POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Robotyka (ARR)

PRACA DYPLOMOWA INŻYNIERSKA

System pisania testów na platformie Android

A test writing system on the Android platform

AUTOR:
Filip Malinowski

PROWADZĄCY PRACĘ:
dr Witold Paluszyński, K-8

OCENA PRACY:

WROCŁAW 2016

Spis treści

Wstęp

Tematem pracy jest stworzenie systemu na platformę Android umożliwiającego pisanie testów na wykładach akademickich i zautomatyzowane przetworzenie wyników testów. W skład systemu wchodzą: aplikacja na system Android oraz aplikacja serwerowa. Aplikacje na systemach Android mają łączyć się z serwerem i przesyłać informacje o wykonanych testach, serwer ma przetwarzać dane i magazynować je. System ma być niezawodny.

Rozdział 1

Analiza problemu

1.1 Systemy pisania testów w czasie rzeczywistym

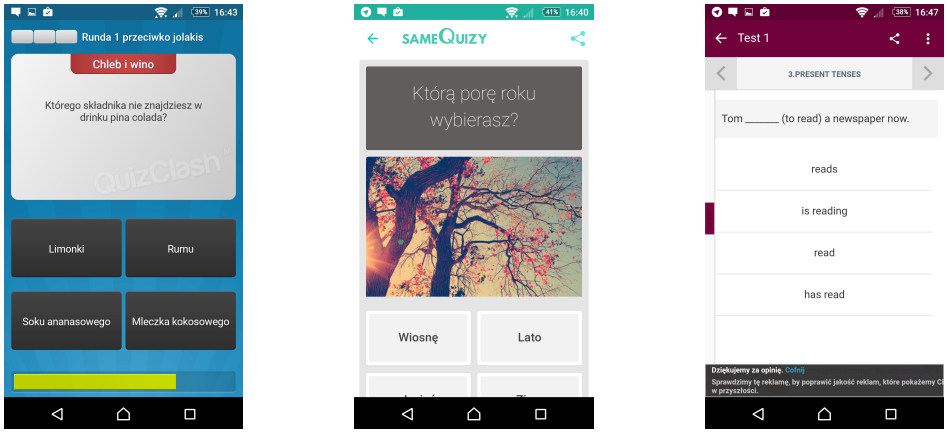
Na rynku istnieją już aplikacje częściowo odpowiadające tematowi pracy inżynierskiej. Ich funkcjonalność spełnia pewne aspekty pożądanego systemu jakimi są: wybór odpowiedzi do pytań na ekranie telefonu, przesyłanie odpowiedzi do serwera oraz pobieranie informacji z serwera na aplikację. Najlepszym przykładem jest aplikacja nazywająca się Quizowanie[1]. Aplikacja ta pozwala na pobieranie i wyświetlenie pytań z serwera, wyświetlenie możliwych odpowiedzi jako przycisków dla użytkownika oraz wysłanie odpowiedzi do serwera i zweryfikowanie ich. Quizowanie jest aplikacją nastawioną na współzawodnictwo pomiędzy użytkownikami, którzy wspólnie odpowiadają na te same pytania zdobywając przy tym punkty. Kod źródłowy tej aplikacji jest zamknięty.

Inne aplikacje, które można wymienić to English Grammar Test[2] lub też sameQuizy[3]. English Grammar Test jest aplikacją bardzo rozbudowaną i ukierunkowaną na prostą naukę języka Angielskiego. Schematy odpowiadania na pytania, wybór odpowiedzi oraz poruszanie się po menu aplikacji są zbliżone do tych, jakie występują w aplikacji Quizowanie. W sameQuizy użytkownik, tak jak w poprzednich aplikacjach, w prosty sposób poprzez naciskanie przycisków wybiera odpowiedzi na zadane na ekranie aplikacji pytania. Kod źródłowy obu tych aplikacji jest zamknięty.

