JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 11 (2019) 155-162
Wysłane: 2019-04-11
Przyjęte: 2019-04-16

Analiza możliwości współpracy aplikacji mobilnych z usługami sieciowymi typu REST i Web Service

Mateusz Roman Daraż*, Piotr Kopniak
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono wyniki przeglądu bibliotek na platformę Android, wspomagających integrację z usługami sieciowymi typu REST i SOAP. Przeprowadzono analizę wydajności i złożoności programów klienckich korzystających z najpopularniejszych rozwiązań. Metody badawcze oparto na autorskiej aplikacji mobilnej umożliwiającej symulację komunikacji asynchronicznej. Wynikiem badań jest analiza porównawcza bibliotek, ułatwiająca wybór adekwatnych rozwiązań.

Słowa kluczowe: aplikacje mobilne; Android; SOAP; REST

*Autor do korespondencji.

Adres e-mail: mateusz.daraz@gmail.com

Analysis of the possibilities of cooperation of mobile applications with network services of the type REST and Web Service

Mateusz Roman Daraż*, Piotr Kopniak
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents the results of a review of libraries for the Android platform, supporting the integration with REST and SOAP network services. An analysis of the performance and complexity of client programs using the most popular solutions. The research methods were based on the author's mobile application that enable the simulation of asynchronous communication. The result of the research is a comparative analysis of libraries, that facilitate the selection of adequate solutions.

Keywords: mobile applications; Android; SOAP; REST

*Corresponding author.

E-mail address: mateusz.daraz@gmail.com

1. Wstęp

Dynamiczny rozwój rozwiązań mobilnych opartych na usługach sieciowych, stawia przed architektami systemów i programistami aplikacji mobilnych, coraz wyższe wymagania dotyczące implementacji, wydajności oraz skalowalności programów. W praktyce budowa systemów mobilnych, wiąże się z wykorzystaniem oprogramowania wspomagającego integrację, a co za tym idzie koniecznością wyboru adekwatnych rozwiązań. Zastosowanie odpowiedniej technologii jest istotne zarówno w kontekście aplikacji klienckiej, jak również usług sieciowych z którymi współpracuje.

Niniejszy artykuł pokazuje jak ważnym problemem są możliwości gotowych bibliotek systemowych i zewnętrznych, wykorzystanych do budowy programów mobilnych zintegrowanych z usługami typu REST i SOAP. Bada prawdziwość tezy, mówiącej, że usługi sieciowe typu REST są lepszym rozwiązaniem od usług SOAP w przypadku korzystania z nich poprzez mobilne aplikacje klienckie. Podstawę do użycia odpowiedniej technologii stanowi jej wydajność, którą analizowano na podstawie czasu dostępu do danych zawartych w przetworzonej odpowiedzi serwera. Poza tym, ze względu na rosnące wymagania odnośnie funkcjonalności, jak i prostoty obsługi aplikacji nie bez znaczenia pozostaje kwestia złożoności implementacyjnej, która jest istotnym elementem procesu wytwórczego.

Wykonane analizy porównawcze, przedstawiają możliwości implementacyjne jakimi dysponują twórcy programów mobilnych i ułatwiają zastosowanie rozwiązania, które spełni wymagania stawiane przed systemem. Badano oprogramowanie na popularną platformę Android, posiadającą największy udział w rynku mobilnych systemów operacyjnych, według statystyk organizacji StatCounter GlobalStats, który w lutym 2019 roku wynosił 74,15 % [1]. W ramach prac badawczych przeprowadzono analizę skomplikowania programów klienckich wykonaną w oparciu o metryki statyczne oraz analizę wydajnościową, sprawdzające prawdziwość poniższych hipotez:

1) programy mobilne zintegrowane z usługami typu SOAP są bardziej skomplikowane od wykonujących te same czynności aplikacji zintegrowanych z usługami REST;
2) rozmiar aplikacji mobilnych współpracujących z usługami typu REST jest mniejszy od programów zintegrowanych z usługami w standardzie SOAP;
3) programy mobilne zintegrowane z usługami sieciowymi typu REST, działają szybciej od programów korzystających z usług SOAP.

2. Usługi sieciowe

Zgodnie z definicją międzynarodowej organizacji World Wide Web Consortium (W3C) [2], zajmującej się opracowywaniem standardów i zasad dotyczących Internetu, webservice jest zestawem oprogramowania zaprojektowanym do wsparcia wymiany danych pomiędzy maszynami poprzez sieć, zazwyczaj z wykorzystaniem protokołu HTTP i języka XML oraz innych ustandaryzowanych technologii związanych z siecią [3].

Integracja usług wymaga użycia odpowiednich metod komunikacyjnych, które oferują efektywną integrację komponentów. Metody mają zagwarantować efektywność zarówno w działaniu i konfiguracji programów jak również w procesie ich wytwarzania. Stosowane technologie komunikacji obejmują standardy usług sieciowych RESTful Web services (RS WS) oraz XML Web services (XML WS).

Standard XML Web services w zestawieniu ze standardem RESTful Web services wyróżnia się większą złożonością konfiguracji, co wymaga od deweloperów wykorzystania skomplikowanych bibliotek, których zadaniem jest budowanie wiadomości SOAP. Z kolei prostota konfiguracji charakterystyczna dla REST, wymaga kontroli i prawidłowej obsługi zawartości przesyłanych treści [4]. Bez względu na wykorzystywane rozwiązanie, konsument i dostawca usługi muszą być zgodni co do semantyki przesyłanych danych.

3. Konfiguracja środowiska badawczego

Maszyna wirtualna pełniąca rolę serwera usług sieciowych, działa w oparciu o platformę Ubuntu Server. W ramach systemu pracuje serwer bazy danych MariaDB, który stanowi magazyn danych dla usług sieciowych. Z kolei same usługi są udostępniane za pomocą serwera Apache.

