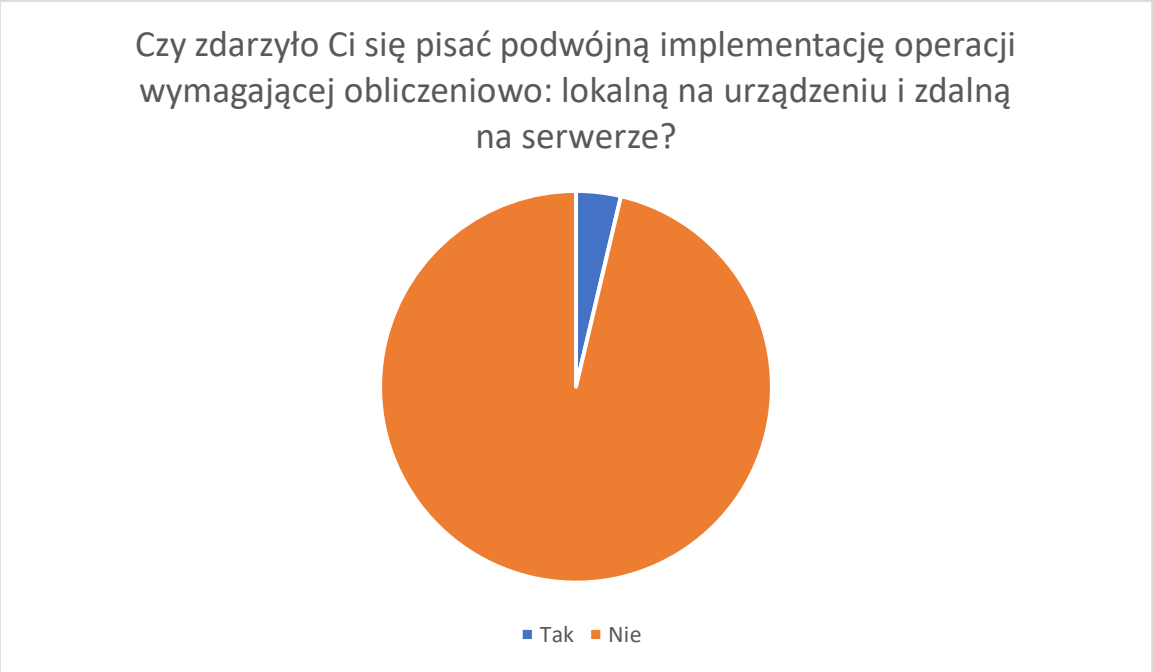
Pytanie: Czy zdarzyło Ci się pisać podwójną implementację operacji wymagającej obliczeniowo: lokalną na urządzeniu i zdalną na serwerze?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Tak - 8 / 3,69%,

B) Nie - 209 / 96,31%.



Wykres 13: Wyniki pytania 5 ankiety dla programistów

Zgodnie z oczekiwaniami większość programistów nigdy nie stosowało podwójnej implementacji. Wielu architektów uważa dublowanie logiki biznesowej w różnych środowiskach uruchomieniowych za antywzorzec projektowy i jest to uzasadnione, gdyż pociąga za sobą konieczność utrzymywania obydwu algorytmów oraz propagowania zmian pomiędzy nimi. Część odpowiedzi negatywnych może wynikać również z faktu, że programiści nie mieli potrzeby tworzenia podwójnej implementacji, ponieważ tworzone przez nich aplikacje nie wykonują żadnych skomplikowanych obliczeniowo operacji, co wynika z pytania 4. Jak pokazują wyniki pytania jednak dla części programistów stosowanie takiego antywzorca ma sens, gdyż widzą w tym inne, być może cenniejsze korzyści. Należą do nich głównie pozytywne opinie użytkowników. Należy bowiem pamiętać, że

aplikacje mobilne to produkty przeznaczone zwykle do bardzo szerokiego spektrum użytkowników. Pomimo, że aplikacja mobilna może być najmniejszym elementem skomplikowanego i wielowarstwowego systemu oprogramowania to jej jakość i responsywność mierzona ocenami na sklepie Google Play jest kluczowa dla powodzenia całego projektu. Przykładem porażki początkowej produktu może być rynek okazjonalnego współdzielenia samochodów, zwanego „carsharing" w Polsce. W roku 2017 pojawiły się dwie firmy oferujące wynajem krótkoterminowy pojazdów w ograniczonej strefie. Nowością była możliwość wypożyczenia samochodu jak i jego zwrot w dowolnym miejscu miasta objętym strefą wynajmu, bez pomocy osób trzecich, wykorzystując do tego celu aplikację mobilną zainstalowaną na smartfonie. Dwie firmy oferowały podobną klasę aut oraz podobne stawki za wynajem. Również liczba samochodów, co za tym idzie dostępność samej usługi, była bardzo zbliżona. Obie firmy korzystały z zaawansowanej technologii i posiadały złożone systemy zarządzania i śledzenia floty aut, wykrywania anomalii i analizy zachowań użytkowników. Jednak ilość pobrań aplikacji dostępowej, a co za tym idzie popularność usługi, jednej firmy była o rząd wielkości większa od drugiej. Według danych sklepu Google Play z końca roku 2017, aplikację firmy Traficar zainstalowało na swoich urządzeniach ponad 100 000 użytkowników [5] podczas gdy aplikacja Panek znalazła tylko ponad 10 000 odbiorców [6]. Ogromna różnica w ilości pobrań wynika po części z jakości samych aplikacji. Pod koniec 2017 roku aplikacja firmy Panek posiadała ocenę 2.7 w sklepie Google Play [6] a użytkownicy w komentarzach narzekali na przede wszystkim absurdalny pobór prądu przez aplikację oraz nieintuicyjny interfejs. Aplikacja Traficar mogła poszczycić się oceną 4.2 w sklepie Google Play [5] co powodowało lepsze pozycjonowanie w wynikach wyszukiwania i znacząco większą popularność w stosunku do konkurenta. Przykład pokazuje jak ważna jest jakość aplikacji mobilnej i jej odbiór przez użytkowników, pomimo że jest ona niewielkim elementem całego systemu informatycznego usługi.

Celem pytania 6 ankiety dla programistów było sprawdzenie czy autorzy aplikacji mobilnych podejmują decyzję o miejscu wykonywania złożonych obliczeń w sposób przemyślany, poparty głębszą analizą czy raczej w sposób arbitralny lub przypadkowy. Pytanie sugerowało pewne warianty odpowiedzi, ale zawierało też możliwość swobodnej wypowiedzi. Jako że wariantów odpowiedzi mogło być kilka, respondenci mieli możliwość zaznaczenia do trzech różnych opcji. Wyniki przedstawia Wykres 14.

Pytanie: Jakie czynniki wpływają na decyzję o miejscu wykonywania obliczeń: lokalnie lub zdalnie?

Typ pytania: wielokrotnego wyboru z możliwością odpowiedzi otwartej.

Odpowiedzi i wyniki (ilość odpowiedzi / procent):

A) Obecność serwera sieciowego - 101 / 46,54%,

B) Czas implementacji - 90 / 41,47%,

C) Założenia UX (User Experience) - 71 / 32,72%,

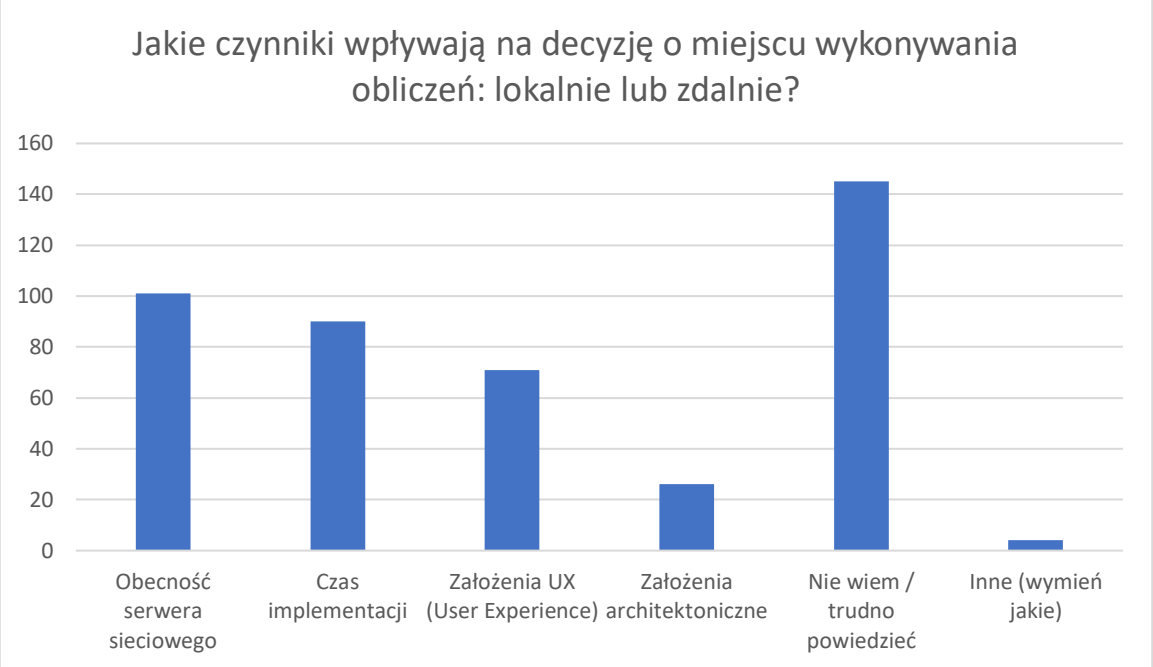
D) Założenia architektoniczne - 26 / 11,98%,

E) Nie wiem / trudno powiedzieć - 145 / 66,82%,

F) Inne (wymień jakie) - 4 / 1,84%.

Odpowiedzi otwarte (styl odpowiedzi zmodyfikowany przez autora):

- Zasoby osobowe, tj. kto akurat ma czas: czy osoba odpowiedzialna za stronę serwerową czy programista aplikacji mobilnych),
- Negatywne opinie użytkowników dotyczące powolnego działania aplikacji dietetycznej oraz przewaga urządzeń o niskiej mocy obliczeniowej na rynku docelowym,
- Funkcjonalność aplikacji, pozwalająca na swobodne korzystanie z niej w trybie bez połączenia internetowego wymusza przeniesienie niezbędnej logiki biznesowej do aplikacji mobilnej,
- (brak odpowiedzi).



Wykres 14: Wyniki pytania 6 ankiety dla programistów

Dwie trzecie respondentów wybrało opcję „nie wiem / trudno powiedzieć” co może wskazywać na brak zainteresowania materią lub brak możliwości wyboru odpowiedniej architektury z powodu narzuconych wymagań projektowych. Fakt dużej liczby odpowiedzi wskazującej na obecność serwera sieciowego jako czynnik wpływający na decyzję o miejscu wykonywania obliczeń nie powinien nikogo zaskoczyć, gdyż w przypadku aplikacji bez serwera taka decyzja nie ma racji bytu. Przypadkiem skrajnym byłoby tworzenie dedykowanej aplikacji serwerowej mającej na celu dublowanie funkcjonalności aplikacji mobilnej by poprawić odczucia i opinie użytkowników dotyczące oprogramowania na ich smartfonie. Prawie jedna trzecia badanych wskazała UX jako czynnik istotny dla decyzji o miejscu podejmowaniu obliczeń. Jest to informacja szczególnie istotna w kontekście proponowanej biblioteki decyzyjnej, gdyż jej głównym celem jest właśnie poprawienie odczuć użytkowników z używania aplikacji. Oznacza to, że zastosowanie biblioteki zaoszczędzi czas na etapie projektowania rozwiązania, ponieważ decyzja zostanie podjęta automatycznie na etapie wykonania programu a nie, jak w przypadku większości projektów, na etapie tworzenia architektury aplikacji.

Pytanie siódme miało na celu sprawdzenie czy programiści aplikacji mobilnych kiedykolwiek stosowali własne systemy, podobne do proponowanej biblioteki. Wyniki przedstawia Wykres 15.

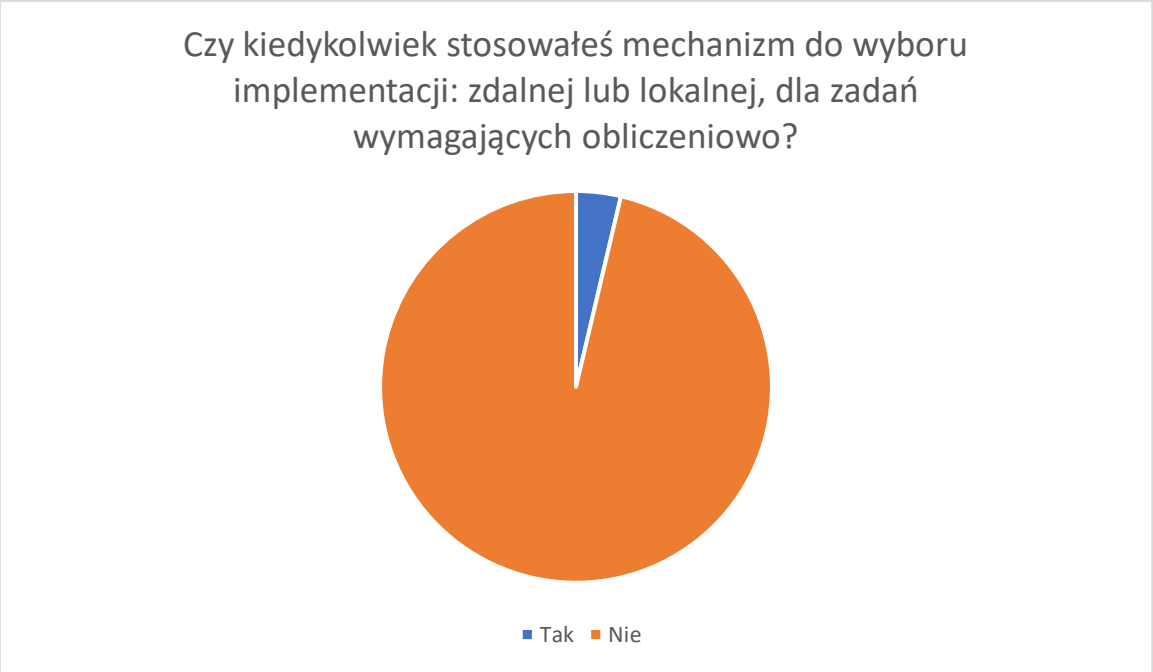
Pytanie: Czy kiedykolwiek stosowałeś mechanizm do wyboru implementacji: zdalnej lub lokalnej, dla zadań wymagających obliczeniowo?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

A) Tak - 8 / 3,69%,

B) Nie - 209 / 96,31%.



Wykres 15: Wyniki pytania 7 ankiety dla programistów

W odpowiedziach na pytanie siódme widać korelację z odpowiedziami na pytanie piąte kwestionariusza. Podobnie jak w pytaniu o stosowanie podwójnej implementacji, duża liczba negatywnych odpowiedzi może wynikać z braku konieczności stosowania takiego rozwiązania, ponieważ aplikacja nie wykonywała żadnych skomplikowanych operacji. Można z dużą dozą prawdopodobieństwa stwierdzić, że wszyscy programiści stosujący podwójną implementację logiki biznesowej wykorzystują również system decyzyjny wybierający implementację w zależności od jakichś kryteriów. Proponowana biblioteka byłaby idealnym, gotowym rozwiązaniem dla osób, które jako kryteria wybrały maksymalne zadowolenie użytkownika z aplikacji.

Pytanie ósme wprost sprawdza czy programiści aplikacji Android byliby zainteresowani skorzystaniem z proponowanej biblioteki. Wyniki przedstawia Wykres 16.

Pytanie: Czy byłbyś / byłabyś zainteresowany/a skorzystaniem z biblioteki, która podejmuje decyzję o miejscu wykonywania złożonych obliczeń (lokalnie / zdalnie) automatycznie w zależności od statystyk poprzednich wywołań, stanu baterii, temperatury urządzenia oraz innych czynników?

Typ pytania: jednokrotnego wyboru.