Każda z trzech aplikacji wyróżnia się jednak innym podejściem przy tworzeniu interfejsu użytkownika. English Grammar Test ma prosty i czytelny wygląd. Interfejs aplikacji sameQuizy oprócz wyświetlania pytań i propozycji odpowiedzi, urozmaicony jest wieloma obrazami przyciągającymi wzrok. Interfejs aplikacji Quizowanie ma interfejs pośredni pomiędzy tym jaki można zobaczyć w English Grammar Test i sameQuizy.

Interfejs odpowiada zastosowaniu aplikacji. Aplikacje ukierunkowane na rozrywkę mają kolorowy, zwracający uwagę wygląd. Aplikacje ukierunkowane na naukę mają interfejs prosty i czytelny, a te przeznaczone do gier i rywalizacji mają interfejs bardziej urozmaicony, ale w stopniu takim, który nie pogarsza czytelności treści wyświetlanych na ekranie.

[1] https://play.google.com/store/apps/details?id=se.feomedia.quizkampen.pl.lite
[2] https://play.google.com/store/apps/details?id=english.grammar.test.app
[3] https://play.google.com/store/apps/details?id=pl.filing.samequizy



(a) Quizowanie (b) sameQuizy (c) English Grammar Test

Rysunek 1.1: Przykładowe pytania wraz z odpowiedziami w analizowanych aplikacjach

1.2 Środowisko tworzenia aplikacji na Android

Środowiska, w jakich można tworzyć aplikacje mobilne to Android Studio[4] oraz Qt[5]. Oba środowiska oferują duże wsparcie ze strony społeczności programistycznej oraz szeroki zestaw narzędzi.

1.2.1 Android Studio

Środowisko to jest oparte na IntelliJ IDEA[6]. Dedykowane jest tworzeniu aplikacji na systemy Android. Do budowania aplikacji wykorzystywany jest Gradle[7]. Biblioteki mogą być dodawane ręcznie lub za pomocą menadżera Maven[8]. Wbudowane jest wiele narzędzi do testowania pisanych aplikacji, integrowania ich z repozytoriami Git. Środowisko to pozwala także na wprowadzanie zmian w uruchomionej aplikacji bez konieczności jej ponownego budowania. W tym środowisku można tworzyć aplikacje w języku Java ale także C++ za pomocą Android NDK[9]. Aplikacje można pisać w Javie lub C++. Interfejs aplikacji tworzony jest w XML. Aplikacje stworzone w tym środowisku nie będą przenośne na inne systemy operacyjne niż Android.

1.2.2 Qt Creator

Środowisko to jest multiplatformowe. Można na nim tworzyć aplikacje na systemy Symbian, Maemo, MeeGo oraz Android. Bazowo nie jest to środowisko tak rozbudowane jak Android Studio, pozwala jednak na integrację wielu dodatkowych narzędzi. Pozwala na pisanie aplikacji w takich językach jak Python, Ruby, Perl, Java i C++. Interfejs tworzony jest w oparciu o QML, języku opartym na JavaScript, składnią zbliżonym do XML. Aplikacje stworzone w tym środowisku są łatwo przenośne na inne systemy operacyjne. Posiada większość odpowiedników klas występujących w Android Studio, potrzebnych do pisania aplikacji na systemy operacyjne Android.

[4]https://developer.android.com/studio/index.html
[5]https://www.qt.io/
[6]https://www.jetbrains.com/idea/
[7]https://gradle.org/
[8]https://maven.apache.org/
[9]https://developer.android.com/ndk/index.html

1.3 Komunikacja internetowa

Komunikacja internetowa jest kluczowym zagadnieniem występującym w tej pracy. Z racji tego, że użytkownikami systemu są prowadzący kursów i studenci, a zaliczenie kursów przez studentów w dużej mierze zależy od poprawności działania komunikacji sieciowej, musi ona być zrealizowana w sposób bezpieczny i niezawodny.