Parametry wirtualnego serwera:

- Procesor: Intel Core i5 2,4 GHz (1 rdzeń)
- Pamięć RAM: 2048 MB
- Dysk twardy: SSD (rozmiar wirtualny 20 GB).
- Karta sieciowa Wi-Fi: Intel PRO/1000 (mostkowana)

Oprogramowanie wirtualnego serwera:

- System operacyjny: Ubuntu Server 18.10
- Serwer bazy danych: MariaDB 10.1.29
- Serwer HTTP: Apache 2.4.34
- PHP: 7.2.10

Tak jak w przypadku usług sieciowych, aplikacja mobilna działa na urządzeniu wirtualnym (typu telefon), uruchamianym za pomocą menadżera Android Virtual Device (AVD), wbudowanego w środowisko Android Studio.

Parametry wirtualnego urządzenia mobilnego:

- Model telefonu: Nexsus 5X
- Rozdzielczość: 1080 x 1920: 420dpi
- System operacyjny: Android 8.1 x86 (Oreo)
- Wersja API: 27

4. Przegląd bibliotek

W przeglądzie nie uwzględniano oprogramowania o względnie małej popularności oraz bibliotek przestarzałych takich jak kSOAP, RoboSpice oraz HttpClient, zastąpionych przez nowsze oprogramowanie. Inspekcja dotyczyła zarówno bibliotek działających jako standardowy klient HTTP, jak również bibliotek specjalistycznych, zaprojektowanych do współpracy z usługami sieciowymi konkretnego rodzaju. Zestawienie nie obejmuje bibliotek takich jak Picasso czy Fresco, dedykowanych do obsługi obrazów. Wyniki przeglądu przedstawiono w tabeli 1.

Tabela 1. Biblioteki wspomagające integrację z usługami sieciowymi

Nazwa	Wersja	Rodzaj	Zastosowanie
Android HTTP	b.d.	Systemowa	Ogólne
Android Volley	1.1.1	Zewnętrzna	Ogólne
OkHttp	3.12.1	Zewnętrzna	Ogólne
Loopj	1.4.9	Zewnętrzna	Ogólne
Retrofit	2.5.0	Zewnętrzna	Spec. dla REST
kSOAP2	3.6.3	Zewnętrzna	Spec. dla SOAP

Przy uwzględnieniu aspektu praktycznego, wyselekcjonowano siedem kryteriów, na podstawie których wykonano analizę porównawczą dostępności podstawowych mechanizmów (tabela 2). Uwzględniano wyłączenie funkcje wspierane natywnie, czyli takie, które nie wymagają użycia przez dewelopera dodatkowych komponentów. Przykładowo biblioteki Android HTTP i kSOAP2, pozwalają na obsługę żądań asynchronicznych, wyłącznie po zastosowaniu klasy AsyncTask, na podstawie czego stwierdzono, że nie posiadają wbudowanej funkcji do obsługi komunikacji asynchronicznej. Wyjątkiem są rozwiązania, w których wspomniane komponenty zostały użyte wewnątrz badanej biblioteki i stanowią jej integralną cześć.

Tabela 2. Porównanie funkcjonalności bibliotek

	Żądania synchroniczne	Żądania asynchroniczne	Kolejkowanie żądań	Priorytety żądań	Ponawianie	Pule wątków	Pamięć podręczna
Android HTTP	tak	nie	nie	nie	nie	nie	nie
Volley	tak	tak	tak	tak	tak	tak	tak
OkHttp	tak	tak	tak	nie	tak	tak	tak
Loopj	tak	tak	tak	nie	tak	tak	tak
Retrofit	tak	tak	tak	nie	tak	tak	tak
kSOAP2	tak	nie	nie	nie	nie	nie	-

Funkcja ponawiania jest istotna ze względu na pewność komunikacji i umożliwia ponowne wysyłanie żądań, w przypadku problemów z połączeniem sieciowym. Z kolei mechanizm ustawiania priorytetów ustala kolejność obsługi żądań, zaś pule wątków zapewniają ich równolegle przetwarzanie zwiększając wydajność programów. Biblioteki korzystające z pamięci podręcznej, umożliwiają tymczasowe przechowywanie danych w pamięci urządzenia, dzięki czemu mogą być one ponownie użyte, co skraca czas dostępu do

danych (brak informacji dla kSOAP2). Z punktu widzenia użyteczności, ważne jest również zapewnienie obsługi żądań asynchronicznych, które nie są wykonywane w głównym wątku programu oraz żądań synchronicznych, mających zastosowanie w procesach wykonywanych sekwencyjnie.

W wyniku przeglądu do przeprowadzenia badań wydajności i złożoności implementacyjnej **zakwalifikowano biblioteki Volley, Retrofit oraz kSOAP2**. Rozwiązania wyselekcjonowano na podstawie ich popularności, która wynika zarówno z funkcjonalności jak również ze specjalistycznego przeznaczenia, które gwarantuje łatwość integracji z usługami REST i SOAP. Volley ma szerokie zastosowanie ze względu rozbudowaną funkcjonalność, z kolei Retrofit oferuje wsparcie dla technologii REST, która zyskuje na popularności wraz z szybkim rozwojem aplikacji mobilnych. Tymczasem kSOAP2 jest jedyną biblioteką na platformę Android liczącą się na rynku, zaprojektowaną do współpracy z usługami typu SOAP.