Odpowiedzi i wyniki (liczba odpowiedzi / procent):

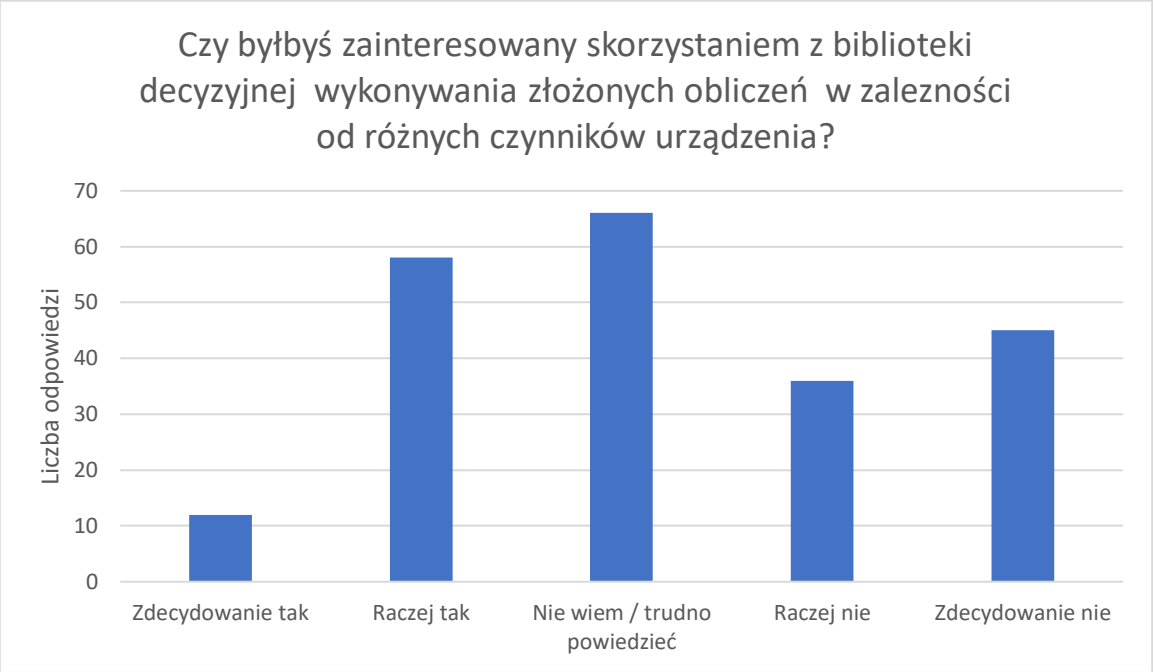
A) Zdecydowanie tak - 12 / 5,53%,

B) Raczej tak - 58 / 26,73%,

C) Nie wiem / trudno powiedzieć - 66 / 30,41%,

D) Raczej nie - 36 / 16,59%,

E) Zdecydowanie nie - 45 / 20,74%.



Wykres 16: Wyniki pytania 8 ankiety dla programistów

Wyniki pytania ósmego są podzielone na trzy zbliżone do siebie pod względem objętościowym grupy. Jedną grupę stanowią osoby, które raczej lub na pewno by się zdecydowały na skorzystanie z biblioteki. Prawdopodobnie są to osoby, które miały styczność z problemem niskiej wydajności niektórych urządzeń działających pod kontrolą systemu Android i chcące poprawić wrażenia użytkowników ze swoich aplikacją. Jeszcze liczniejszą grupę stanowią, które raczej lub na pewno by nie skorzystały z tej biblioteki. Prawdopodobnie nie widzą sensu stosowania tak rozbudowanego systemu decyzyjnego dla prostych aplikacji mobilnych. Nie bez znaczenia pozostaje fakt konieczności zdublowania logiki biznesowej na dwóch różnych środowiskach uruchomieniowych, który może być traktowany jako antywzorzec architektoniczny. Pozostali respondenci, a więc nieco ponad 30% ankietowanych, nie mają zdania na ten temat.

3.4.3 Podsumowanie ankiety

W ankiecie sumarycznie wzięło udział 248 internautów z czego 217 osób wypełniło cały kwestionariusz. Poziom odpowiedzi sięgnął 87,5% co jest zbliżonym wynikiem do poziomu odpowiedzi ankiety dla użytkowników. Tak wysoki poziom ankieta zawdzięcza swojej prostocie, gdyż większość

pytań kwestionariusza stanowiły pytania zamknięte, na które można szybko udzielić odpowiedzi. Pomimo braku dodatkowych motywatorów w postaci upominków czy gadżetów, liczba respondentów jest znacznie wyższa niż oczekiwana przez magistranta. Do wysokiego poziomu odpowiedzi mogła się przyczynić również solidarność zawodowa, ponieważ w liście wprowadzającym magistrant umieścił swój skrócony profil osobowy.

Średni czas wypełniania arkusza wynosił 7 minut i 54 sekundy co jest czasem znacząco większym niż w przypadku ankiety dla użytkowników. Wynika to z innej natury pytań, czasem wymagającej głębszego zastanowienia oraz sięgnięcia pamięcią do przeszłości w celu analizy dotychczasowych zrealizowanych projektów.

3.4.4 Wnioski z ankiety

Podobnie jak w przypadku ankiety dla użytkowników, w tym podpodrozdziale zostaną umieszczone wnioski w większości opisane w poszczególnych pytaniach.

Z ankiety wynika, że zdecydowana większość aplikacji mobilnych tworzona jest z myślą o łączeniu się z zewnętrznym serwisem. Ponad 93% respondentów stwierdziło, że ich programy często lub zawsze łączą się z API webowym. Jest to popularna architektura aplikacyjna, gdyż pozwala na skupienie logiki biznesowej na serwerze i traktowanie aplikacji mobilnej jako lekkiego klienta do wyświetlania danych. Nie zawsze jednak taka architektura ma zastosowanie, szczególnie jeżeli w wymaganiach klienta jest mowa o konieczności działania aplikacji w trybie bez dostępu do sieci internetowej. Możliwość pracy offline jest własnością aplikacji mobilnych, która wyróżnia je na tle innych rozwiązań, takich jak AWD (Adaptive Web Design) czy RWD (Responsive Web Design) czyli na tle stron internetowych optymalizowanych pod kątem małych ekranów smartfonów sterowanych dotykowo, ponieważ strony te nie mogą działać bez dostępu do Internetu. Dostęp do takich stron jest niemożliwy w przypadku podróżowania samolotem czy przebywania za granicą z powodu wysokich cen transferu danych w roamingu międzynarodowym. Dlatego programiści tworząc aplikację rozproszoną muszą wziąć pod uwagę użyteczność aplikacji w trybie bez dostępu do Internetu, co czasem wiąże się z dublowaniem logiki biznesowej w aplikacji na Androida i aplikacji backendowej.

Ankieta potwierdza też tezę, że smartfony nie służą jedynie do prostych działań, ale także do wykonywania złożonych obliczeniowo operacji. Ponad 97% respondentów potwierdza fakt implementowania takich operacji w swoich programach. Biorąc pod uwagę rosnącą wydajność urządzeń przenośnych można spodziewać się, że coraz więcej aplikacji zarezerwowanych do tej pory dla komputerów PC zostanie przeniesiona na urządzenia mobilne. Należy jednak pamiętać, że sama wydajność urządzenia to nie wszystko. Trzeba zwrócić uwagę na idący w parze ze wzrostem wydajności samych urządzeń wzrost zapotrzebowania na energię elektryczną. Technologia baterii nie rozwija się

równie szybko jak procesy technologiczne procesorów, pamięci czy układów graficznych. Jak pokazała ankieta dla użytkowników to właśnie czas pracy na baterii jest kluczowy podczas korzystania z urządzeń przenośnych. Istnieje zatem uzasadniony trend przenoszenia niektórych operacji do centrów obliczeniowych.

Programiści aplikacji mobilnych starają się dbać o swoich użytkowników, ponieważ wiedzą, że to od jakości aplikacji mobilnej w dużej mierze zależy sukces danego rozwiązania informatycznego. Smartfon jest elementem styku pomiędzy końcowym odbiorcą a resztą systemu. To aplikacja mobilna odpowiada za prezentację funkcjonalności oraz sterowanie działaniem rozwiązania, dlatego tak ważne jest dopracowanie jej pod względem UX i zadbanie o dobre samopoczucie użytkowników podczas jej używania. To sprawia, że część programistów decyduje się na znaczące zmiany w architekturze całego systemu by sprostać potrzebom użytkowników i dostarczyć im satysfakcjonujący produkt nawet już po jego wdrożeniu.

Pytając programistów wprost o chęć wykorzystania proponowanej biblioteki decyzyjnej zdania są podzielone. Budujące jest to, że ponad 30% badanych raczej lub na pewno spróbowałby wykorzystać takie rozwiązanie w swojej aplikacji. Zakładając łatwość integracji biblioteki z nowymi lub istniejącymi aplikacjami istnieje duże prawdopodobieństwo, że biblioteka stanie się popularna, szczególnie w przypadku aplikacji, dla których krytyczne jest maksymalizowanie zadowolenia użytkowników.

4. Analiza parametrów wpływających na decyzję

Z ankiety można wywnioskować kilka parametrów wpływających na zadowolenie użytkowników z korzystania z aplikacji mobilnej. By dogłębnie zbadać jak faktycznie złożone obliczenia wpływają na te parametry, został przeprowadzony eksperyment. Celem eksperymentu było stwierdzenie czy złożone operacje obliczeniowo w jakikolwiek sposób oddziałują na parametry wpływające na zadowolenie użytkowników. Ponadto, eksperyment miał pomóc w określeniu wartości granicznych parametrów, które mogłoby wpływać na decyzję o miejscu wykonania obliczeń.

4.1. Przebieg eksperymentu

W celu wykonania eksperymentu zostało stworzone proste oprogramowanie testowe. Była to aplikacja mobilna Android wykonująca w pętli operację skalowania i przycinania pliku wideo. Jest to operacja bardzo często wykonywana przez użytkowników przy okazji dodawania filmów na portale społecznościowe, dlatego też została wybrana jako reprezentatywna. Eksperyment został podzielony na dwie fazy:

1. Faza pierwsza polegała na wykonywaniu obliczeń lokalnie,
2. Faza druga polegała na wykonywaniu obliczeń na serwerze oraz transmisji danych poprzez sieć Internet.

Przed rozpoczęciem każdej fazy urządzenie testowe zostało naładowane do pełna a następnie odłączone od źródła zasilania. Ponadto, przed rozpoczęciem każdej fazy urządzenie zostało wyczyszczone ze wszelkich danych oraz aplikacji, tj. przywrócone do stanu fabrycznego. Jedyne operacje wykonywane przed rozpoczęciem fazy niezwiązane bezpośrednio z eksperymentem wynikały z konieczności konfiguracji wstępnej urządzenia, tj. skonfigurowanie sieci Wi-Fi, skonfigurowanie sieci komórkowej oraz instalacja aplikacji testowej. Wynikiem każdej fazy był plik CSV zawierający nagłówek porządkujący dane oraz kolejne wiersze reprezentujące pojedynczą iterację programu. W każdym wierszu zostały zapisane wartości parametrów zebrane z urządzenia mobilnego przy pomocy oprogramowania Android.

Podczas fazy pierwszej wykonywane były następujące czynności:

1. Stworzenie pliku wynikowego z odpowiednimi nagłówkami,
2. Zebranie wartości początkowych parametrów i wpisanie ich do pliku,
3. Zablokowanie wyłączenia ekranu, ustawienie domyślnej jasności wyświetlacza,
4. Wykonanie operacji obliczeniowej,

5. Zebranie aktualnych wartości parametrów,
6. Wyłączenie ekranu,
7. Odczekanie ustalonego limitu czasowego (10 sekund),
8. Skok do punktu 3.

Podczas fazy drugiej wykonywane były następujące czynności:

1. Stworzenie pliku wynikowego z odpowiednimi nagłówkami,
2. Zebranie wartości początkowych parametrów i wpisanie ich do pliku,
3. Zablokowanie wyłączenia ekranu, ustawienie domyślnej jasności wyświetlacza,
4. Wysłanie danych wejściowych do serwera poprzez sieć Wi-Fi,
5. Oczekiwanie na zakończenie operacji,
6. Odebranie danych wyjściowych z serwera poprzez sieć Wi-Fi oraz zapis pliku wynikowego w pamięci lokalnej urządzenia,
7. Zebranie aktualnych wartości parametrów,
8. Wyłączenie ekranu,
9. Odczekanie ustalonego limitu czasowego (10 sekund),
10. Skok do punktu 3.

Każda faza była kończona poprzez całkowite rozładowanie urządzenia mobilnego. Po każdej fazie następowało pełne ładowanie urządzenia, zebranie pliku wynikowego z pamięci smartfonu oraz następowało rozpoczęcie kolejnej fazy.

Podczas eksperymentu ekran był włączony podczas wykonywanej operacji oraz wyłączony podczas oczekiwania na kolejną iterację. Jest to zabieg celowy mający na celu maksymalne odwzorowanie realnego schematu użycia aplikacji przez użytkowników. Testy na popularnych portalach społecznościowych potwierdziły, że podczas obróbki filmu video użytkownik zmuszony jest na oczekiwanie na zakończenie operacji. Same obliczenia nie wykonują się w tle, jednak ekran urządzenia pozostaje włączony na bieżąco informując użytkownika o aktualnym postępie pracy. Włączony wyświetlacz ma bardzo duży wpływ na czas pracy urządzenia na baterii, toteż musiał zostać uwzględniony w eksperymencie.

4.2. Urządzenia testowe

Eksperyment został przeprowadzony na kilku urządzeniach mobilnych celem zróżnicowania wyników oraz potwierdzenia lub obalenia tezy, że decyzja o miejscu wykonywania obliczeń powinna być uzależniona od klasy urządzenia. Dobór urządzeń był ograniczony możliwościami finansowymi magistranta, gdyż dysponował on ograniczoną liczbą urządzeń. Charakter pracy zawodowej magistranta pozwolił na częściowe rozszerzenie liczby urządzeń testowych. Podczas doboru urządzeń dołożono najwyższych starań by maksymalnie zróżnicować typy urządzeń. Kryteria jakimi się kierowano to wiek, rozmiar ekranu oraz segment urządzenia.

4.2.1 Segment budżetowy

W segmencie budżetowym przetestowano następujące urządzenia:

Samsung Galaxy Mini II

Jest to najtańsze urządzenie wykorzystane w eksperymencie. Pomimo swojego wieku jest nadal popularne z uwagi na bardzo niską cenę zakupu oraz liczne promocje operatorów na ten model. Jego specyfikacja znajduje w Tabela 1.

Tabela 1: Specyfikacja techniczna urządzenia Samsung Galaxy Mini II [8]

Procesor	Qualcomm MSM7227A Zegar procesora: 800 MHz Liczba rdzeni: 1 GPU: Adreno 200 enhanced
Ekran	TFT 256k kolorów 320 x 480 px (3.27") 176 pp
Pamięć RAM	512 MB
Bateria	Li-Ion 1300 mAh

Xiaomi Redmi 4A

Urządzenie chińskiego producenta jest bardzo popularne w Polsce, ponieważ oferuje bardzo dobry stosunek jakości do ceny. Jest to również jeden z pierwszych modeli marki Xiaomi sprzedawany w oficjalnej dystrybucji w Polsce. Urządzenie zostało zaprezentowane pod koniec 2016 roku. Specyfikację urządzenia przedstawia Tabela 2.

Tabela 2: Specyfikacja techniczna urządzenia Xiaomi Redmi 4A [7]

Procesor	Qualcomm Snapdragon 425 8917 Zegar procesora: 1.40 GHz Liczba rdzeni: 4 GPU: Adreno 308 @500 MHz
Ekran	TFT 16M kolorów 720 x 1280 px (5.00") 294 ppi
Pamięć RAM	2 GB
Bateria	Li-Ion 3120 mAh

4.2.2 Segment średni

W segmencie średnim przetestowano następujące urządzenia:

Samsung Galaxy J5 2017

To urządzenie charakteryzuje się wydajnym ekranem AMOLED, zarówno pod względem jakości obrazu, nasycenia kolorów jak i oszczędności energii. Smartfon jest bardzo popularny na polskim rynku ze względu na uznanie marki Samsung i przystępną cenę. Jego specyfikację przedstawia Tabela 3.

Tabela 3: Specyfikacja techniczna urządzenia Samsung Galaxy J5 2017 [9]

Procesor	Samsung Exynos 7870 Zegar procesora: 1.60 GHz Liczba rdzeni: 8 GPU: ARM Mali-T830 MP1 @700 MHz
Ekran	Super AMOLED 16M kolorów 720 x 1280 px (5.20") 282 ppi
Pamięć RAM	2 GB
Bateria	Li-Ion 3000 mAh

4.2.3 Segment flagowy

W segmencie telefonów flagowych przetestowano następujące urządzenia:

Sony Xperia Z3

Smartfon japońskiego producenta miał swoją premierę pod koniec roku 2014. W świetle dzisiejszych rywali jego parametry plasują go wśród urządzeń klasy średniej niemniej jednak w momencie premiery było to urządzenie z najwyższej półki. Ponadczasowy wygląd i bardzo dobra wytrzymałość sprawiają, że nadal korzysta z niego wielu użytkowników. Jego specyfikację przedstawia Tabela 4.