1.3.1 Protokoły

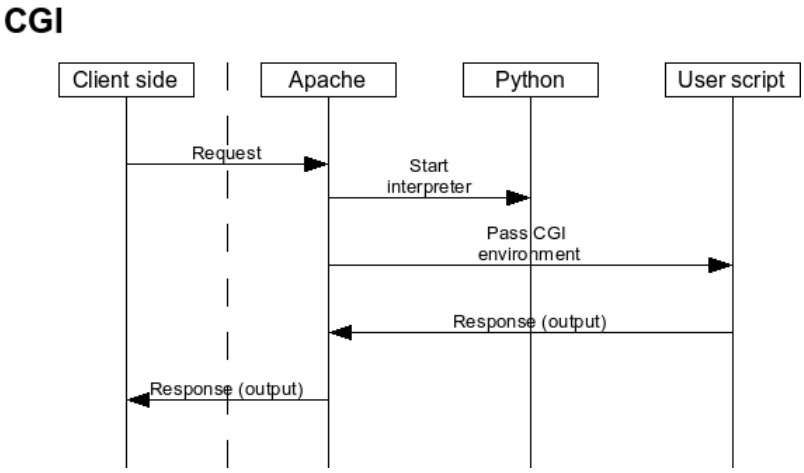
Przy tworzeniu systemu gdzie występuje komunikacja serwer-klient przez Internet jest wiele możliwych protokołów, które można użyć. Są to np. TCP, UDP, POP, SMTP, HTTP czy FTP. Wybór protokołu wpływa znacząco na sposób komunikacji pomiędzy klientem a serwerem. Każdy z protokołów różni się sposobem przesyłania danych oraz ich bezpieczeństwem. Protokoły bezpołączeniowe jak UDP nie pozwalają na bezpieczne przesyłanie danych, ponieważ nie ma informacji o ich poprawnym dotarciu do odbiorcy. Protokoły jak TCP, HTTP są protokołami połączeniowymi. W ich przypadku nadawca otrzymuje informację o poprawności przesyłu danych do odbiorcy. Każdy z protokołów domyślnie też korzysta z różnych portów do przesyłu danych. HTTP standardowo korzysta z portu 80, POP z kolei korzysta z portu 110, a FTP z portów 20 i 21.

1.3.2 Serwer HTTP

Pierwszym rodzajem systemu do obsługi zapytań aplikacji mobilnych jaki można wykorzystać to serwer HTTP połączony z FastCGI. Najpopularniejsze[10] aplikacje, które współpracują z FastCGI to między innymi: Apache[11], nginx[12], Microsoft-IIS[13], OpenLiteSpeed[14]. Każdy z serwerów obsługuje FastCGI (FCGI) pozwalające na napisanie programu przyjmującego i przetwarzającego przychodzące zapytania sieciowe. Proces napisany z pomocą FCGI łączy się z serwerem FCGI, do którego przekierowywane są zapytania przyjmowane przez serwer HTTP. Tym różni się od interfejsu CGI, że programy FCGI nie są uruchamiane przy każdym zapytaniu i kończone po ich obsłużeniu, tylko istnieją cały czas jako wątki, Oczekują na zapytania, obsługują je i nie kończą swojego działania po ich obsłużeniu. Ten interfejs jest kompatybilny z wieloma językami programowania, np. C++, Python, Java. W ten sposób można stworzyć mało rozbudowane ale szybki system przetwarzający zapytania sieciowe. Oprócz tego można też wykorzystać Serwer HTTP z CGI zamiast FCGI. W tym przypadku przy każdym zapytaniu uruchomi się interpreter skryptu mający wygenerować odpowiedź do zapytania[15]. W przypadku FCGI każdorazowe uruchamianie się interpretera nie zachodzi.

Wymienione serwery jednak różnią się między sobą wydajnością obsługiwania zapytań. Na podstawie Linux Web Server Performance Benchmark[16] można stwierdzić, że nginx oferuje największą szybkość obsługi zapytań.