5. Metryki oprogramowania

Analizę porównawczą bibliotek wykonano w oparciu o metryki dynamiczne i metryki statyczne. Metryka jest liczbową charakterystyką oprogramowania pozwalająca na ilościową ocenę produktów (kodu źródłowego, zestawu testów itd.), procesów wytwarzania oprogramowania oraz ludzi biorących udział w procesach wytwórczych [5]. Zgodnie ze standardem IEEE 1061-1998 metryka jest funkcją odwzorowującą jednostkę oprogramowania w wartość liczbową, która jest intepretowana jako stopień spełnienia pewnej własności jakości jednostki oprogramowania [6]. Metryki pozwalają w sposób obiektywny porównać ze sobą poszczególne technologie i umożliwiają ocenę czy produkt odpowiada ustalonym względem niego wymaganiom. Wśród nich funkcjonuje elementarny podział na metryki dynamiczne i statyczne. Metryki statyczne pozwalają na ocenę jakości kodu źródłowego już na etapie tworzenia oprogramowania, a także kontrolę jego złożoności. Wśród metryk statycznych można wyodrębnić: metryki obiektowe, rozmiaru, złożoności oraz pakietów. Pierwsze z nich badają jakość programów obiektowych oraz wykorzystanie właściwości programowania zorientowanego obiektowo. Z kolei metryki rozmiaru i złożoności służą do określenia skali programu oraz jego złożoności na różnych poziomach abstrakcji. Zadaniem metryk pakietów jest analiza prawidłowości powiązań pomiędzy poszczególnymi pakietami. Metryki dynamiczne, wyznaczane w oderwaniu od kodu źródłowego, polegają na analizie zachowania uruchomionego programu. Są stosowane do pomiaru wydajności i niezawodności produktu. Metryki stały się integralną częścią procesu produkcji oprogramowania i są wykorzystywane na każdym etapie jego rozwoju [7].

5.1. Metryki statyczne

W rozdziale przedstawiono metryki stosowane do pomiaru skomplikowania aplikacji klienckich korzystających z bibliotek Volley, Retrofit i kSOAP2.

Liczba linii kodu (LOC) – podstawowa metryka statyczna do pomiaru liczby linii kodu źródłowego, mówiąca o skali badanego oprogramowania i stosowana jako wskaźnik klas i metod przekraczających zalecany rozmiar.

Liczba klas (TLC) – prosta metryka rozmiaru zdefiniowana dla pakietów, dostarczająca dane o całkowitej liczbie klas z wyłączeniem klas wewnętrznych, której zalecana wartość jest mniejsza od 40 [8].

Złożoność cyklomatyczna McCabe'a (CC) – podstawowa metryka złożoności ozn. CC lub v(G), wykorzystywana do analizy struktury decyzyjnej programu używana do obliczania złożoności metod lub procedur. Niski poziom CC charakteryzuje prosty i zrozumiały kod, natomiast wartości powyżej 10, charakteryzują złożony kod źródłowy, w którym ryzyko wystąpienia błędu jest wysokie [9].

Zestaw metryk Chidamber-Kemerer – metryki obiektowe opracowane przez Shyama R. Chidambera i Chrisa F. Kemerera, nazywane również zestawem CK, zostały zaprojektowane do badania złożoności i utrzymywalności oprogramowania obiektowego [10]. Zestaw oryginalnie zawiera 6 metryk wyliczanych dla każdej klasy:

- **ważona liczba metod w klasie (WMC)** – uproszczona wersja metryki, której wartość stanowi suma metod zdefiniowanych w analizowanej klasie. Rekomendowany maksymalny próg wartości metryki wynosi 20 [8];
- **głębokość drzewa dziedziczenia (DIT)** – najdłuższa ścieżka dziedziczenia od badanej klasy do klasy korzenia. Drzewa o dużej głębokości charakteryzują złożone klasy, o większym stopniu specjalizacji [8];
- **liczba bezpośrednich potomków klasy (NOC)** – liczba bezpośrednich potomków klasy, czyli podklas. Zbyt duża wartość NOC, może wynikać z nieprawidłowego użycia mechanizmu dziedziczenia [8];
- **powiązania między klasami (CBO)** – liczba klas, które są związane z analizowaną klasą. Niski poziom CBO ułatwia wprowadzanie zmian w programie, w wyniku mniejszej liczy zależności pomiędzy klasami, co przekłada się na niższy stopień złożoności [8];
- **odpowiedź klasy (RFC)** - miara potencjalnej komunikacji między klasą a innymi klasami. Wysokie wartości RFC wskazują na dużą funkcjonalność i złożoność klasy [8];
- **brak spójności metod (LCOM)** – miara poziomu spójności metod. Zbyt wysokie wartości metryki wskazują klasy o dużej złożoności, które powinny zostać podzielone na mniejsze, pełniące mniej obowiązków. Do realizacji prac badawczych wybrano miarę w wersji LCOM1, która jest jedną z kilku z odmian metryki LCOM [8].

Rozmiar programu – miara polega na pomiarze liczby plików tworzących projekt oraz rozmiaru programu zapisanego w formie pliku APK, wyrażanego w KB. Dla każdego wariantu aplikacji rozmiar mierzono na tej samej maszynie, zapisując APK na wolumenie z systemem plików NTFS, aby wyeliminować wpływ różnych systemów plików na odczytywany rozmiar danych.

5.2. Metryki dynamiczne

W rozdziale przedstawiono metryki stosowane do oceny dynamiki aplikacji implementujących badane biblioteki.

Całkowity czas przetwarzania (TPT) – miara wyrażana w milisekundach, przedstawiająca czas jaki upłynął pomiędzy wysłaniem żądania przez aplikację mobilną do odczytania przez nią przetworzonej odpowiedzi.

Czas przetwarzania przez bibliotekę (PT) – miara wydajności bibliotek, wyrażana w milisekundach, prezentująca różnicę pomiędzy całkowitym czasem odpowiedzi, a sumą czasu odpowiedzi usługi sieciowej oraz sieci komputerowej, wyznaczana wzorem:

$$PT = TPT - (SRT + NRT) \quad (1)$$

gdzie: PT – czas przetwarzania przez bibliotekę, TPT – całkowity czas przetwarzania, SRT – czas odpowiedzi usługi, NRT – czas odpowiedzi sieci.