Tabela 4: Specyfikacja techniczna urządzenia Sony Xperia Z3 [10]

Procesor	Qualcomm Snapdragon 801 8974AC Zegar procesora: 2.50 GHz Liczba rdzeni: 4 GPU: Adreno 330 @578 MHz

Ekran	IPS TFT 16M kolorów 1080 x 1920 px (5.20") 424 ppi
Pamięć RAM	3 GB
Bateria	Li-Ion 3100 mAh

Samsung Galaxy S8

Wprowadzony na rynek w połowie 2017 roku wyznaczył nowe trendy w dziedzinie wyglądu oraz wydajności smartfonów. Samsung postanowił zerwać z dotychczasową stylistyką i postawić na nowoczesny, bezramkowy wygląd. Jego specyfikację przedstawia Tabela 5.

Tabela 5: Specyfikacja techniczna urządzenia Samsung Galaxy S8 [11]

Procesor	Samsung Exynos 8895 Zegar procesora: 2.30 GHz Liczba rdzeni: 8 GPU: ARM Mali-G71 @550 MHz
Ekran	Super AMOLED 16M kolorów 1440 x 2960 px (5.80")
Pamięć RAM	4 GB
Bateria	Li-Ion 3000 mAh

4.3. Wyniki eksperymentu

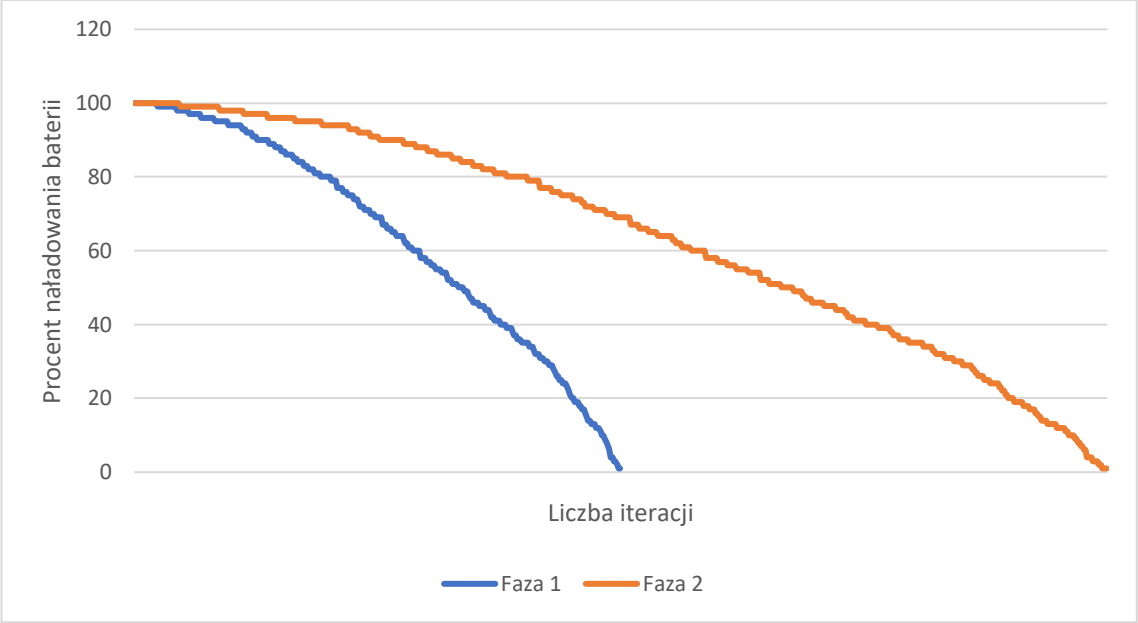
Obydwie fazy eksperymentu zostały przeprowadzone na wszystkich wymienionych powyżej urządzeniach, pomimo że na wielu z nich wynik był łatwy do przewidzenia. Poniżej zaprezentowano wyniki eksperymentu oraz przedstawiono wnioski.

4.3.1 Analiza baterii

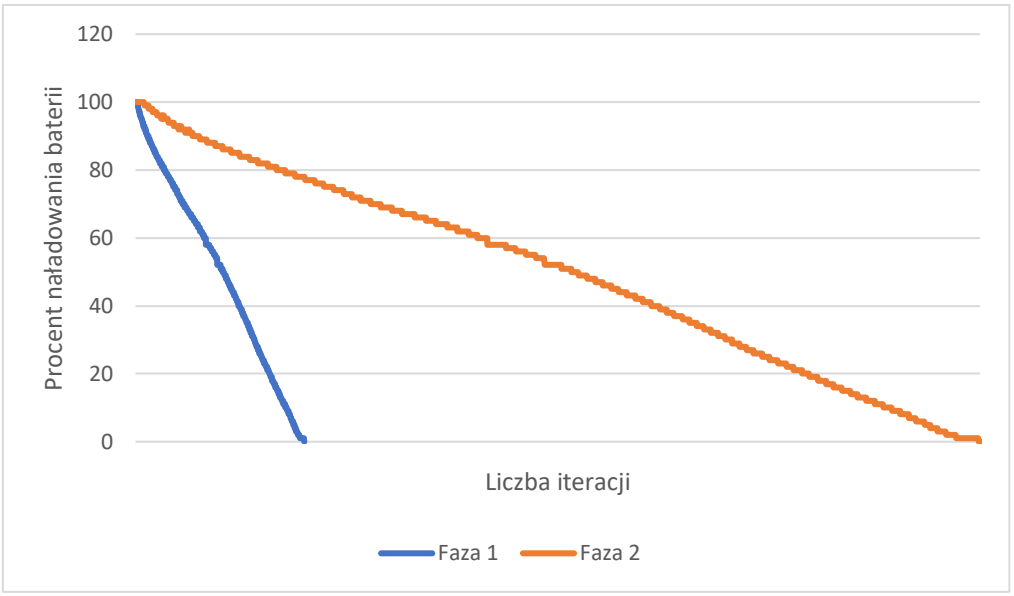
Stan baterii jest kluczowy w podjęciu decyzji o miejscu wykonania obliczeń. Jak wykazała ankieta skierowana do użytkowników to właśnie krótki czas pracy na baterii jest najbardziej irytującym czynnikiem podczas obcowania z urządzeniami mobilnymi.

Celem eksperymentu jest dobranie optymalnego punktu stanu akumulatora, w którym należy za wszelką cenę oszczędzać energię elektryczną. Należało zatem zbadać czy zależność pomiędzy ilością wykonanych operacji a stanem baterii jest liniowa. Jeżeli nie, to można wyznaczyć optymalny punkt przełączenia strategii na tryb oszczędzania energii by zmaksymalizować zadowolenie użytkownika.

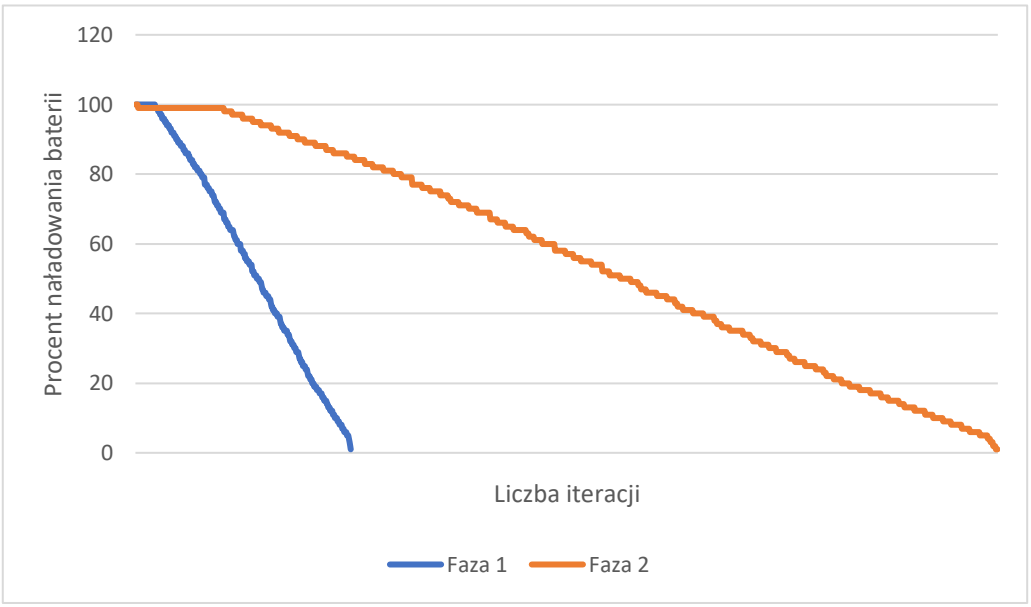
Poniżej przedstawiono wyniki eksperymentu (Wykres 17, Wykres 18, Wykres 19, Wykres 20, Wykres 21):



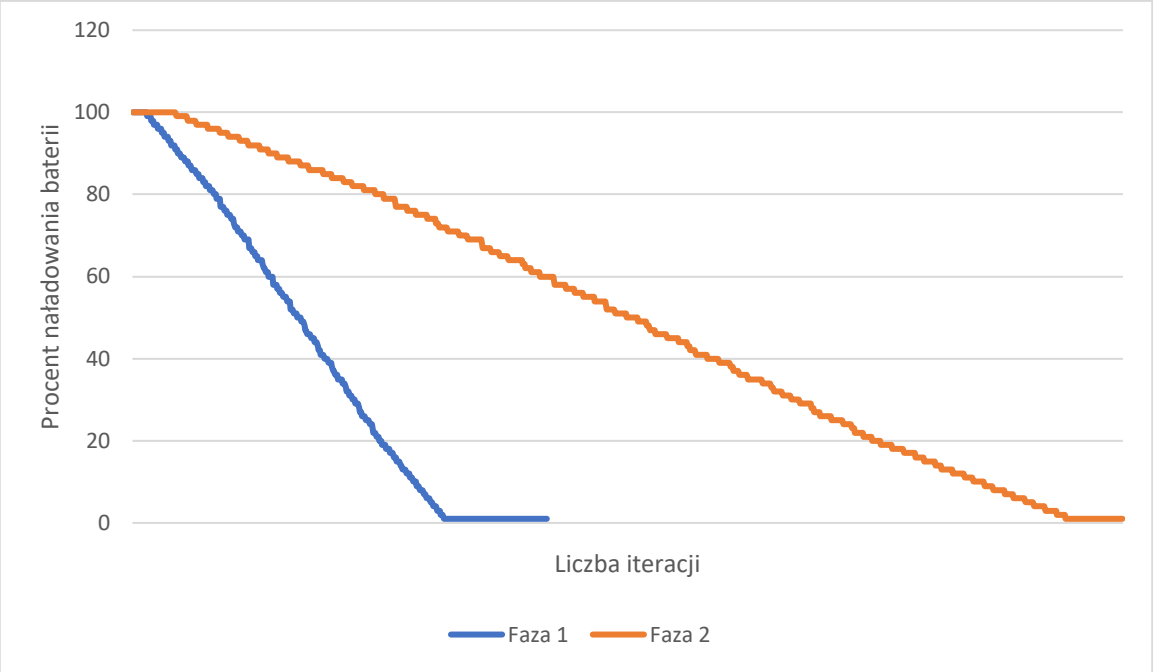
Wykres 17: Krzywa rozładowywania akumulatora w urządzeniu Samsung Galaxy Mini II



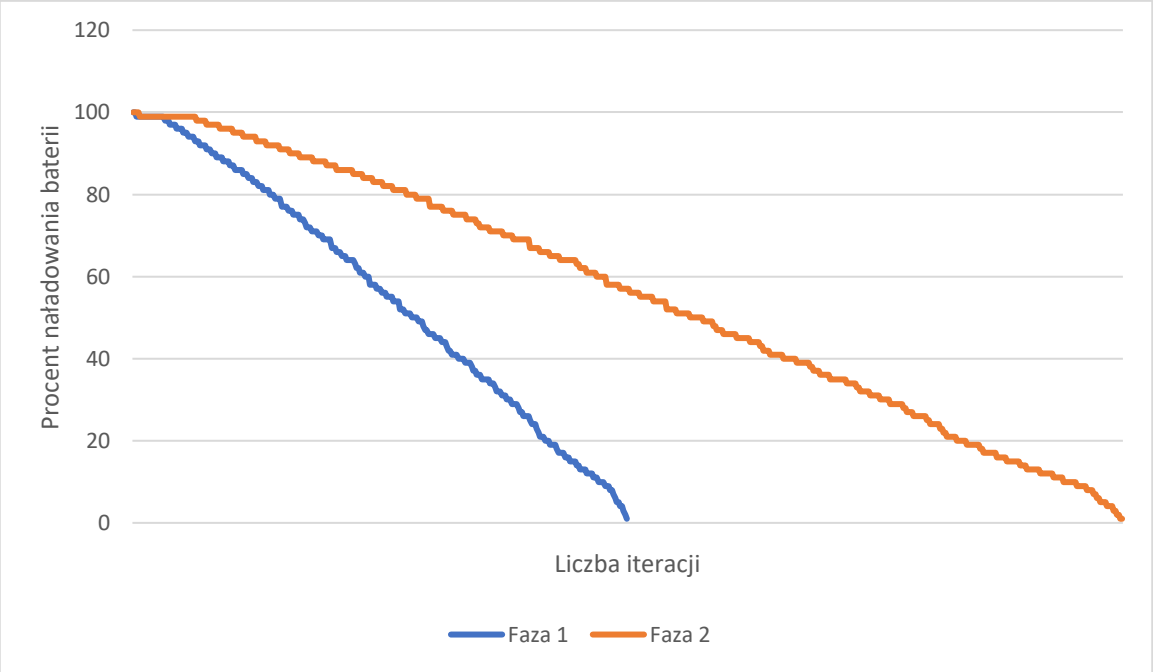
Wykres 18: Krzywa rozładowywania akumulatora w urządzeniu Xiaomi Redmi 4A



Wykres 19: Krzywa rozładowywania akumulatora w urządzeniu Samsung J5



Wykres 20: Krzywa rozładowywania akumulatora w urządzeniu Sony Xperia Z3



Wykres 21: Krzywa rozładowywania akumulatora w urządzeniu Samsung S8

Poza najstarszym urządzeniem, wszystkie testowane egzemplarze charakteryzuje liniowy przebieg krzywej rozładowywania akumulatora. Samsung Galaxy Mini II to telefon, który miał swoją

premierę w roku 2012, czyli 6 lat przed datą napisania niniejszej pracy. Na rynku technologicznym jest to bardzo długi czas, nic więc dziwnego, że producenci udoskonalili swoje algorytmy obliczania pozostałego ładunku w akumulatorze. Można zatem przyjąć, że akumulator rozładowuje się liniowo. Nie da się zatem wyznaczyć matematycznie optymalnego punktu progowego dla zmiany decyzji o miejscu wykonywania obliczeń.

Nie ma natomiast wątpliwości co do faktu, że to wykonywanie obliczeń zdalnie na serwerze jest dużo bardziej efektywne energetycznie niż wykonywanie ich lokalnie nie urządzeniu. Transmisja danych poprzez sieć Internet nie potrzebuje tak dużej mocy obliczeniowej procesora zatem jest zdecydowanie bardziej wydajne energetycznie, co potwierdzają wyniki eksperymentu.

Z punktu widzenia oszczędzania energii baterii biblioteka powinna zawsze podejmować decyzję o zdalnym wykonywaniu obliczeń.

4.3.2 Analiza zużycia danych

Analiza zużytych danych jest kluczowa dla analizy kosztowej dla użytkownika końcowego. Usługi cen telekomunikacyjnych, w tym Internetu mobilnego, ciągle spadają. Według raportu przygotowanego przez Komisję Europejską dotyczącego cen Internetu mobilnego Polscy operatorzy oferują usługi nawet 70% niż średnia z 28 krajów Unii Europejskiej [12]. Pomimo tego znaczna część użytkowników nadal kontroluje koszty związane z Internetem oraz monitoruje zużycie danych przez aplikacje.

W przypadku fazy pierwszej eksperymentu wynik jest trywialny. Wyniki ze wszystkich urządzeń wskazały zerowe zużycie transferu danych. Gdy wszystkie obliczenia wykonywane są lokalnie na urządzeniu nie jest wymagane żadne połączenie internetowe. Jest to ważna informacja w kontekście zapewnienia maksymalnej dostępności aplikacji w różnych warunkach. Biblioteka decyzyjna musi uwzględniać fakt istnienia aktywnego połączenia internetowego i w przypadku jego braku wykonać operację lokalnie na urządzeniu, bez względu na pozostałe czynniki.

W przypadku fazy drugiej wynik zależy od ilości przesyłanych danych. W przeprowadzonym eksperymencie suma przesłanych i odebranych danych podczas jednej iteracji wynosi około 24 megabajtów. W przypadku połączenia Wi-Fi koszt przesłania takiej ilości danych jest zerowy, jednak przy połączeniu komórkowym nawet tak stosunkowo niewielka ilość danych może być uciążliwa dla użytkownika.

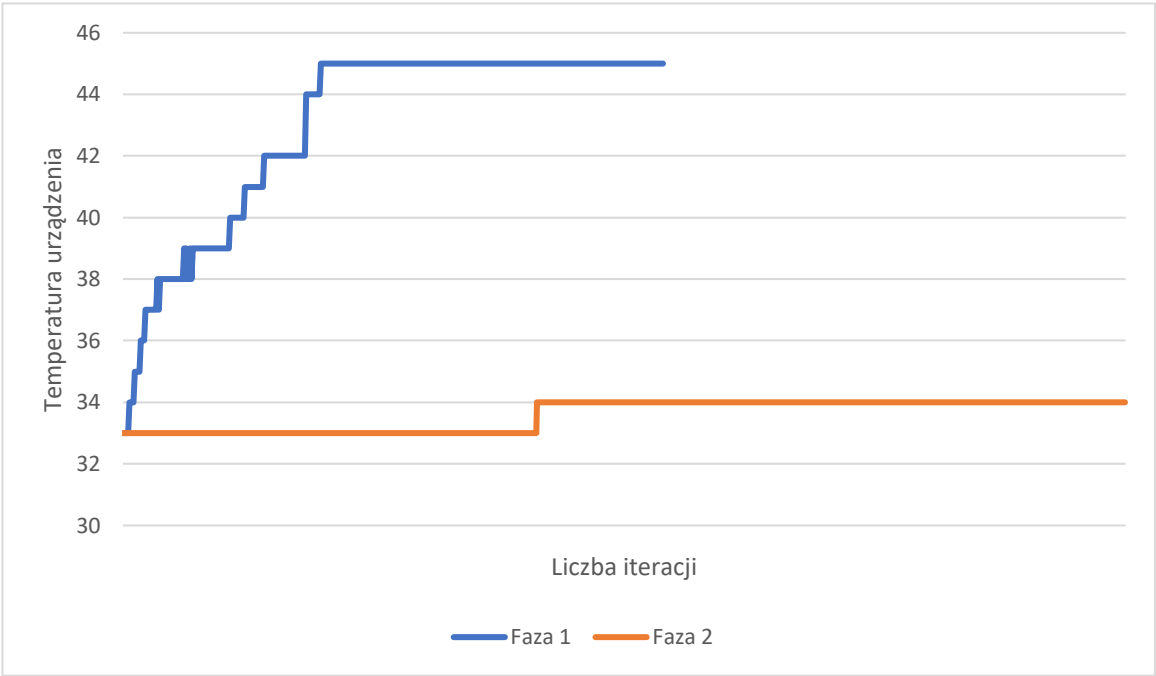
Z punktu widzenia biblioteki decyzyjnej, w zakresie zużytych danych preferowane jest wykonywanie operacji lokalnie na urządzeniu na równi z wykonaniem zdalnym w przypadku połączenia poprzez Wi-Fi. Przy połączeniu komórkowym należy uzależnić decyzję od ilości danych do wysłania

oraz do odebrania. Istnieje możliwość zastosowania rozkładu decyzji by zapewnić najwyższą jakość dla użytkownika przy zbalansowanym użyciu danych.

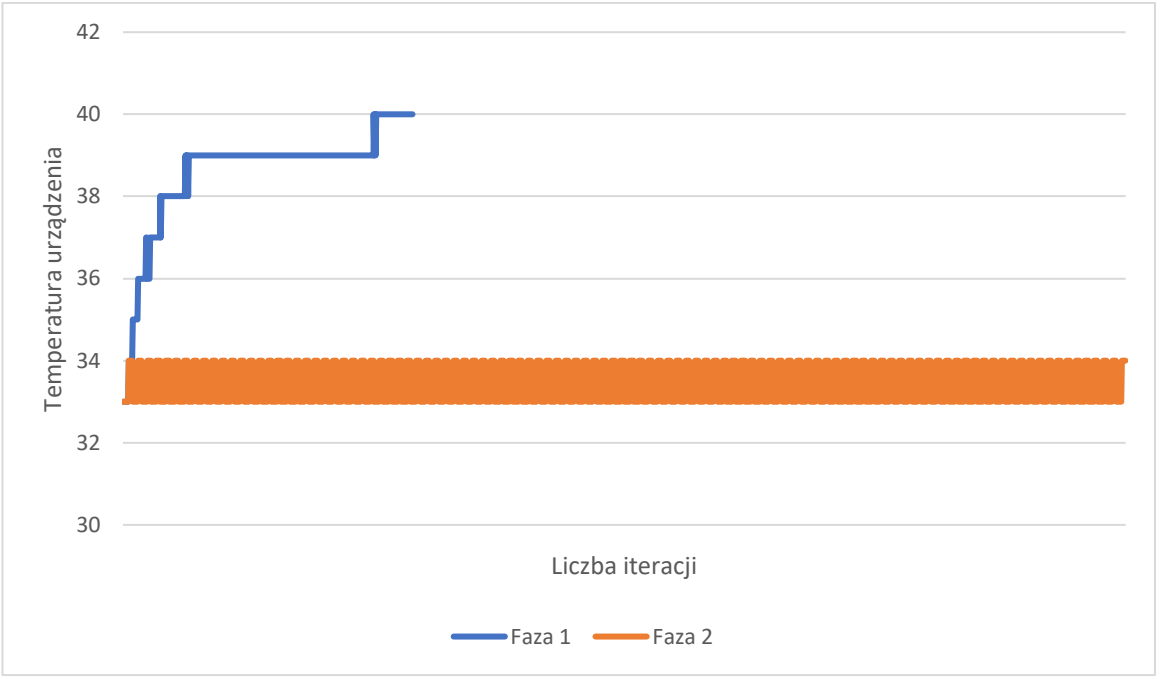
4.3.3 Analiza temperatury urządzenia

Nagrzewanie się urządzenia jest normalnym objawem występującym podczas korzystania ze smartfonu. Jednak podczas trzymania urządzenia w ręku, fakt nadmiernego ogrzewania urządzenia może spowodować dyskomfort u użytkowników. Według opinii internautów [13] telefon jest uznawany za „ciepły" gdy jego temperatura osiąga 34-35 stopni Celsjusza. Gdy temperatura przekracza 37 stopni użytkownicy wyraźnie odczuwają dyskomfort z jego użytkowania i podejrzewają nieprawidłową pracę urządzenia. W takim przypadku analizują aplikacje, które były uruchomione podczas przegrzewania telefonu i zazwyczaj usuwają je. Należy pamiętać, że operacje złożone obliczeniowo potrzebują pełnej mocy procesora, który jest głównym generatorem ciepła w urządzeniu mobilnym.

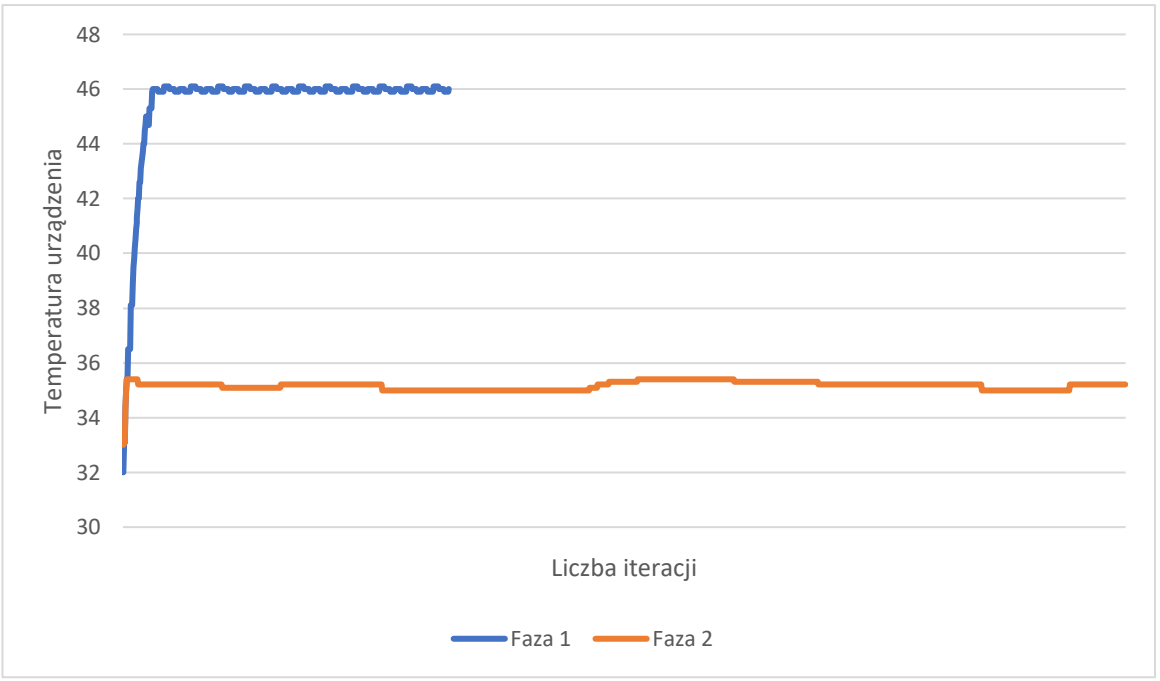
Wyniki eksperymentu przedstawiono na wykresach (Wykres 22, Wykres 23, Wykres 24, Wykres 25, Wykres 26):



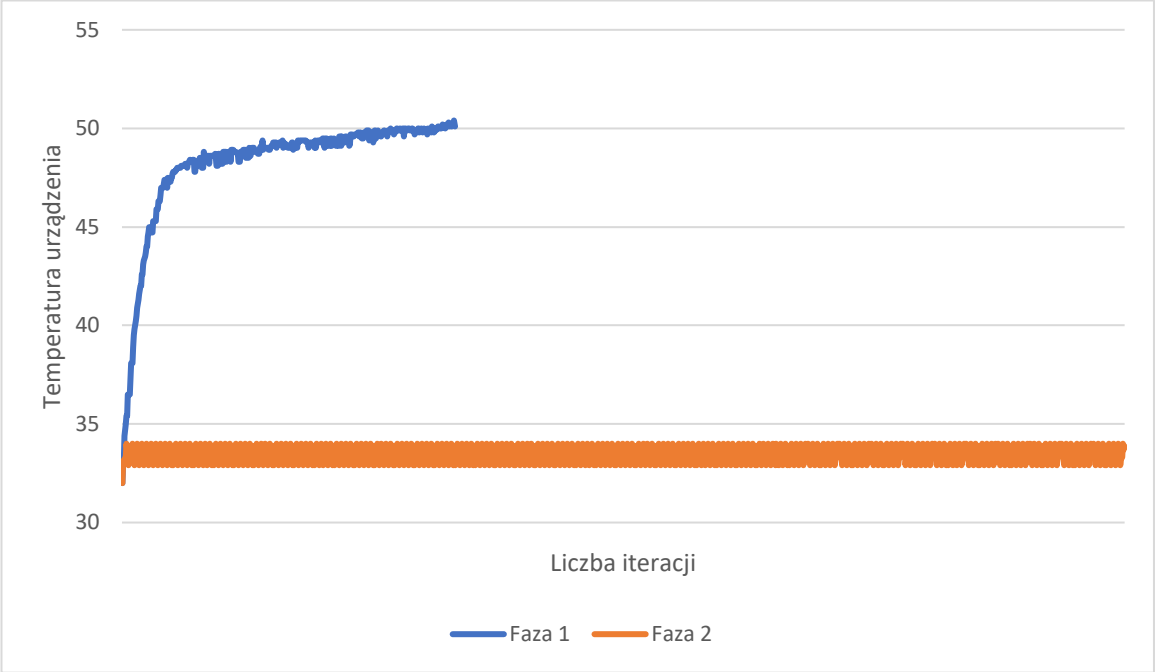
Wykres 22: Analiza temperatury urządzenia Samsung Galaxy Mini II podczas wykonywania eksperymentu



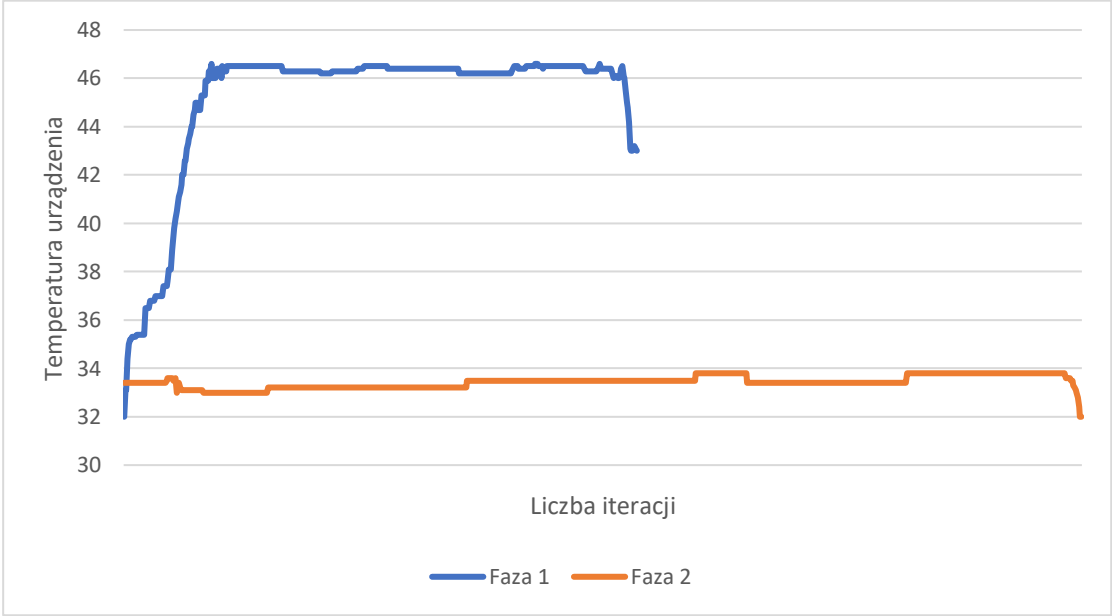
Wykres 23: Analiza temperatury urządzenia Xiaomi Redmi 4A podczas wykonywania eksperymentu



Wykres 24: Analiza temperatury urządzenia Samsung J5 podczas wykonywania eksperymentu



Wykres 25: Analiza temperatury urządzenia Sony Xperia Z3 podczas wykonywania eksperymentu



Wykres 26: Analiza temperatury urządzenia Samsung S8 podczas wykonywania eksperymentu

W fazie pierwszej widoczny jest zauważalny wzrost temperatury na samym początku trwania eksperymentu. Urządzenie stało się ciepłe już po kilkudziesięciu operacjach i w sposób liniowy osiągnęło temperaturę nawet 50 stopni Celsjusza. Taka temperatura jest z pewnością nieakceptowalna przez użytkownika i należy unikać rozgrzewania urządzenia do tego stopnia.

W fazie drugiej temperatura jest niemal stała. Zmiany są niewielkie i są raczej związane ze zmianą temperatury otoczenia niż z rozgrzewaniem urządzenia. Jedynie na początku widać minimalny wzrost temperatury, spowodowany prawdopodobnie stale włączonym wyświetlaczem i jego podświetleniem.

Wynik eksperymentu potwierdza, że operacje złożone obliczeniowo mają ogromny wpływ na temperaturę urządzenia i należy wziąć ten czynnik pod uwagę podczas podejmowania decyzji o miejscu wykonania obliczeń. Natomiast transmisja danych zupełnie nie wpływa na temperaturę urządzenia. Z punktu widzenia temperatury biblioteka powinna preferować zdalne wykonywanie obliczeń.