Pomiar polega na obliczeniu opóźnień wynikających z pracy badanych bibliotek z uwzględnieniem opóźnień wynikających z wydajności sieci komputerowej i usług sieciowych.

Czas odpowiedzi usługi (SRT) – metryka stosowana do pomiaru czasu reakcji serwera, polegająca na pomiarze czasu jaki upłynął od wysłania żądania do usługi sieciowej do momentu odebrania pierwszego bajtu odpowiedzi. Badanie przeprowadzono przy użyciu narzędzia Wireshark umożliwiającego przechwytywanie i nagrywanie ruchu sieciowego oraz programu Response Time Viewer for Wireshark służącego do analizy i wizualizacji czasów odpowiedzi na podstawie zarejestrowanych pakietów danych. W aplikacji analitycznej, czas odpowiedzi usługi jest określany jako czas odpowiedzi aplikacji i funkcjonuje pod nazwą Appliacation Response Time w skrócie ART.

Czas odpowiedzi sieci (NRT) – metryka pozwalająca na zmierzenie opóźnień wynikających z działania sieci komputerowej, wrażanych jako czas odpowiedzi sieci mierzony w milisekundach (uzgodnienie TCP). Tak jak w przypadku SRT, badanie przeprowadzono przy użyciu narzędzia Wireshark oraz programu Response Time Viewer for Wireshark do analizy zarejestrowanych pakietów danych.

6. Metody badawcze

6.1. Metoda pomiaru złożoności

Pomiar złożoności oprogramowania, wykonano z użyciem różnego typu metryk statycznych, przedstawionych w rozdziale 5.1. Poza tym porównano rozmiar programów, mierząc liczbę plików tworzących projekt i zajętość dysku twardego dla programów w formie pliku APK. Pomiary realizowano wieloetapowo, badając niezależnie złożoność programów implementujących biblioteki Volley, Retrofit i kSOAP2. Narzędzia i rozszerzenia wykorzystane podczas pomiarów zostały opisane w ramach poszczególnych etapów. Miary użyte w trakcie badań, wyznaczają poszczególne etapy badawcze, które opisano szczegółowo w dalszej części rozdziału. Metoda badawcza, zawiera opis czynności wykonywanych podczas realizacji poszczególnych etapów i prowadzi do uzyskania wyników niezbędnych do przeprowadzenia analizy porównawczej.

Etap 1 – Liczba linii kodu

Pomiar liczby linii kodu źródłowego (LOC) przeprowadzono z poziomu środowiska programistycznego Android Studio z użyciem rozszerzenia MetricsReloaded w wersji 1.8. Miarę zastosowano do całego projektu, ale LOC odczytywano dla pakietu realizującego najważniejsze funkcje programu, uwzględniając wszystkie tworzące go klasy i interfejsy oraz kod wykonywalny wygenerowany automatycznie przez IDE. Z pomiarów wykluczono jedynie kod źródłowy, odpowiedzialny za automatyczne testowanie aplikacji. Pomiar LOC dla pakietów grupujących klasy tworzące np. GUI lub realizujące inne funkcje niezwiązane bezpośrednio z komunikacją z usługą sieciową nie jest istotny z punktu widzenia analizy porównawczej. W trakcie wykonywania prac badawczych w oknie ustawień MetricsReloaded, wybrano profil Lines of code metrics, dedykowany do tego typu pomiarów.

Etap 2 – Liczba klas

Pomiar liczby klas przeprowadzono z poziomu IDE z użyciem rozszerzenia MetricsReloaded, analogicznie jak w przypadku metryki LOC. Miarę zastosowano do całego projektu, ale liczbę klas odczytywano dla jednego pakietu, pomijając klasy przeznaczone do wykonywania testów automatycznych. W trakcie wykonywania prac badawczych w oknie ustawień MetricsReloaded, wybrano profil Class count metrics. Liczbę klas odczytano dla pakietu, zawierającego kod źródłowy odpowiedzialny za użycie biblioteki i komunikację z usługą sieciową.

Etap 3 – Złożoność cyklomatyczna

Badanie złożoności cyklomatycznej wykonano z użyciem rozszerzania MetricsReloaded, wykorzystując profil Complexity metrics. Tak jak w przypadku innych użytych miar, wykluczono z analizy klasy i metody odpowiedzialne za testowanie programu. Badanie polegało na pomiarze średniej złożoności oprogramowania dla pakietu zawierającego kod źródłowy odpowiedzialny za komunikację z API.

Etap 4 – Zestaw metryk Chidamber-Kemerer

Do obliczenia wartości 6 metryk tworzących zestaw CK, również użyto rozszerzanie MetricsReloaded, wykorzystując profil Chidamber-Kemerer metrics. Tak jak w przypadku innych miar, wykluczono z analizy klasy i metody odpowiedzialne za testowanie programu. Pomiar wykonano dla klas wykorzystywanych do integracji z usługą sieciową, wymaganych do prawidłowego działania komunikacji. Ze względu na różnice w badanym zbiorze klas wynikające z właściwości bibliotek, nie porównywano ze sobą konkretnych klas, ale średnie wartości metryk.

Etap 5 – Rozmiar programu

Tak jak w poprzednich etapach do pomiaru ilości plików zastosowano MetricsReloaded, w którym wybrano profil Number of files metrics. Z analizy wykluczono klasy, przeznaczone do wykonywania testów automatycznych.

W pomiarach uwzględniono wszystkie pliki projektu, również te które nie są bezpośrednio związane z wykorzystaniem API. W praktyce tylko niewielka ich część zależy od rodzaju biblioteki zastosowanej w projekcie. Zajętość programu na dysku, mierzono zapisując pliki APK na partycji z systemem plików NTFS i odczytując ich rozmiar z właściwości pliku.

6.2. Metoda pomiaru wydajności

Programy mobilne implementujące badane biblioteki, współpracują z usługami sieciowymi typu REST i SOAP i mierzą całkowity czas przetwarzania dla trzech rodzajów wywołań. Każdy z programów został zbudowany w oparciu o ten sam szablon projektu oraz identyczne GUI, zapewniające użytkownikowi funkcje potrzebne do przeprowadzenia pomiarów. Zasadniczo różnice w konstrukcji kodu źródłowego, wynikają wyłącznie ze specyfiki używanej biblioteki i rodzaju API z którego korzysta program. Każda odmiana aplikacji zapewnia taką samą funkcjonalność, ograniczoną do funkcji niezbędnych do wykonania badań i posiada prosty interfejs prezentujący wyniki pomiarów.

Program kliencki obsługuje trzy rodzaje wywołań i dla każdego z nich oblicza całkowity czas przetwarzania, mierzony do mementu zamiany odpowiedzi serwera na obiekty Java. Poza tym w trakcie pomiarów zlicza wysłane żądania oraz oblicza średni czas ich obsługi. Po wciśnięciu przycisku, wywołana jest metoda odpowiedzialna za wysłanie nowego żądania oraz odebranie i konwersję odpowiedzi. Program kliencki obsługuje następujące żądania:

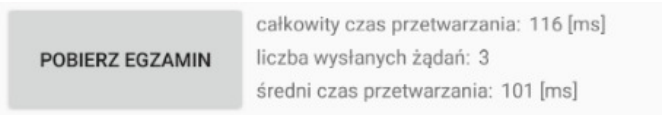
- **pobierz egzamin** – program wysyła do serwera identyfikator egzaminu, którego dane chce pobrać z serwera. Po otrzymaniu prawidłowego żądania usługa wysyła odpowiedź zawierającą wszystkie dane egzaminu w ustalonym formacie. Po odebraniu, odpowiedź jest konwertowana do postaci obiektów Java odpowiadających strukturom danych, zawartym w odpowiedzi;
- **pobierz egzaminy** – program wywołuje akcję pobierającą z bazy danych egzaminy i w odpowiedzi odbiera listę wszystkich egzaminów zdawanych przez studentów. Następnie odpowiedź jest zamieniana na kolekcję lub wektor elementów, zawierający obiekty Java;
- **dodaj studenta** – w przypadku tego żądania aplikacja wysyła do serwera obiekt reprezentujący studenta, który odebrany przez usługę sieciową, zostaje dodany w formie nowego rekordu do tabeli przechowującej dane studentów.

Badanie wydajności bibliotek w oparciu o metryki dynamiczne jest procesem wieloetapowym, wymagającym użycia dodatkowych narzędzi pomiarowych. Metoda badawcza zawiera opis czynności wykonywanych podczas realizacji poszczególnych etapów i prowadzi do uzyskania wyników niezbędnych do przeprowadzenia późniejszej analizy porównawczej. W ramach każdego etapu, wykonywane są pomiary całkowitego czasu przetwarzania (TPT), czasu odpowiedzi usługi i czasu odpowiedzi sieci na podstawie których obliczany jest faktyczny czas przetwarzania danych przez bibliotekę. TPT jest mierzony dla każdej akcji niezależenie przez program kliencki, który dodatkowo zlicza wysłane żądania i oblicza średni czas ich przetwarzania.

Procedurę badawczą podzielono na trzy niezależne etapy i w każdym z nich przeprowadzono pomiar wydajności jednej biblioteki. Każdy z etapów realizowano w taki sam sposób, bez względu na to jaką zastosowano bibliotekę lub jaki rodzaj usługi sieciowej wykorzystywał klient mobilny. Zaprezentowana sekwencja czynności dotyczy wyłączenie akcji jednego rodzaju, ale ma zastosowanie dla pozostałych dwóch. Praktyczne aspekty metody badawczej opisano w sposób uniwersalny, mającym zastosowanie na każdym etapie badań. W celu zwiększenia precyzji pomiarów w ramach prac badawczych analizowano TPT dla dziesięciu żądań każdego rodzaju. W ramach poszczególnych etapów wyodrębniono kroki konieczne do uzyskania żądanych wyników, przedstawione w dalszej części rozdziału.

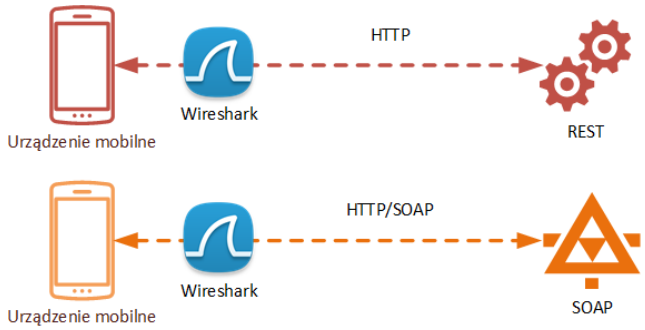
Krok 1 – Pomiar TPT

W pierwszym kroku zmierzono całkowity czas przetwarzania dla wywołania pobierającego dane egzaminu o wskazanym identyfikatorze. Po kliknięciu przycisku „Pobierz egzamin”, program mobilny wysyła żądanie do usługi sieciowej i otrzymaną odpowiedź konwertuje do postaci obiektów Java. W GUI prezentuje całkowity czas przetwarzania, średni czas przetwarzania i liczbę żądań wysłanych od rozpoczęcia pomiaru (rysunek 1). TPT jest obliczany dla każdego żądania niezależnie, natomiast średni czas przetwarzania i liczba wysłanych żądań są obliczane dla całego cyklu pomiarowego.