4.3.4 Analiza czasu trwania operacji

Bardzo ważnym z punktu widzenia użytkownika aspektem pracy aplikacji jest czas oczekiwania na długotrwałe operacje. Użytkownicy są świadomi, że pewne operacje muszą potrwać, jednak ich cierpliwość zwykle kończy się po około 10 sekundach. Granica tolerancji przesuwa się, gdy użytkownik widzi wyraźny wskaźnik progresu operacji, na przykład rosnący pasek postępu czy procent ukończenia. Wskaźniki aktywności, reprezentowane zazwyczaj przez obracające się pierścienie, sprawiają, że użytkownik nie wyłącza aplikacji, ponieważ widzi, że aplikacja się nie zawiesiła tylko czeka na zakończenie operacji w tle.

Czas trwania operacji złożonej obliczeniowo zależy głównie od dwóch czynników:

1. Od rodzaju samej operacji i jej złożoności,
2. Od sprzętu, na którym została uruchomiona.

Urządzenia mobilne są bardzo zróżnicowane pod względem użytych podzespołów obliczeniowych. Ponadto, rozwój technologii mikroprocesorowej sprawia, że coraz wydajniejsze układy trafiają nawet do najtańszych modeli. Kolejne generacje urządzeń mobilne przejmują układy obliczeniowe z wyższych segmentów. Oznacza to, że kilkuletni smartfon z segmentu flagowego dysponuje podobną mocą obliczeniową co dzisiejsze urządzenie klasy budżetowej. Zastosowanie biblioteki decyzyjnej pozwoli zoptymalizować czas wykonania operacji, gdy podzespoły urządzenia okażą się niewystarczające do zapewnienia sprawnego działania aplikacji.

Podczas fazy pierwszej eksperymentu czas wykonania operacji nie jest stały. Na początku jest znacznie krótszy niż w późniejszych iteracjach. Można też zauważyć korelację czasu wykonania zadania z temperaturą urządzenia. Widać, że wraz ze wzrostem temperatury urządzenia, wydłuża się czas

potrzebny na zakończenie obliczeń. Związane jest to z mechanizmem „Throttlingu", czyli obniżania wydajności podzespołów obliczeniowych w celu ochrony komponentów przed przegrzaniem. Jest to bardzo ważne z punktu widzenia implementacji strategii biblioteki decyzyjnej. W przypadku wyższej temperatury urządzenia należy spodziewać się, że czas wykonania operacji będzie dłuższy i należy wziąć to pod uwagę podczas podejmowania decyzji.

W drugiej fazie eksperymentu czasy wykonania były relatywnie stałe. By nie zaciemnić wyników eksperymentu, urządzenia były podłączone do stabilnej sieci Wi-Fi. Największym czynnikiem wpływającym na czas wykonania na serwerze jest szybkość transmisji sieciowej i wielkość przesyłanych danych.

Średnie czasy wykonania operacji pokazuje Tabela 6. Czas wykonania operacji zdalnie zawiera także czas potrzebny na transmisję danych między serwerem i urządzeniem testowym.

Tabela 6: Zestawienie średnich czasów wykonania operacji

Urządzenie	**Średni czas lokalnie [s]**	**Średni czas zdalnie [s]** (uwzględnia wysyłanie / odbieranie danych)
Samsung Galaxy Mini II	46,2	19,12
Xiaomi Redmi 4A	30,74	13,85
Samsung J5	17,06	18,9
Sony Xperia Z3	30,2	18,22
Samsung S8	11,04	13,2
MacBook Pro 15 2017	1,32	nie dotyczy

Wyniki eksperymentu wskazują na duże zróżnicowanie czasu wykonania operacji, zarówno przy wykonywaniu obliczeń lokalnie jak i zdalnie. Na uwagę zasługuje fakt, że urządzenie flagowe sprzed kilku lat ma wydajność porównywalną z najtańszym modelem współczesnym. Oznacza to, że nie można kwalifikować zdolności obliczeniowej na podstawie segmentu urządzenia. W dwóch przypadkach czas wykonania operacji lokalnie był krótszy niż czas potrzebny na transmisję danych. Nie da się jednak jednoznacznie przewidzieć z góry jaka będzie różnica między czasem wykonania. Z tego powodu należy wykonać kilka prób lokalnie i zdalnie by dysponować wystarczającym materiałem do celów statystycznych. Dla porównania w Tabela 6 została dodana pozycja „MacBook Pro 15". Urządzenie to

stanowiło serwer lokalny podczas testów i to na nim de facto były wykonywane obliczenia zdalne. Czas wykonania operacji na tej maszynie jest o rząd wielkości krótszy niż na urządzeniach kieszonkowych.

5. Wykorzystane technologie

Biblioteka decyzyjna jest przeznaczona do użytku przy tworzeniu aplikacji na system Android. Szkielet i biblioteki Androida są napisane w języku Java, dlatego też implementacja prototypu została również wykonana w języku Java. W rozdziale zostały opisane pozostałe technologie i biblioteki wykorzystane podczas tworzenia kodu biblioteki.

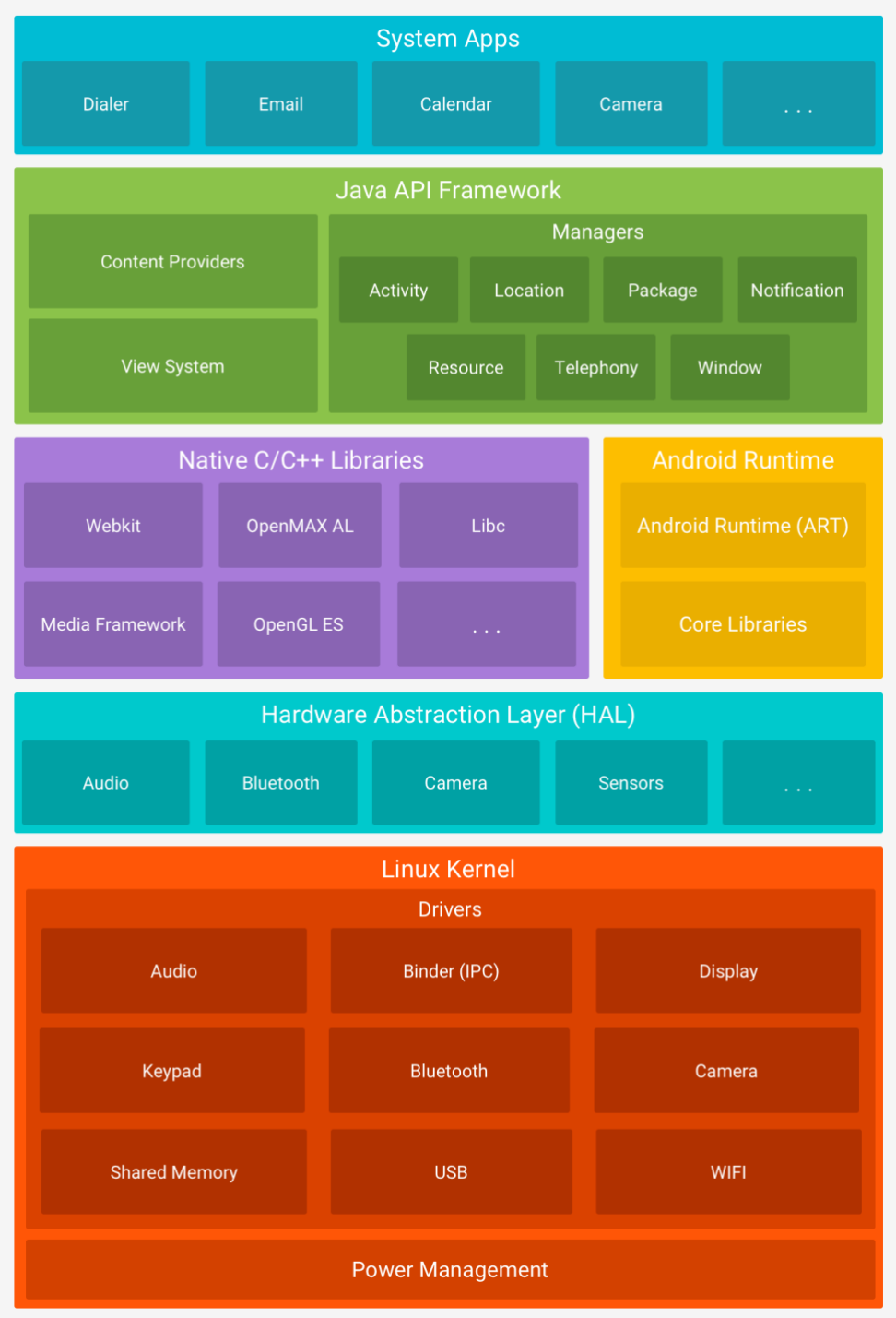
5.1. Android SDK

Według osób odpowiedzialnych za stworzenie i utrzymanie systemu Android, system ten jest najpopularniejszym i najszerzej stosowanym systemem operacyjnym [14]. Charakteryzuje go duża elastyczność, gdyż pod jego kontrolą działają takie urządzenia jak smartfony, tablety, telewizory, zegarki i inne urządzenia. Dobry system operacyjny, poza stabilnością i szybkością działania, charakteryzuje się również rozbudowanym zestawem narzędzi programistycznych służących do tworzenia aplikacji na daną platformę.

Sam system Android składa się z wielu warstw i modułów [22], jak pokazuje to Rysunek 1. Programiści aplikacji mobilnych korzystają głównie z warstwy Java API framework, toteż narzędzia programistyczne koncentrują się na łatwym dostępie do komponentów z tej warstwy. Istnieje wiele języków, w których można pisać aplikację na system Android, jednak oficjalnie wspierane są dwa języki: Java oraz Kotlin. Android SDK jest zintegrowane ze środowiskiem programistycznym Android Studio, bazującym na środowisku IntelliJ.

W bibliotece zostały wykorzystane następujące komponenty SDK:

- `Activity` – ekrany aplikacji,
- `BatteryManager` – odczytywanie stanu baterii urządzenia,
- `SensorManager` – odczytywanie temperatury baterii,
- `TrafficStats` – odczytywanie danych pobranych i przesłanych przez aplikację,
- `ResourceManager` – zarządzanie łańcuchami tekstowymi,
- `ConnectivityManager` – odczytywanie stanu połączenia internetowego.

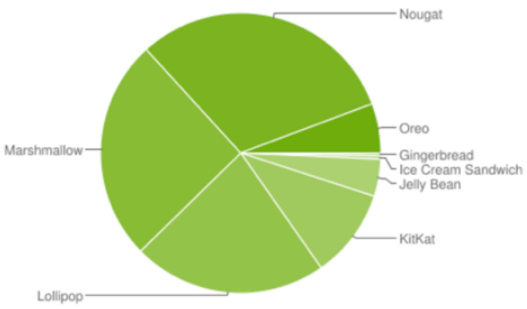


Rysunek 1: Przegląd warstw i modułów systemu Android [17]

Fakt, że system Android jest wykorzystywany przez różnych producentów, powoduje duże zróżnicowanie wersji systemu. Producenci zamiast utrzymywać stare urządzenia i przeprowadzać aktualizacje systemów, chętniej projektują nowe smartfony z nowymi wersjami systemów. Jak pokazuje

Rysunek 2 tylko 6,7% użytkowników (stan na 7 maja 2018) korzysta z najnowszej w momencie pisania pracy wersji systemu Android Oreo.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.5%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.3%
5.0	Lollipop	21	4.8%
5.1		22	17.6%
6.0	Marshmallow	23	25.5%
7.0	Nougat	24	22.9%
7.1		25	8.2%
8.0	Oreo	26	4.9%
8.1		27	0.8%



Rysunek 2: Rozkład wersji systemu Android [18]

Każda wersja systemu wprowadza nowe funkcjonalności oraz nowe moduły. By zapewnić kompatybilność wsteczną dla nowych aplikacji, twórcy systemu wprowadzili zestaw bibliotek wsparcia, pozwalających na korzystanie z nowych funkcjonalności także na urządzeniach ze starszą wersją systemu operacyjnego. Takie rozwiązanie pozwala na tworzenie nowoczesnych aplikacji, z nowymi trendami designu także na starsze wersje systemu.

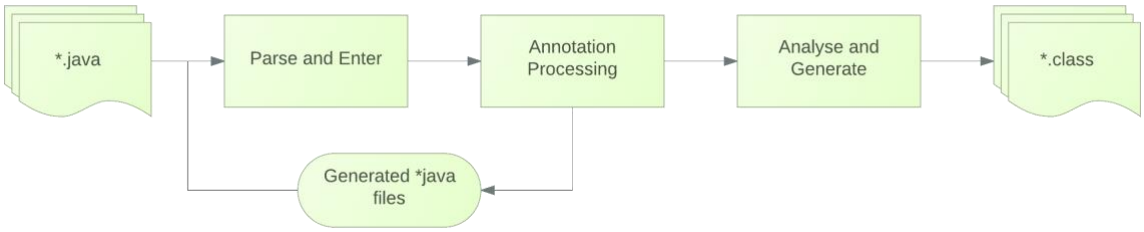
5.2. Adnotacje

Adnotacje zostały wprowadzone w Java wersji 6. Są to dodatkowe informacje stanowiące metadane kodu i nie są częścią samego programu [21]. Same w sobie nie oddziałują na kod programu. Adnotacje mają kilka zastosowań, m.in.:

- Są informacjami dla kompilatora – pomagają wykryć błędy lub sytuacje niepożądane podczas pisania kodu programu,
- Umożliwiają procesorom adnotacji generowanie kodu przed etapem kompilacji,
- Niektóre adnotacje są dostępne w trakcie działania programu.

Adnotacja oznaczana jest symbolem „`@`”, po którym następuje nazwa adnotacji, na przykład: `@Override`. Oprócz nazwy, adnotacja może zawierać dodatkowe argumenty, niezbędne podczas procesowania. W zależności od rodzaju adnotacji, może ona zostać umieszczona nad definicją klasy, metody lub pola, a także przed deklaracją typów.

Przetwarzanie adnotacji jest jednym ze standardowych kroków podczas kompilacji kodu Java. Jak pokazuje Rysunek 3, może on być wykonywany wielokrotnie, dopóki wszystkie adnotacje, także w wygenerowanych plikach źródłowych, nie zostaną przetworzone. Takie rozwiązanie pozwala na znaczne zmniejszenie ilości kodu pisanego ręcznie przez programistę, gdyż znaczną część powtarzalnego kodu można wygenerować. Procesory adnotacji mogą odczytywać wszystkie adnotacje, nie tylko zdefiniowane przez programistę. Dzięki temu wiele adnotacji można używać ponownie i nie ma potrzeby tworzenia nowych adnotacji, jeżeli w projekcie już istnieje taka, która spełnia potrzeby konkretnego przypadku.



Rysunek 3: Proces kompilacji programu Java [19]

W bibliotece decyzyjnej będącej efektem niniejszej pracy magisterskiej został napisany własny procesor adnotacji, a także została stworzona dodatkowa adnotacja. Procesor ma za zadanie wygenerowanie dodatkowych klas w projekcie, których zadaniem jest wywołanie odpowiednich metod

lokalnych lub zdalnych w zależności od podjętej decyzji, a także przeprowadzenie pomiarów parametrów i zapisanie ich do lokalnej bazy danych do celów statystycznych.

5.3. Dagger 2

Dagger 2 jest szkieletem znacznie ułatwiającym zastosowanie wzorca wstrzykiwania zależności w kodzie. Został stworzony przez firmę Square a następnie wykupiony i rozwinięty przez Google.

Sama idea wstrzykiwania zależności i odwrócenia kontroli nie jest nowa. Jej główną zasadą jest teza, że klasa nie powinna sama tworzyć sobie obiektów innych klas od których zależy. Te obiekty powinny zostać jej dostarczone z zewnątrz przez inne klasy lub dedykowany mechanizm zarządzania zależnościami. Wzorzec wstrzykiwania zależności nie wymaga stosowania żadnych dodatkowych bibliotek, jest tylko sposobem tworzenia kodu. Obiekty klas zależnych mogą być wstrzykiwane przy pomocy konstruktora albo setterów. Zastosowanie wstrzykiwania zależności ma wiele zalet:

- Możliwość podmiany obiektów zależnych w zależności od potrzeb, np. podczas tworzenia oprogramowania można łatwo zamienić połączenie z serwerem produkcyjnym na połączenie z serwerem testowym,
- Możliwość stosowania różnych implementacji dla testów automatycznych,
- Zwiększenie modułowości i rozdzielności kodu,
- Zwiększenie elastyczności rozwiązania.