Rys. 1. Wynik pomiaru TPT dla wywołania pobierającego dane egzaminu

Procesem wykonywanym w trakcie wysyłania żądań jest nagrywanie ruchu sieciowego, za pomocą programu Wireshark. Jego zadaniem jest przechwytywanie danych wymienianych z użyciem protokołu HTTP, pomiędzy klientem mobilnym, a usługą sieciową (rysunek 2).



Rys. 2. Nagrywanie ruchu sieciowego pomiędzy klientem i usługą sieciową

Krok 2 – Pomiar SRT i NRT

Pomiar czasu odpowiedzi usługi i czasu odpowiedzi sieci komputerowej wymagał zastosowania narzędzia analitycznego, którego zadaniem było odczytanie tych parametrów na podstawie zarejestrowanego ruchu sieciowego.

Do tego celu wykorzystano aplikację SolarWinds Response Time Viewer, która umożliwia analizę pakietów przechwyconych przez program Wireshark. Pliki z ruchem sieciowym otwierano w SolarWinds Response Time Viewer i poddawano analizie, w wyniku której odczytywano czas odpowiedzi sieci (NRT) i czas odpowiedzi usługi (SRT) dla protokołów HTTP lub SOAP (tylko dla kSOAP2).

Krok 3 – Obliczenie PT

Czas przetwarzania przez bibliotekę (PT), obliczono na podstawie wcześniej zmierzonych parametrów TPT, SRT, NRT dla każdego wywołania niezależnie. Poza tym obliczono średni czas przetwarzania dla dziesięciu wywołań tego samego rodzaju, czyli dla jednego cyklu pomiarowego. Eliminacja opóźnień wynikających z czasu odpowiedzi sieci i usług sieciowych, zimniejsza błąd pomiarowy i pozwala na porównanie rzeczywistych czasów przetwarzania.

7. Wyniki badań

7.1. Wyniki pomiaru złożoności

Metryki rozmiaru użyte w metodzie badawczej przedstawiają różnice w skali programów klienckich implementujących poszczególne biblioteki. Ze względu na zróżnicowanie w konstrukcji programów nie porównywano LOC dla poszczególnych klas, zaś dla całych pakietów Java dla których nie ma rekomendowanych wartości tej metryki. Poza TLC, również pozostałe metryki rozmiaru nie posiadają zalecanych zakresów wartości i posłużono się nimi wyłącznie na potrzeby analizy porównawczej. Tabela 3 przedstawia wartości metryk rozmiaru z zaznaczonymi wartościami największymi (ozn. ▲) oraz najmniejszymi (ozn. ▼).

Tabela 3. Porównanie wartości metryk rozmiaru

Metryka	Wartość zalecana	Volley	Retrofit	kSOAP2
LOC_{sum}	-	468	441 ▼	771 ▲
TLC	< 40	18 ▲	15	13 ▼
Rozmiar w KB	-	1967 ▲	1928	1840 ▼
Liczba plików	-	27	28 ▲	26 ▼

W przypadku metryk CC i WMC, otrzymane wyniki pomiarów porównano z wartościami rekomendowanymi. Dla pozostałych metryk z zestawu CK, literatura mówi wyłącznie o wartościach wysokich (ozn. ↑) lub niskich (ozn. ↓), nie określając konkretnie zalecanych zakresów wartości miar. Tabela 4 przedstawia uśrednione wartości metryk obiektowych i złożoności z zaznaczonymi wartościami największymi (ozn. ▲) oraz najmniejszymi (ozn. ▼).

Tabela 4. Porównanie wartości metryk obiektowych i złożoności

Metryka	Wartość zalecana	Volley	Retrofit	kSOAP2
$CC_{śr}$	< 10	1,08	1,05 ▼	3 ▲
$WMC_{śr}$	< 20	10.86	9,86 ▼	13,44 ▲
$DIT_{śr}$	↓	2,43 ▲	2 ▼	2,11
$NOC_{śr}$	↓	0	0	0
$CBO_{śr}$	↓	2,43	3 ▲	2,33 ▼
$RFC_{śr}$	↓	11,29 ▲	10,38	9,89 ▼
$LCOM_{śr}$	↓	3,57	3,71 ▲	2,44 ▼

7.2. Wyniki pomiaru wydajności

Tabela 5. Pomiary wydajności dla Retrofit i akcji "Pobierz egzamin"

Retrofit - Pobierz egzamin						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	201	82	14	105	100	94
2	205	71	29	105		
3	214	71	11	132		
4	172	71	11	90		
5	200	80	23	97		
6	181	76	19	86		
7	274	130	59	85		
8	231	74	25	132		
9	174	71	15	88		
10	176	75	24	77		

Tabela 6. Pomiary wydajności dla Retrofit i akcji "Pobierz egzaminy"

Retrofit - Pobierz egzaminy						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	349	15	21	313	363	385
2	302	15	9	278		
3	327	14	23	290		
4	339	15	16	308		
5	476	15	36	425		
6	459	15	28	416		
7	422	16	18	388		
8	416	15	19	382		
9	418	14	12	392		
10	468	16	10	442		

Tabela 7. Pomiary wydajności dla Retrofit i akcji "Dodaj studenta"

Retrofit - Dodaj studenta						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	192	83	24	85	88	87
2	153	68	13	72		
3	168	66	31	71		
4	162	75	15	72		
5	237	69	66	102		
6	203	73	28	102		
7	209	72	20	117		
8	162	66	13	83		
9	160	61	10	89		
10	173	68	14	91		

Tabela 8. Pomiary wydajności dla Volley i akcji "Pobierz egzamin"

Volley - Pobierz egzamin						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	254	70	36	148	163	157
2	217	69	11	137		
3	221	71	17	133		
4	285	65	23	197		
5	264	67	14	183		
6	257	67	24	166		
7	222	70	20	132		
8	288	68	26	194		
9	279	67	7	205		
10	217	71	9	137		

Tabela 9. Pomiary wydajności dla Volley i akcji "Pobierz egzaminy"

Volley - Pobierz egzaminy						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	416	15	14	387	443	444
2	420	15	15	390		
3	577	14	42	521		
4	479	16	10	453		
5	462	16	11	435		
6	485	15	9	461		
7	581	14	52	515		
8	426	14	25	387		
9	548	16	18	514		
10	406	14	22	370		

Tabela 10. Pomiary wydajności dla Volley i akcji "Dodaj studenta"

Volley - Dodaj studenta						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	133	65	16	52	75	79
2	153	65	15	73		
3	134	62	15	57		
4	125	62	13	50		
5	172	67	11	94		
6	168	64	18	86		
7	170	63	16	91		
8	175	68	11	96		
9	143	73	9	61		
10	168	62	10	96		

Tabela 11. Pomiary wydajności dla kSOAP2 i akcji "Pobierz egzamin"

kSOAP2 - Pobierz egzamin						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	190	74	22	94	81	86
2	179	70	25	84		
3	166	67	28	71		
4	188	72	17	99		
5	160	71	19	70		
6	171	70	22	79		
7	119	69	18	32		
8	196	71	23	102		
9	200	78	33	89		
10	177	67	16	94		

Tabela 12. Pomiary wydajności dla kSOAP2 i akcji "Pobierz egzaminy"

kSOAP2 - Pobierz egzaminy						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	890	185	20	685	652	655
2	872	179	36	657		
3	781	204	28	549		
4	882	193	35	654		
5	907	192	21	694		
6	899	202	23	674		
7	874	198	26	650		
8	931	190	43	698		
9	878	195	38	645		
10	880	225	37	618		

Tabela 13. Pomiary wydajności dla kSOAP2 i akcji "Dodaj studenta"

kSOAP2 - Dodaj studenta						
Nr żądania	TPT [ms]	SRT [ms]	NRT [ms]	PT [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
1	163	65	37	61	71	163
2	161	64	36	61		
3	188	66	38	84		
4	171	61	30	80		
5	160	58	35	67		
6	189	69	31	89		
7	141	58	25	58		
8	189	60	39	90		
9	159	67	28	64		
10	151	63	30	58		

Na bazie wyników pomiarów, obliczono przeciętny (mediana) i średni czas przetwarzania, zgodnie z metodami badawczymi. Tabela 14 przedstawia zestawienie tych czasów z zaznaczonymi wartościami największymi (ozn. ▲) oraz najmniejszymi (ozn. ▼) dla poszczególnych wywołań.

Tabela 14. Porównanie przeciętnych i średnich czasów przetwarzania

Wywołanie	Volley		Retrofit		kSOAP2	
	$PT_{śr}$ [ms]	PT_{me} [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]	$PT_{śr}$ [ms]	PT_{me} [ms]
Pobierz egzamin	100	94	163 ▲	157 ▲	81 ▼	86 ▼
Pobierz egzaminy	363 ▼	358 ▼	443	444	652 ▲	655 ▲
Dodaj studenta	88 ▲	87 ▲	75	79	71 ▼	65 ▼

8. Wnioski

8.1. Złożoność oprogramowania

Na podstawie badań złożoności przeprowadzonych dla bibliotek Retrofit, Volley i kSOAP2, można sformułować następujące wnioski. Metryki rozmiaru wykazały, że program korzystający z biblioteki kSOAP2 ma największy rozmiar pod względem liczby linii kodu źródłowego, zaś najmniejszy pod względem liczby klas, zajmowanej przestrzeni dyskowej po zbudowaniu APK oraz liczby plików w projekcie. Poziom TLC dla żadnej biblioteki nie przekracza wartości rekomendowanej, więc analizowane pakiety nie wymagają podziału. Z kolej program implementujący bibliotekę Volley jest największy pod względem rozmiaru pliku APK i liczby użytych klas. Program oparty na Retrofit, posiada najwięcej plików w projekcie i najmniejszą sumaryczną wartość LOC. Ze względu na najwyższą wagę metryki LOC i relatywnie małe różnice w wartościach pozostałych metryk rozmiaru stwierdzono, że najbardziej rozbudowany jest program korzystający z kSOAP2. Wysoki poziom LOC przekłada się na zwiększenie pracochłonności i czasochłonności procesu implementacji programu. Analiza porównawcza potwierdza słuszność hipotezy 1, mówiącej że programy mobilne zintegrowane z usługami REST, mają mniejszy rozmiar od aplikacji zintegrowanych z usługami SOAP.

Złożoność cyklomatyczna McCabe'a (CC) oraz ważona liczby metod w klasie (WMC), mieszczą się w zalecanych zakresach wartości, kształtując się na względnie niskim poziomie. Metryki osiągają wartości najwyższe dla biblioteki