Dagger 2 wyróżnia się spośród pozostałych bibliotek wstrzykiwania zależności sposobem w jaki wstrzykuje odpowiednie obiekty. Większość tego typu rozwiązań stosuje refleksję i konfiguracje XML by spełnić wymogi zależności. Dagger 2 bazuje na generowaniu kodu w momencie kompilacji. Jego konfiguracja odbywa się poprzez odpowiednie adnotacje oraz dodatkowe klasy konfiguracyjne. Do zarządzania czasem życia obiektów służą zakresy (scopes) tworzone przy pomocy dedykowanych adnotacji. Wygenerowany kod jest zbliżony do tego tworzonego przez programistów. Dzięki tej własności Dagger 2 stał się szczególnie popularny wśród programistów Androida, którzy ze względów wydajnościowych nie mogą sobie pozwolić na stosowanie refleksji. Inną zaletą Daggera 2 jest wykrywanie błędów w konfiguracji i brakujących zależności na etapie kompilacji co w przypadku aplikacji mobilnych znacząco przyspiesza tworzenie programu.

5.4. GreenDAO

GreenDAO jest biblioteką typu ORM (Object-Relational Mapping), tj. pozwala na dostęp do bazy danych poprzez standardowe obiekty Java. Dzięki zastosowaniu tej biblioteki nie ma potrzeby tworzenia tabel czy dodawania obiektów przy pomocy języka SQL. GreenDAO stanowi warstwę pośredniczącą pomiędzy bazą danych SQLite a kodem Java, jak pokazuje Rysunek 4.

Rysunek 4: Schemat architektury GreenDAO [20]

GreenDAO ma następujące zalety:

- Jest najszybszym systemem ORM dla Androida,
- Posiada rozwinięte API do zapytań i złączeń,
- Zużywa niewiele pamięci operacyjnej,
- Jej rozmiar nie przekracza 100kB,
- Wspiera SQLCipher, pozwala na zaszyfrowanie bazy danych,

Tworzenie encji i relacji odbywa się przy pomocy adnotacji. GreenDAO posiada własny procesor adnotacji a także własną wtyczkę do systemu budowania Gradle. Na etapie kompilacji tworzone są klasy pomocnicze, znacząco ułatwiające zarządzanie bazą danych. Biblioteka posiada własny mechanizm budowania zapytań, dzięki czemu jest łatwa w użyciu nawet dla osób nieznających języka zapytań SQL. Gdy jednak potrzeby programisty wykraczają poza możliwości API, GreenDAO pozwala zastosować standardowe zapytania SQL.

Biblioteka GreenDAO jest uznanym produktem na rynku aplikacji mobilnych. Magistrant w swojej pracy zawodowej miał okazję korzystania z wielu różnych bibliotek ORM i uznał GreenDAO jako najlepsze narzędzie do szybkiego uruchomienia aplikacji. Użycie prostych adnotacji zaoszczędza dużo czasu podczas tworzenia schematu bazy danych. Według portalu AppBrain [18] z biblioteki GreenDAO korzysta wiele aplikacji uznanych na rynku mobilnym, m.in.:

- UC Browser – przeglądarka internetowa,

- Pintrest,
- Musical.ly,
- Xiaomi File Manager.

5.5. Reactive Extensions

Reactive Extensions, zwana również Reactive X lub RX, jest to wieloplatformowa abstrakcja nad strumieniami zdarzeń, danych lub obiektów. Wiele języków dostarcza własne implementacje strumieni danych, jednak ich interfejs znacząco się różni w zależności od platformy. Reactive Extensions uspójnia ten interfejs i dostarcza implementacje do języków, które jej nie miały. Przykładem może być tutaj Java w wersji 6, która nie wspiera strumieni. Jest to ostatnia w pełni wspierana wersja przez SDK Androida. Implementacja strumieni i kompozycji strumieni została wprowadzona dopiero w Java 8, która z kolei jest wspierana dopiero od Androida w wersji 7 Nougat. Częstym błędem popełnianym przez początkujących programistów jest korzystanie z API strumieni Java podczas tworzenia aplikacji i testowania ich na emulatorze, który zazwyczaj zawiera najnowszą wersję systemu Android. W pracy zawodowej magistranta zdarzyło się, że jeden z młodszych programistów w zespole wykorzystał najnowsze API, uzyskał akceptację podczas recenzji kodu oraz pozytywną opinię testera i feralny kod został włączony do gałęzi produkcyjnej. Po wydaniu aktualizacji do Sklepu Play ponad połowa użytkowników nie mogła uruchomić aplikacji, gdyż miała starszą wersję systemu operacyjnego. Reactive Extensions zapobiega takim pomyłkom, ponieważ jest biblioteką zewnętrzną, zupełnie niezależną od API czy wersji Androida.

Do zalet Reactive Extensions należą:

- Duża elastyczność,
- Możliwość generowania pojedynczych zdarzeń, a także skończonych i nieskończonych strumieni danych,
- Łatwe komponowanie strumieni przy pomocy operatorów,
- Łatwe zarządzanie pulą wątków wykonania operacji,
- Zapobiega tworzeniu zagnieżdżonych funkcji anonimowych przy sekwencyjnym przetwarzaniu danych (tzw. „callback hell”),
- Implementacje we wszystkich popularnych językach programowania, a także w wielu mniej popularnych,
- Spójny interfejs pomiędzy implementacjami

W prototypie biblioteki decyzyjnej została wykorzystana implementacja Reactive Extensions dla języka Java, która nazywa się RxJava w wersji drugiej. Jest to popularna biblioteka wśród programistów aplikacji mobilnych z uwagi na wsparcie przez pozostałe wykorzystywane biblioteki.

6. Biblioteka decyzyjna

W ramach pracy dyplomowej został stworzony prototyp biblioteki decyzyjnej, która będzie automatycznie podejmowała decyzję o miejscu wykonania obliczeń. Prototyp został napisany w języku Java z wykorzystaniem Android SDK i innych bibliotek pomocniczych. Celem biblioteki było przekazanie programistom aplikacji Android narzędzia, które jest łatwe w użyciu, wymaga minimalnego nakładu pracy podczas integracji zarówno z nowym jak i istniejącym kodem, posiada domyślną konfigurację, którą można zmienić w trakcie pracy programu oraz pozwala programistom na tworzenie własnych rozwiązań w zakresie strategii podejmujących decyzję.

6.1. Przypadki użycia

Biblioteka przeznaczona jest dla aplikacji wykonujących skomplikowane operacje na urządzeniach mobilnych o zróżnicowanej mocy obliczeniowej. Ponadto, do działania biblioteki wymagany jest serwer REST posiadający metody lustrzane do tych zaimplementowanych w samej aplikacji. Biblioteka łączy interfejs lokalny i interfejs zdalny generując nową klasę o interfejsie identycznym jak lokalny. Dzięki takiemu rozwiązaniu programista w swoim kodzie korzysta z identycznego interfejsu jak dotychczas, zmienia się tylko implementacja.

Lista wybranych ogólnych przypadków użycia biblioteki:

- Zapewnienie najlepszego możliwego doświadczenia z aplikacją dla użytkowników,
- Wykonywana operacja trwa zbyt długo na starszych urządzeniach, jednocześnie trwa krótko na nowszych urządzeniach,
- Ograniczenie zużycia baterii urządzenia w przypadku, gdy istnieje możliwość wykonania obliczeń zdalnie,
- Ograniczenie ilości przesyłanych danych, w przypadku, gdy warunki środowiskowe pozwalają na wykonanie obliczeń lokalnie,
- Ograniczenie ilości wydzielanego ciepła w przypadku, gdy istnieje możliwość wykonania obliczeń zdalnie,
- Skupienie obliczeń na serwerze, z możliwością wykonania lokalnego w przypadku braku dostępnej sieci,
- Przeniesienie części obliczeń na urządzenia użytkowników w przypadku dużego obciążenia serwera,

- Rozproszenie obliczeń w celu odciążenia serwera w przypadku, gdy operacje można wykonać lokalnie,
- Zapewnienie bezbłędnego działania aplikacji, także w przypadku braku dostępu do sieci,
- Wymuszenie wykonania operacji używając najnowszej wersji algorytmu, nawet w przypadku, gdy użytkownik nie zaktualizował aplikacji.

Wymienione powyżej ogólne przypadki użycia stanowią przesłanki do rozważania nad sensem wykorzystania biblioteki decyzyjnej w kodzie aplikacji Android. Poniżej przedstawiono wybrane, konkretne przypadki użycia biblioteki:

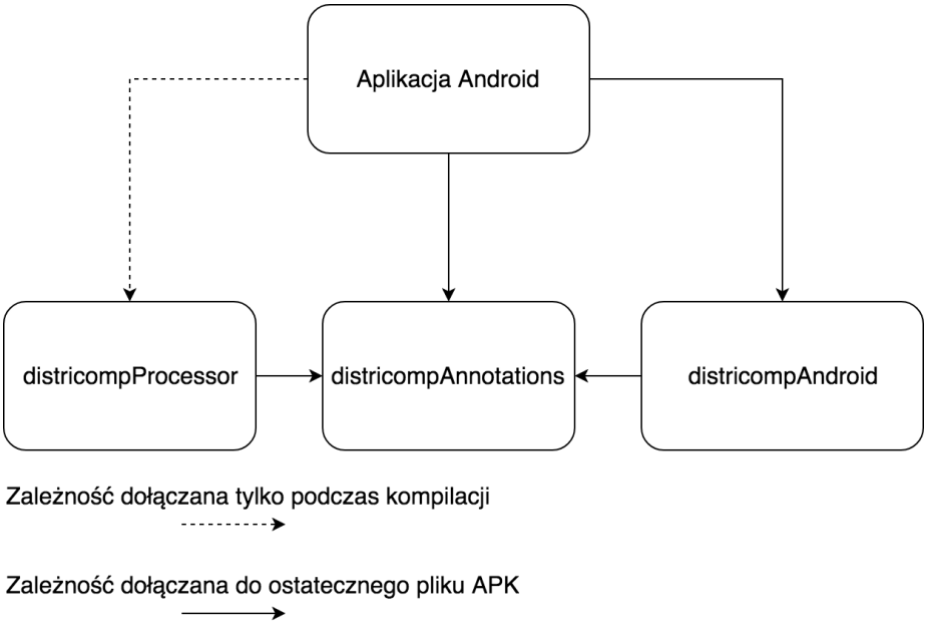
- Kompresja danych na urządzeniu, jeżeli dane na serwerze mają być przechowywane w formie skompresowanej i jest potrzeba maksymalnego odciążenia serwera,
- Przetwarzanie filmów, transkodowanie plików video, gdyż czas trwania takiej operacji na urządzeniu z małą ilością pamięci RAM i niskiej jakości układem obliczeniowo-graficznym jest niesatysfakcjonujący dla użytkownika końcowego,
- Wykonanie operacji, której algorytm zmienia się dynamicznie, a wymagane jest wykonanie zawsze na najnowszej wersji algorytmu.

6.2. Architektura rozwiązania

W skład biblioteki wchodzą trzy moduły:

- districompAnnotations – moduł Java zawierający adnotację,
- districompProcessor – moduł Java zawierający procesor adnotacji,
- districompAndroid – moduł Android zawierający pakiet bazy danych,

Zależności pomiędzy modułami przedstawia Rysunek 5. Moduł districompProcessor jest dołączany do aplikacji jedynie na etapie kompilacji co obrazuje linia przerywana. Oznacza to, że może on zależeć dodatkowo od wielu różnych bibliotek, które ostatecznie nie zostaną załączone do wynikowego pliku APK.



Rysunek 5: Diagram zależności pomiędzy aplikacją a modułami biblioteki

6.2.1 Moduł adnotacji

Moduł districompAnnotations składa się jedynie z jednej prostej klasy będącej adnotacją wykorzystywaną w projekcie. Adnotacja ta nazywa się `@DualComputation` i jest dostępna wyłączanie na poziomie kodu źródłowego. Oznacza to, że elementy oznaczone tą adnotacją będą brane pod uwagę podczas wstępnego procesowania. Może ona być umieszczona wyłącznie przy metodzie klasy. Programista, który chciałby skorzystać z biblioteki musi oznaczyć swoją metodę lokalną właśnie tą adnotacją. Nie musi jednak oznaczać konkretnej implementacji, może być to metoda interfejsu by zwiększyć elastyczność kodu. Adnotacja zawiera trzy dodatkowe parametry:

- `remoteApiInterface` – parametr obowiązkowy typu Class. Programista musi wskazać, który interfejs posłuży do stworzenia klienta do API zewnętrznego.
- `partNames` – parametr opcjonalny typu tablicowego String, domyślną wartością jest pusty łańcuch znaków. Jeżeli API zewnętrzne wykorzystuje mechanizm Multipart do przesyłania danych to w tym parametrze programista może podać nazwy poszczególnych części zapytania, zgodnie z kolejnością argumentów wywołania metody.
- `remoteMethodName` – parametr opcjonalny typu String. Domyślną wartością jest pusty łańcuch znakowy. Parametr ten pozwala programiście na wskazanie nazwy

metody z interfejsu zdalnego odpowiadającej metodzie interfejsu lokalnego. Jeżeli nazwa metody jest taka sama to parametr można zostawić pusty.

Listing 1 przedstawia przykładowe użycie adnotacji w kodzie aplikacji Android.

Listing 1: Przykład użycia adnotacji w kodzie aplikacji

```
public interface VideoEditor {

    @DualComputation(remoteApiInterface = VideoProcessingApi.class,
            partNames = {"file"},
            remoteMethodName = "cropVideo")
    Observable<File> cropVideo(File inputFile);

    @DualComputation(remoteApiInterface = VideoProcessingApi.class, partNames = {"file"})
    Observable<File> blurVideo(File inputFile);

    @DualComputation(remoteApiInterface = VideoProcessingApi.class, partNames = {"file"})
    Observable<File> redColorCast(File inputFile);

    @DualComputation(remoteApiInterface = VideoProcessingApi.class, partNames = {"file"})
    Observable<File> increaseContrast(File inputFile);
}
```

Pomimo tylko jednej klasy, moduł districompAnnotations musiał zostać wydzielony z pozostałych modułów. Przez ograniczenia wtyczek Java i Android do systemu budowania Gradle moduły Java nie mogą zależeć od modułów Android. Jednocześnie należy unikać zależności pomiędzy modułem Android i procesorem adnotacji by nie zwiększać niepotrzebnie rozmiarów pliku APK oraz ilości metod w aplikacji, która jest ograniczona do 65535 metod. Po przekroczeniu tej liczby należy zastosować MultiDex, tj. dzielenie bajtkodu wykonywalnego Dalvik na części.

6.2.2 Moduł procesora adnotacji

Moduł districompProcessor również zawiera tylko jedną klasę, której zadaniem jest wygenerowanie kodu pomocniczego dla programisty. Klasa `DistricompProcessor` w pakiecie `pl.edu.pja.districompprocessor` rozszerza klasę `AbstractProcessor` będącą klasą wbudowaną w język Java. Metoda `proces` odpowiedzialna za generowanie kodu korzysta z biblioteki pomocniczej JavaPoet, która znacznie upraszcza proces tworzenia plików źródłowych Java i pozwala uniknąć wielu pomyłek.

Procesor adnotacji zostaje wywołany przez system budowania Gradle. Przed właściwą kompilacją kodu, system budowania sprawdza dostępne procesory adnotacji oraz przetwarza wstępnie wszystkie pliki źródłowe w poszukiwaniu adnotacji. Na każdym procesorze adnotacji wywoływana jest metoda `getSupportedAnnotationTypes` zwracająca zbiór adnotacji, które dany procesor obsługuje. Dzięki temu proces budowania jest w stanie określić, które adnotacje są przetwarzane przez dany procesor i które pliki źródłowe z kodu właściwego powinien dany procesor przetworzyć. Klasa biblioteczna `DistricompProcessor` przetwarza jedynie jedną klasę adnotacji jaką jest `@DualComputation` opisaną w rozdziale 6.2.1.

Podczas samego przetwarzania, procesor sprawdza w ilu klasach pojawiła się wspierana adnotacja. Dla każdej ze znalezionych klas tworzony jest zbiór metod opatrzonych adnotacją. Następnie, tworzona jest mapa, w której kluczami są odnalezione klasy a wartościami utworzone dla danej klasy zbiory metod. Ten krok jest niezbędny, by można było wygenerować osobne pliki źródłowe dla poszczególnych klas, zawierające tylko adnotowane metody. Dzięki temu klasy wynikowe mają jednakowy interfejs jak klasa źródłowa, co z kolei pozwala na lepsze zarządzanie zależnościami i zmniejsza powiązania pomiędzy poszczególnymi komponentami kodu właściwego aplikacji. Ułatwia również integrację biblioteki do już napisanej aplikacji.