kSOAP2, co oznacza, że posiada ona średnio najwięcej metod w klasie i są one najbardziej skomplikowane, co w praktyce przekłada się na większe prawdopodobieństwo wystąpienia błędu. Z kolei najniższe wartości miar CC i WMC, zmierzono dla biblioteki Retrofit, dla której poziom skomplikowania metod okazał się blisko trzykrotnie mniejszy niż dla kSOAP2, zaś liczba metod w klasie mniejsza w przybliżeniu o cztery. Dla Volley wartości obu metryk były porównywalne z wartościami zmierzonymi dla biblioteki Retrofit. Wartość stopnia dziedziczenia (DIT) dla wszystkich bibliotek była zbliżona, z czego najwyższa dla Volley, zaś najniższa dla Retrofit. Z kolei średnia liczba podklas (NOC) dla wszystkich badanych bibliotek była równa zeru, ze względu na niestosowanie mechanizmu dziedziczenia w analizowanym pakiecie. Również średnie wartości metryk obiektowych CBO i RFC były bardzo zbliżone, na podstawie czego stwierdzono że klasy tworzące programy klienckie mają podobną funkcjonalność oraz zdolność do wprowadzania zmian. Jedynie poziom spójności metod (LCOM) zmierzony dla kSOAP2, różnił się w sposób zauważalny od wartości LCOM dla pozostałych bibliotek i był w przybliżeniu o 1/3 mniejszy. Taka dysproporcja wskazuje na mniejszą liczbę obowiązków, a co za tym idzie mniejszą złożoność implementacyjną klas użytych w oprogramowaniu implementującym kSOAP2. Podsumowując, ze względu na duże wartości CC i WMC w porównaniu z Volley i Retrofit oraz zbliżone wartości pozostałych metryk z wyjątkiem LCOM, stwierdzono że najwyższy poziom złożoności charakteryzuje programy oparte na kSOAP2. Z kolei aplikacje korzystające z Volley i Retrofit mają podobną złożoność i nie można jednoznacznie wskazać, która technologia charakteryzuje programy bardziej skomplikowane. Wyniki badań potwierdzają słuszność hipotezy 2, mówiącej że programy mobilne współpracujące z usługami SOAP są bardziej skomplikowane od wykonujących te same czynności aplikacji zintegrowanych z usługami REST.

8.2. Wydajność programów klienckich

Na podstawie badań wydajności bibliotek zaobserwowano, że szybkość działania programów mobilnych, zintegrowanych z usługami sieciowymi, zależy zarówno od typu wykorzystywanej biblioteki, jak również od rodzaju wywoływanej akcji. Uśredniona wartość PT dla wywołania pobierającego dane egzaminu, okazała się najmniejsza dla biblioteki kSOAP2, zaś największa dla Volley. W przypadku pobierania większej porcji danych w postaci 100 egzaminów, najszybsza okazuje się biblioteka Retrofit, natomiast czas przetwarzania danych jest najdłuższy dla kSOAP2. Właśnie dla wywołań pobierających dane, różnice w wartościach PT między bibliotekami są największe i sięgają blisko 300 [ms] dla wywołania pobierającego wiele egzaminów. Czasy zmierzone dla najszybszej biblioteki są blisko dwukrotnie mniejsze niż dla biblioteki najwolniejszej. Inaczej jest w przypadku wywołania dodającego studenta do systemu, gdzie czasy przetwarzania bibliotek są porównywalne, przy czym średnia wartość PT jest największa dla Retrofit i najmniejsza dla kSOAP2. Celem pracy było zbadanie prawdziwości stwierdzenia, że programy zintegrowane z usługami typu REST, działają szybciej od aplikacji współpracujących z usługami typu SOAP. Z uwagi na to, że największe opóźnienia występują podczas pobierania wielu egzaminów oraz zważywszy na to że rozmiar danych pojedynczego egzaminu w mniejszym stopniu odpowiada rozmiarom danych pobieranych w rzeczywistych systemach, stwierdzono że najniższy poziom wydajności charakteryzuje kSOAP2. Z kolei aplikacje korzystające z Volley i Retrofit mają zbliżoną wydajność i analogicznie jak w przypadku analizy złożoności nie można jednoznacznie wskazać, która technologia cechuje programy o większej wydajności. Nie mniej jednak uzyskane wyniki potwierdzają hipotezę 3, że wydajność aplikacji mobilnych zintegrowanych z usługami typu REST jest większa, niż programów korzystających z usług SOAP.

Teza mówiąca o tym że usługi sieciowe typu REST są lepszym rozwiązaniem od usług SOAP w przypadku korzystania z nich poprzez mobilne aplikacje klienckie okazała się prawdziwa. Programy korzystające z usług typu REST, charakteryzują się mniejszym rozmiarem, a co za tym idzie ich przygotowanie wymaga mniejszego nakładu pracy i jest mniej czasochłonne. Poza tym na niekorzyść programów zintegrowanych z SOAP przemawia większa złożoność implementacyjna, która zwiększa ryzyko wystąpienia błędu, zmniejsza czytelność kodu i utrudnia proces testowania. Za stosowaniem integracji z REST przemawia również większa szybkość działania bibliotek Retrofit i Volley w porównaniu z biblioteką kSOAP2 przeznaczoną do integracji z SOAP.

Literatura

[1] Mobile Operating System Market Share Worldwide, http://gs.statcounter.com/os-market-share/mobile/worldwide [20.02.2019].

[2] W3C Organization, http://www.w3.org [02.11.2018].

[3] Web Services Architecture, http://www.w3.org/TR/ws-arch/ [12.12.2018].

[4] B. P. Upadhyaya, Rest client pattern. Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE), IEEE, 2014.

[5] K. Subieta, Słownik terminów z zakresu obiektowości, Wyd. 1, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1999

[6] Software & Systems Engineering Standards Committee, 1061-1998 - IEEE Standard for a Software Quality Metrics Methodology, IEEE, 1998.

[7] M. Shereshevsky, R. Gunnalan, H. H. Ammar, Pseudo dynamic metrics [software metrics], The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.

[8] S. H. Kan, Metryki i modele w inżynierii jakości oprogramowania, PWN, 2006.

[9] T. McCabe, A Software Complexity Measure, IEEE Transactions on Software Engineering, 1976.

[10] S. R. Chidamber, C. F. Kemerer, A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 1994.