Procesor adnotacji przetwarza każdą klasę osobno. W każdej klasie pobiera z przygotowanej wcześniej mapy metody będące bazą do generowania pliku wynikowego. Każda metoda przetwarzana jest jedna po drugiej a po zakończeniu przetwarzania wszystkich metod w danej klasie generowany jest plik wynikowy będący kodem źródłowym Java. Nazwa klasy wygenerowanej jest taka sama jak nazwa klasy źródłowej z przyrostkiem `Worker`. Nowo utworzona klasa jest umieszczana w tym samym pakiecie co klasa źródłowa. Na tym etapie dodawane są również zależności nowej klasy od innych, istniejących klas:

- `Retrofit` – zależność od obiektu klasy `Retrofit` jest niezbędna do utworzenia pełnoprawnego interfejsu zdalnego.
- `DecisionStrategy` – jest to klasa abstrakcyjna pochodząca z modułu districompAndroid i odpowiada za podejmowanie decyzji o miejscu wykonywania obliczeń. Kilka zaimplementowanych strategii przedstawiono w rozdziale □.
- `MetricsCollector` – interfejs odpowiadający za rozpoczęcie i zakończenie pomiarów związanych z wykonaniem operacji oraz zwrócenie obiektu zawierającego dane wartości. Implementacja tego interfejsu znajduje się w module districompAndroid i nazywa się `MetricsCollectorImpl`.

- `MethodCallInfoManager` – inferfejs do lokalnej bazy danych, implementacja tego interfejsu znajduje się w module districompAndroid i nazywa się `MethodCallInfoManagerImpl`.
- `Local` – ostatnią zależnością jest implementacja interfejsu zawierającego adnotowaną metodę.

Oprócz samych zależności dodawane są metody pomocnicze pozwalające na zmianę strategii w trakcie wykonania programu oraz metoda zapisująca zmierzone parametry wykonania do lokalnej bazy danych. By ułatwić programistom korzystanie z wygenerowanej klasy, wszystkie zależności są wstrzykiwane przez konstruktor. Strategia decyzyjna, oprócz wstrzyknięcia przez konstruktor, może być także wstrzyknięta poprzez wygenerowany setter. Konstruktor zostaje także opatrzony adnotacją `@Inject` z pakietu `javax.inject`. Adnotacja ta jest rozpoznawana przez najpopularniejszy system zarządzania zależnościami stosowany w programowaniu aplikacji Android pod nazwą Dagger 2, który został szerzej opisany w rozdziale 5.3.

Pierwszym etapem przetwarzania danej metody lokalnej jest sprawdzenie poprawności danych. Procesor sprawdza poprawność otrzymanych danych, tj.:

- Sprawdza czy w interfejsie zdalnym występuje metoda o podanej w adnotacji nazwie lub, jeżeli nazwy nie podano, czy w interfejsie zdalnym znajduje się metoda o takiej samej nazwie jak metoda lokalna.
- Sprawdza czy zwracany typ metody zdalnej i lokalnej jest taki sam
- Sprawdza czy listy argumentów w metodzie zdalnej i lokalnej się pokrywają.

Jeżeli którykolwiek z powyższych warunków nie jest spełniony to procesor informuje programistę o wystąpieniu błędu przetwarzania oraz sugeruje możliwe przyczyny oraz rozwiązania problemu. Przetwarzanie zostaje na tym etapie przerwane i nie są generowane żadne pliki źródłowe.

Gdy weryfikacja wstępna zostanie pomyślnie zakończona, procesor przystępuje do generowania ciała metody. Na początku generowana jest linia kodu odpowiedzialna za podjęcie decyzji o miejscu wykonania operacji na podstawie wstrzykniętej strategii podejmowania decyzji. By ułatwić proces tworzenia oprogramowania innym programistom, zostaje dodana również linia logująca podjętą decyzję w standardowym loggerze Androida, tj. LogCat. Następnie generowane jest właściwe wywołanie metody zdalnej lub lokalnej w zależności od podjętej decyzji. Do każdego z wywołań zostają dodane instrukcje pomocnicze mające na celu rozpoczęcie oraz zakończenie pomiaru parametrów wywołania danej moetody oraz zapisanie ich do bazy danych. Ponadto, generowany jest dodatkowy blok kodu

odpowiadający za mechanizm ponawiania wykonania operacji w innym miejscu niż pierwotnie w przypadku błędu podczas pierwszej próby. Mechanizm ten został szerzej opisany w rozdziale 6.2.3.

Listing 2: Przykład klasy wygenerowanej przez procesor adnotacji

```
public final class MathComputerWorker {
  private final Retrofit retrofit;

  private final MathComputer local;

  private final MethodCallInfoManager methodCallInfoManager;

  private final MetricsCollector metricsCollector;

  private DecisionStrategy strategy;

  @Inject
  public MathComputerWorker(Retrofit retrofit, MathComputer local, DecisionStrategy strategy,
      MethodCallInfoManager methodCallInfoManager, MetricsCollector metricsCollector) {
    this.retrofit = retrofit;
    this.local = local;
    this.strategy = strategy;
    this.methodCallInfoManager = methodCallInfoManager;
    this.metricsCollector = metricsCollector;
  }

  public void setStrategy(DecisionStrategy strategy) {
    this.strategy = strategy;
  }

  private void stopCollecting(String methodName, ComputationLocation decision) {
    MethodCallInfo info = metricsCollector.stopCollecting();
    info.setClassName(MathComputer.class.getCanonicalName());
    info.setMethodName(methodName);
    info.setComputationLocation(decision);
    methodCallInfoManager.addNewMethodCallData(info);
  }

  public Observable<Integer> fibonacci(Integer value) {
    final ComputationLocation decision = strategy.decide(MathComputer.class, "fibonacci");
    Log.d("DistricompAndroid", "Decision is: " + decision.toString());
    MathRemote api = retrofit.create(MathRemote.class);
    Observable<Integer> observable;
    if (decision == ComputationLocation.LOCAL) {
      observable = local.fibonacci(value).subscribeOn(Schedulers.computation())
```

```
        .doOnSubscribe(it -> metricsCollector.startCollecting())
        .doOnTerminate(() -> stopCollecting("fibonacci", decision));
    if (strategy.isFallback()) {
      observable = observable.onErrorResumeNext(api.fibonacci(value)
      .subscribeOn(Schedulers.io())
      .doOnSubscribe(it -> metricsCollector.startCollecting())
      .doOnTerminate(() -> stopCollecting("fibonacci", decision)));}
   } else {
     observable = api.fibonacci(value)
        .subscribeOn(Schedulers.io())
        .doOnSubscribe(it -> metricsCollector.startCollecting())
        .doOnTerminate(() -> stopCollecting("fibonacci", decision));
    if (strategy.isFallback()) {
      observable = observable.onErrorResumeNext(local.fibonacci(value).subscribeOn(Schedulers.computation())
      .doOnSubscribe(it -> metricsCollector.startCollecting())
      .doOnTerminate(() -> stopCollecting("fibonacci", decision)));}
   }
   return observable;
 }
}
```

Listing 2 przedstawia przykładowy wygenerowany kod przez procesor adnotacji. W tym konkretnym przykładzie operacją obliczeniową jest obliczanie n-tego wyrazu ciągu Fibonacciego. Przykładowa klasa `MathComputer` zawiera tylko jedną adnotowaną metodę o nazwie `fibonacci`. Wygenerowany kod jest w pełni rozpoznawalny przez IDE Android Studio, wystarczy przynajmniej raz zbudować projekt.

Po przetworzeniu wszystkich adnotowanych metod we wszystkich klasach wygenerowane zostają odpowiednie pliki źródłowe umieszczane w pakietach odpowiadających klas lokalnych. Pliki te można obejrzeć i przeanalizować, gdyż przez zastosowanie biblioteki pomocniczej JavaPoet są one czytelnie sformatowane.

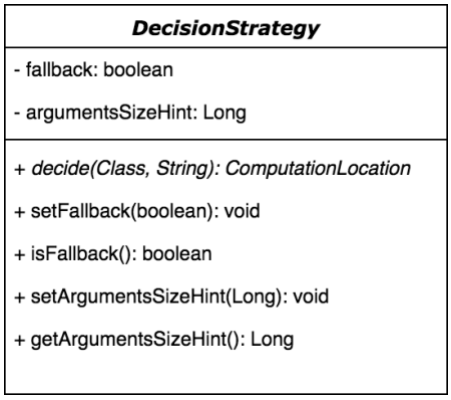
6.2.3 Moduł Android

Ostatnim modułem biblioteki decyzyjnej jest biblioteka Android, mająca dostęp do SDK systemu operacyjnego i oferowanych przez niego funkcjonalności. Dostarcza ona wiele podstawowych klas stanowiących podstawę działania całej biblioteki. Należy do nich prosta klasa wyliczeniowa nazwana `ComputationLocation`, która zawiera dwa pola. Reprezentuje ona miejsce wykonywania obliczeń:

- `LOCAL` – operacja wykonana lokalnie na urządzeniu,
- `REMOTE` – operacja wykonana zdalnie na serwerze.

Wspomniana wyżej klasa wyliczeniowa jest bezpośrednio wykorzystywana w klasie abstrakcyjnej `DecisionStrategy` stanowiącej podstawę strategii decyzyjnych opisanych szczegółowo w rozdziale 6.3. Jej diagram UML przedstawiono na Rysunek 6. Jest to prosta klasa abstrakcyjna posiadająca dwa pola:

- `fallback` – pole typu `boolean` stanowiące flagę informującą bibliotekę o ponowieniu próby wykonania operacji w przypadku błędu. Mechanizm ten jest szczególnie przydatny w sytuacji wykonywania operacji zdalnie, gdy wystąpi błąd sieciowy. Wtedy biblioteka podejmie próbę wykonania operacji lokalnie.
- `argumentsSizeHint` – pole typu `Long` dostępne dla strategii reprezentujące szacunkową wielkość danych transmitowanych przez sieć podczas wykonania zdalnego operacji. Dzięki tej informacji można lepiej przewidzieć, jak dużo czasu będzie potrzebne na przesłanie danych przez sieć i na tej podstawie poprawić jakość podejmowanych decyzji.

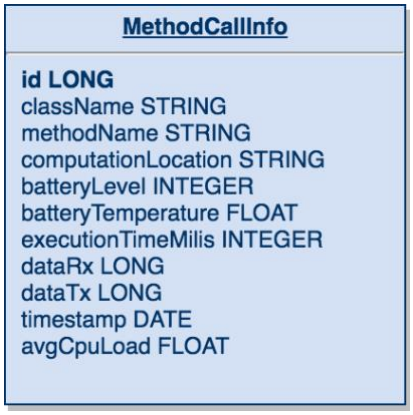


Rysunek 6: diagram UML klasy bazowej strategii

Klasa `DecisionStrategy` posiada również jedną metodę abstrakcyjną `decide`, która jako argumenty przyjmuje obiekt typu `Class` identyfikujący klasę lokalną oraz obiekt typu `String` będący nazwą metody, której dotyczy dane wywołanie strategii decyzyjnej. To właśnie ta metoda jest wywoływana przez wygenerowaną przez procesor adnotacji klasę pomocniczą, opisaną w podpodrozdziale 6.2.2. Klasy rozszerzające klasę bazową mogą korzystać z dodatkowych zależności by dostosować decyzję do własnych potrzeb.

Biblioteka decyzyjna korzysta z lokalnej bazy danych do przechowywania parametrów zebranych podczas wywoływania metod lokalnych oraz zdalnych. Dane te są niezbędne do poprawienia jakości

podejmowanych decyzji przez strategie decyzyjne. Baza danych składa się z jednej tabeli. Jej schemat przedstawia Rysunek 7.



Rysunek 7: Schemat tabeli MethodCallInfo

Dostęp do danych odbywa się poprzez specjalizowany manager. W razie potrzeby, programista może dostać się bezpośrednio do bazy danych i wykonać na niej własne zapytania. Jest to operacja ryzykowna i powinno się korzystać w miarę możliwości z implementacji interfejsu `MethodCallInfoManager`. Zawiera ona następujące metody:

- `addNewMethodCallData` – metoda służąca do dodawania nowych danych do bazy, wykorzystywana przez wygenerowaną przez procesor adnotacji klasę pomocniczą.
- `getAvgTime` – metoda przyjmuje jako parametry nazwę klasy oraz nazwę metody. Ponadto przyjmuje parametr typu `ComputationLocation` wskazujący na miejsce wykonania obliczeń. Zwraca średni czas wykonania metody zdalnej lub lokalnej w zależności od parametru. Czas zdalny zawiera również czas potrzebny na przesłanie i odebranie danych z serwera. Istnieje również przeciążenie tej metody przyjmujące również wartość temperatury i zwracające średni czas wykonania operacji w podobnych warunkach temperaturowych.
- `getAvgCpuLoad` - metoda przyjmuje jako parametry nazwę klasy oraz nazwę metody. Ponadto przyjmuje parametr typu `ComputationLocation` wskazujący na miejsce wykonania obliczeń. Zwraca średnie obciążenie procesora podczas wykonywania operacji zdalnie lub lokalnie, w zależności pod podanego parametru wejściowego.

- `getTotalInvocations` - metoda przyjmuje jako parametry nazwę klasy oraz nazwę metody. Ponadto przyjmuje parametr typu `ComputationLocation` wskazujący na miejsce wykonania obliczeń. Zwraca liczbę wywołań zdalnych lub lokalnych w zależności od podanego parametru wejściowego.
- `getAvgDataTx` - metoda przyjmuje jako parametry nazwę klasy oraz nazwę metody. Ponadto przyjmuje parametr typu `ComputationLocation` wskazujący na miejsce wykonania obliczeń. Zwraca średnią ilość danych przesłanych do serwera podczas wykonania zdalnego lub lokalnego w zależności od podanego parametru wejściowego.
- `getAvgDataRx` - metoda przyjmuje jako parametry nazwę klasy oraz nazwę metody. Ponadto przyjmuje parametr typu `ComputationLocation` wskazujący na miejsce wykonania obliczeń. Zwraca średnią ilość danych odebranych od serwera podczas wykonania zdalnego lub lokalnego w zależności od podanego parametru wejściowego.
- `getTotalDataFromLastMonth` - metoda przyjmuje jako parametry nazwę klasy oraz nazwę metody. Zwraca sumaryczną ilość danych odebranych lub przesłanych do serwera w ciągu ostatniego miesiąca.

Opisane metody pozwalają na odczytanie najważniejszych danych niezbędnych do poprawienia jakości podejmowanych decyzji. Jeżeli są one niewystarczające dla programisty aplikacji, może on rozszerzyć klasę implementującą ten interfejs i dopisać swoje metody korzystając z wygenerowanej klasy dostępu do bazy danych `MethodCallInfoDao`. Klasa ta została wygenerowana przy pomocy biblioteki ORM greenDAO, która została szerzej opisana w rozdziale 5.4.

By skorzystać z biblioteki, programista musi wykonać metodę inicjalizacyjną podczas startu programu. Metoda inicjalizacyjna jest metodą statyczną i znajduje się w klasie `DistricompAndroid` pod nazwą `init`. Jako parametr wejściowy metoda przyjmuje obiekt klasy `Context` pochodzącej z Android SDK. Najlepszym miejscem na umieszczenie metody inicjalizacyjnej jest metoda `onCreate` klasy rozszerzającej klasę `Application`.

6.3. Przegląd zaimplementowanych strategii decyzyjnych

W prototypie biblioteki zostały zaimplementowane przykładowe strategie decyzyjne by ułatwić programistom integrację biblioteki z własnym kodem. Twórcy programów mogą skorzystać z gotowych

strategii, zaimplementować własne w całości, wykorzystać strategie pomocnicze lub rozszerzyć istniejące.

6.3.1 Strategie trywialne

W prototypie biblioteki zaimplementowano dwie strategie trywialne:

- `AlwaysLocalStrategy` – strategia zawsze podejmuje decyzję o lokalnym wykonaniu obliczeń
- `AlwaysRemoteStrategy` – strategia zawsze podejmuje decyzję o zdalnym wykonaniu obliczeń

Znajdują się w pakiecie: `pl.edu.pja.districompandroid.strategy.basic`. Pomimo swojej trywialności, strategie te mają wiele zastosowań, m.in.:

- Testowanie aplikacji podczas pisania oprogramowania,
- Testowanie aplikacji przy pomocy testów automatycznych,
- Czasowe wyłączanie funkcjonalności decyzyjnej w przypadku awarii serwera zewnętrznego lub nieaktualnego algorytmu lokalnego,
- Kompozycja z innymi strategiami.

Inną strategią trywialną jest strategia `GeneratorStrategy`, która została umieszczona w pakiecie: `pl.edu.pja.districompandroid.strategy`. Ta strategia podejmuje decyzję na podstawie liczby poprzednich wywołań danej metody w zależności od decyzji. Jeżeli liczba wywołań lokalnych jest mniejsza lub równa liczbie wywołań zdalnych to strategia podejmuje decyzję o wykonaniu lokalnym. W przeciwnym wypadku strategia zwraca decyzję o wykonaniu zdalnym. Jej głównym zastosowaniem jest generowanie danych w bazie danych. Może ona zostać wykorzystana do celów statystycznych lub do celów testowych.

6.3.2 Strategie kompozycyjne

W bibliotece zaimplementowano kilka strategii, które same w sobie nie podejmują decyzji a jedynie są odpowiednimi kontenerami na pozostałe strategie i na podstawie ich decyzji podejmowana jest decyzja ostateczna.

Są one w pakiecie: `pl.edu.pja.districompandroid.strategy.composite`. Znajduje się tam strategia bazowa `CompositeStrategy`, która stanowi podstawę do tworzenia pozostałych strategii kompozycyjnych:

- `LocalPreferredCompositeStrategy` – przyjmuje dowolną liczbę strategii. Podczas podejmowania decyzji strategia odpytuje wszystkie zawarte w sobie strategie o decyzję i jeżeli którakolwiek z nich wskazuje na lokalne wykonanie obliczeń to ostateczna decyzja jest lokalna.
- `RemotePreferredCompositeStrategy` – podobnie jak powyższa, przyjmuje dowolną liczbę strategii. Podczas podejmowania decyzji odpytywane są wszystkie zawarte strategie i jeżeli którakolwiek zwróci decyzję zdalną to ostateczna decyzja też jest zdalna.
- `WeightedCompositeStrategy` – strategia ta przyjmuje dowolną liczbę strategii. Przy dodawaniu strategii należy podać również wagę, z jaką należy brać pod uwagę decyzję z danej konkretnej strategii do obliczania decyzji ostatecznej. Obliczanie odbywa się poprzez sumowanie wag i sprawdzanie która decyzja uzyskała lepszy wynik.
- `NetworkSafeCompositeStrategy` – w odróżnieniu od pozostałych strategii kompozycyjnych strategia ta przyjmuje tylko jedną dodatkową strategię. Jej celem jest zapewnienie wykonania działania, nawet w przypadku braku połączenia internetowego. Strategia początkowo sprawdza czy urządzenie posiada aktywne połączenie internetowe. W przypadku braku takiego połączenia, zawsze podejmuje decyzję o lokalnym wykonaniu obliczeń. W innym przypadku, ostateczna decyzja jest taka jaką podejmie strategia dodatkowa.

Strategie kompozycyjne pozwalają w łatwy sposób łączyć inne strategie by osiągnąć zakładane cele bez duplikowania kodu.

6.3.3 Strategie ogólne

W prototypie biblioteki decyzyjnej zaimplementowano kilka strategii ogólnego przeznaczenia. Są to strategie, które zależą od dodatkowych klas dostarczających dodatkowe informacje mogące mieć wpływ na podjętą decyzję. Poniżej przedstawiono opisy zaimplementowanych strategii:

- `CpuLoadOptimizationStrategy` – jest to strategia, która swoją decyzję podejmuje na podstawie poprzednich wywołań metody oraz aktualnego obciążenia procesora. Jeżeli aktualne obciążenie procesora jest większe niż 50% oraz poprzednie wywołania lokalne wymagały ponad 50% mocy procesora to zwracana jest decyzja o zdalnym wykonaniu operacji. W przeciwnym wypadku operacja zostanie wykonana lokalnie.

- `DataSavingStrategy` – strategia ta sprawdza czy nie przekroczony został miesięczny limit wykorzystanych danych. Posiada ona dodatkowe pole `limit` które stanowi limit miesięczny przesłanych danych dla danej metody wyrażony w bajtach. Podczas podejmowania decyzji, strategia odczytuje z bazy danych sumę pobranych i wysłanych danych przez daną metodę. Jeżeli wartość ta jest większa niż ustawiony limit to strategia zwróci decyzję o lokalnym wykonaniu operacji. W przeciwnym wypadku operacja zostanie wykonana na serwerze. Limit danych w strategii można dowolnie ustawić w trakcie wykonania programu.

- `PowerSaveStrategy` – celem tej strategii jest oszczędzanie energii elektrycznej zgromadzonej w akumulatorze urządzenia. Strategia ta jest szczególnie ważna w kontekście wyników ankiety dla użytkowników, którzy wskazali właśnie krótki czas pracy na baterii jako najbardziej irytujący czynnik podczas korzystania z urządzeń mobilnych. Jej decyzja uzależniona jest od aktualnego stanu naładowania akumulatora. Zawiera ona dodatkowy parametr `threshold`, którego domyślną wartością jest 50%. Domyślny limit jest konsekwencją wyników eksperymentu, który wskazał na liniowy sposób rozładowywania baterii urządzeń. Wartość parametru można dowolnie zmienić, także w trakcie wykonania programu. Jeżeli aktualny stan baterii jest większy od ustalonego progu to strategia zwraca decyzję o lokalnym wykonaniu operacji. W przeciwnym wypadku operacja zostanie wykonana na serwerze.

- `TemperatureStrategy` – zadaniem tej strategii jest zapobieganie przegrzania urządzenia. Wyniki eksperymentu wskazały na znaczący wzrost temperatury urządzenia podczas lokalnego wykonania operacji. Strategia zawiera dodatkowe pole `temperatureThreshold`, które stanowi próg zwrotny dla podejmowanej decyzji. Jeżeli urządzenie jest chłodniejsze niż zakładany próg to strategia podejmie decyzję o lokalnym wykonaniu operacji. W przeciwnym wypadku obliczenia zostaną wykonane na serwerze. Domyślną wartością progu jest 35 stopni Celsjusza, jednak programista może w dowolnym momencie zmienić tę wartość.

- `TimeOptimizationStrategy` – strategia podejmuje decyzję by zoptymalizować czas wykonania operacji. Strategia korzysta ze wskazówki dotyczącej rozmiaru przesyłanych danych podawanej przez programistę przed wykonaniem operacji. Na podstawie rozmiaru przesyłanych danych oraz teoretycznej prędkości transferu danych można oszacować czas potrzebny na przesłanie i odbiór danych poprzez sieć. Czas ten jest porównywany ze średnim czasem wykonania

operacji lokalnie. Jeżeli jest krótszy to podejmowana jest decyzja o wykonaniu operacji zdalnie. W przeciwnym wypadku operacja zostanie wykonana lokalnie.

- `AutoStrategy` – by maksymalnie ułatwić integrację biblioteki z kodem aplikacji, została utworzona strategia automatyczna. Jest ona kompozycją niektórych powyższych strategii. Algorytm podejmowania decyzji przedstawia się następująco:
  - Najpierw sprawdzana jest dostępność sieci. Jeżeli sieć nie jest dostępna to zostaje natychmiastowo podjęta decyzja o wykonaniu operacji lokalnie.
  - W przeciwnym wypadku wykorzystywana jest strategia oszczędzania energii elektrycznej. Jeżeli zwróci ona wynik zdalny to ten wynik jest zwracany jako ostateczny
  - W przeciwnym wypadku sprawdzana jest ilość zebranych danych dotyczących tego konkretnego wywołania. Jeżeli liczba danych jest niewystarczająca by oszacować prędkość operacji z określoną dokładnością to wykorzystywana jest strategia trywialna generowania danych.
  - W przeciwnym wypadku ostateczną decyzją jest decyzja zwrócona przez strategię optymalizującą czas wykonania operacji.

Strategia automatyczna jest strategią domyślną, zaprojektowaną przez magistranta na podstawie wyników ankiet użytkowników oraz wyników eksperymentu. Według autora jest to strategia optymalna w ogólnym przypadku, jednakże zróżnicowanie przypadków użycia biblioteki pozwala sądzić, że programiści wykorzystujący tę bibliotekę będą starali się korzystać z różnych strategii lub będą tworzyć własne, specjalizowane strategie odpowiadające swoim potrzebom.

7. Podsumowanie pracy

Tworzenie aplikacji mobilnych Android jest stosunkowo proste dzięki bardzo dobrym narzędziom programistycznym oferowanym przez twórców systemu. Problem pojawia się wtedy, gdy aplikacja jest publikowana i zaczynają korzystać z niej użytkownicy. System Android jest powszechnym systemem, stosowanym przez wielu różnych producentów urządzeń mobilnych. Użytkownicy uruchamiający aplikację na starszych urządzeniach lub urządzeniach o niższej mocy obliczeniowej, mogą narzekać na powolne jej działanie i szybkie zużywanie baterii urządzenia. Z drugiej strony, użytkownicy korzystający z topowych urządzeń o wysokiej mocy obliczeniowej mogą narzekać, że niektóre operacje takie jak obróbka filmu czy zdjęcia wymagają aktywnego połączenia internetowego. Zróżnicowanie urządzeń stawia duże wyzwanie programistom aplikacji dotyczące decyzji o miejscu wykonywania obliczeń.

Z badania ankietowego wynika, że użytkownicy coraz częściej zastępują laptop czy komputer stacjonarny urządzeniem mobilnym. Jest to zrozumiałe, gdyż oferta aplikacji w sklepie Play jest na tyle bogata, że użytkownicy nie mają problemu ze znalezieniem odpowiedniej aplikacji. Jednocześnie użytkownicy zwykle są stale podłączeni do Internetu. Z wyników ankiety można wnioskować, że zastosowanie podwójnej implementacji, lokalnej na urządzeniu oraz zdalnej na serwerze, jest uzasadnione i poprawi odbiór danej aplikacji wśród użytkowników. Z drugiej ankiety, skierowanej do programistów, wynika, że problemy niewystarczającej wydajności na urządzeniach mobilnych są znane. Zwykle jednak rozwiązaniem jest całkowite przeniesienie obliczeń do serwera zdalnego co uniemożliwia korzystanie z aplikacji w trybie offline, bez dostępu do sieci.

Przeprowadzony eksperyment potwierdził przewagę wykonania zdalnego nad wykonaniem lokalnym, biorąc pod uwagę najważniejsze kryterium dla użytkowników, tj. zużycie energii elektrycznej. Jednocześnie wyniki eksperymentu z różnych urządzeń potwierdzają relatywnie stały czas wykonania operacji przy wykonaniu zdalnym oraz bardzo zróżnicowany czas wykonania operacji lokalnie. Na urządzeniach sprzed roku czas wykonania operacji testowej był krótszy niż czas potrzebny na transmisję danych na serwer oraz wykonanie obliczeń na serwerze, natomiast nawet na flagowych urządzeniach sprzed kilku lat szybciej było wykonać operację zdalnie.

Stworzony prototyp biblioteki rozwiązuje ten problem podejmując decyzję o miejscu wykonywania obliczeń automatycznie, na podstawie wybranej strategii działania oraz czynników środowiskowych. Wyposażony został w procesor adnotacji, który na podstawie oznaczeń kodu generuje dodatkowe klasy pomocnicze, zarządzające wywołaniem funkcji. Ponadto, wygenerowane klasy odpowiadają za zebranie statystyk dotyczących danego wywołania funkcji oraz umieszczenia ich w bazie danych w celu polepszenia jakości podejmowanej decyzji.

Prototyp biblioteki można rozwinąć o dodatkowe funkcjonalności. Bez wątpienia najbardziej uciążliwym elementem wymaganym do działania biblioteki jest konieczność utrzymywania dwóch jednakowych algorytmów: jednego po stronie aplikacji mobilnej a drugiego po stronie serwera. Jednym z możliwych kierunków rozwoju jest stworzenie systemu migracji kodu wykonywalnego pomiędzy serwerem a aplikacją mobilną. Dzięki temu można wyeliminować konieczność tworzenia tego samego algorytmu dwukrotnie, przez co kod stanie się łatwiejszy w utrzymaniu. Kod Java można stosunkowo łatwo przenosić pomiędzy platformami. Należy jednak pamiętać, że Java dla Androida posiada pewne ograniczenia i trzeba to uwzględnić podczas przenoszenia kodu pomiędzy aplikacją mobilną i serwerem. Innym możliwym rozwiązaniem jest zastosowanie języka Kotlin, który jest już oficjalnie wspierany w pełni na platformie Android, a także zdobywa coraz większą popularność w rozwiązaniach serwerowych.

8. Bibliografia

[1]. PILCH Tadeusz : *Zasady badań pedagogicznych*. - Wyd. 2 popr. i rozsz. - Warszawa : Wydaw. Akadem. "Żak", 1998

[2]. CYBULSKI Krzysztof : Badania Marketingowe 2013, wykład na Katedrze Marketingu, Wydział Zarządzania, Uniwersytet Warszawski, url: http://www.wz.uw.edu.pl/pracownicyFiles/id5481-badania-marketingowe08.pdf, dostęp 1.03.2018

[3]. Serwis internetowy Surveymonkey.com, url: https://www.surveymonkey.com/mp/take-a-tour/?ut_source=header , dostęp 1.09.2017

[4]. Statista. (2018, 5 15). *Number of available applications in the Google Play Store from December 2009 to March 2018*. Pobrano z lokalizacji witryna sieci Web firmy Statista: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store

[5]. Google. (2017, 12 4). *Strona pobierania aplikacji Traficar*. Pobrano z lokalizacji Sklep Google Play: https://play.google.com/store/apps/details?id=pl.express.traficar

[6]. Google. (2017, 12 4). *Strona pobierania aplikacji Panek*. Pobrano z lokalizacji Sklep Google Play: https://play.google.com/store/apps/details?id=com.vulog.carshare.panek

[7]. MGSM. (2018, 3 5). *MGSM.pl*. Pobrano z lokalizacji Xiaomi Redmi 4A - Dane techniczne: http://www.mgsm.pl/pl/katalog/xiaomi/redmi4a/

[8]. MGSM. (2018, 3 1). *MGSM.pl*. Pobrano z lokalizacji Samsung Galaxy Mini 2 - Dane techniczne: http://www.mgsm.pl/pl/katalog/samsung/galaxymini2/

[9]. MGSM. (2018, 3 1). *MGSM.pl*. Pobrano z lokalizacji Samsung Galaxy J5 2017- Dane techniczne: http://www.mgsm.pl/pl/katalog/samsung/galaxyj52017/

[10]. MGSM. (2018, 3 1). *MGSM.pl*. Pobrano z lokalizacji Sony Xperia Z3 - Dane techniczne: http://www.mgsm.pl/pl/katalog/sony/xperiaz3/

[11]. MGSM. (2018, 3 1). *MGSM.pl*. Pobrano z lokalizacji Samsung Galaxy S8 - Dane techniczne: http://www.mgsm.pl/pl/katalog/samsung/galaxys8/

[12]. Van Dijk Management Consultunts. (2016). *Mobile Broadband Prices*. Bruksela: Komisja Europejska.

[13]. Android.com.pl. (2018, 3 15). *Forum Android.com.pl*. Pobrano z lokalizacji Grzanie się telefonu: https://forum.android.com.pl/topic/281161-grzanie-si%C4%99-telefonu/

[14]. Developer Android. (2018, 6 10). *Android Platform*. Pobrano z lokalizacji About: https://developer.android.com/about/

[15]. Oracle. (2018, 6 12). *Oracle Java Documentation*. Pobrano z lokalizacji Lesson: Annotations: https://docs.oracle.com/javase/tutorial/java/annotations/

[16]. AppBrain. (2018, 6 14). *AppBrain.* Pobrano z lokalizacji greenDAO - Android Library Statistics: https://www.appbrain.com/stats/libraries/details/greendao/greendao

[17]. Android Developer (2018, 6 12) *Android platform Documentation*. Pobrano z lokalizacji: https://developer.android.com/guide/platform/

[18]. Android Developer (2018, 6 12) Android Versions. Pobrano z lokalizacji: https://developer.android.com/about/dashboards/

[19]. Louis G. Valle (2018, 6 12) *How ButterKnife actually works?* Pobrano z lokalizacji: http://lgvalle.xyz/2015/05/04/butterknife/

[20]. GreenRobot (2018, 6 12) *GreenDAO – Introduction*. Pobrano z lokalizacji: http://greenrobot.org/greendao/documentation/introduction/

[21]. ECKEL Bruce: *Thinking in Java. Edycja polska. Wydanie IV*. Rodział 20: Adnotacje, Wydawnictwo: Helion, 2011

[22]. ANUZZI Joseph, DARCEY Lauren, CONDER Shane, *Android. Wprowadzenie do programowania aplikacji.* Wydanie V. Wydawnictwo: Helion, 2016

9. Spis wykresów

10. Spis tabel

11. Spis rysunków

12. Spis listingów