[10] https://w3techs.com/technologies/overview/web_server/all
[11] https://httpd.apache.org/
[12] https://www.nginx.com/
[13] https://www.iis.net/
[14] http://open.litespeedtech.com/mediawiki/
[15] https://www.electricmonk.nl/docs/apache_fastcgi_python/apache_fastcgi_python.html
[16] https://www.rootusers.com/linux-web-server-performance-benchmark-2016-results/



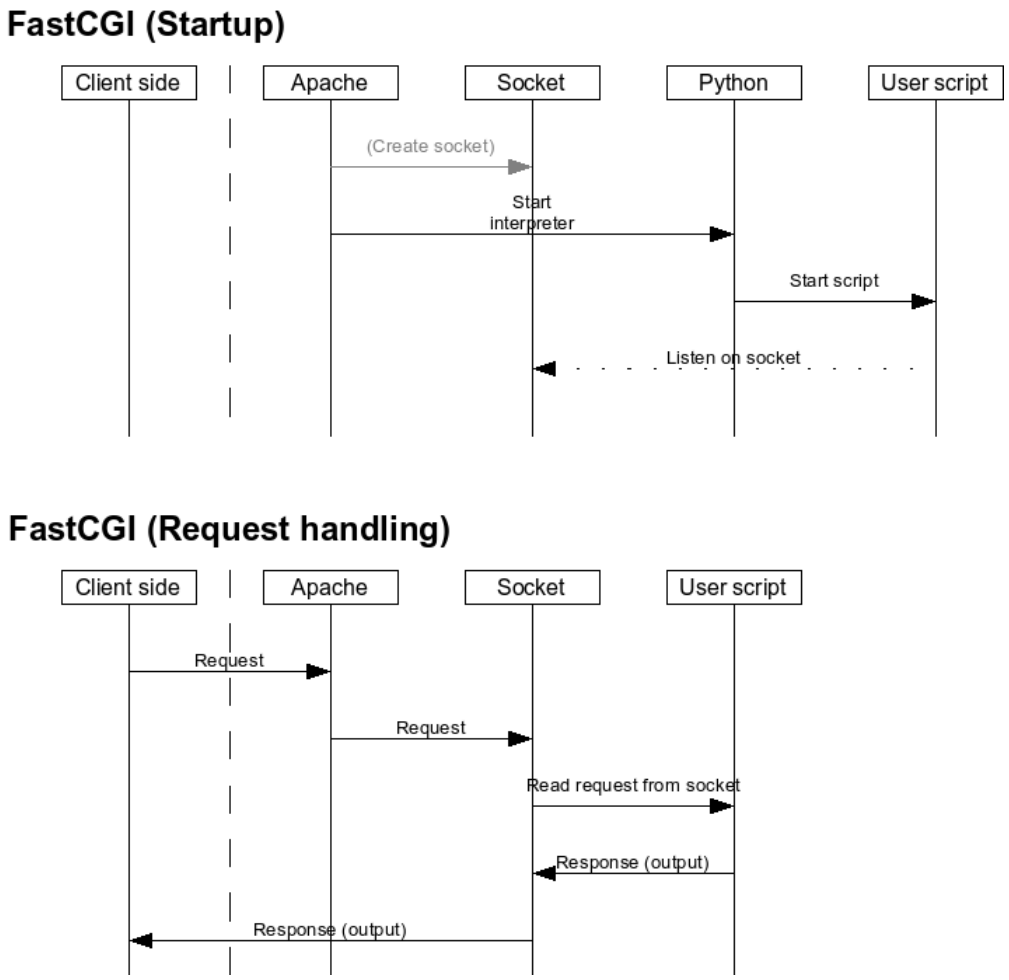
Rysunek 1.2: Schemat działania CGI ze skryptem do generowania odpowiedzi napisanym w Python. Źródło: Apache, FastCGI and Python

1.3.3 Serwer aplikacji

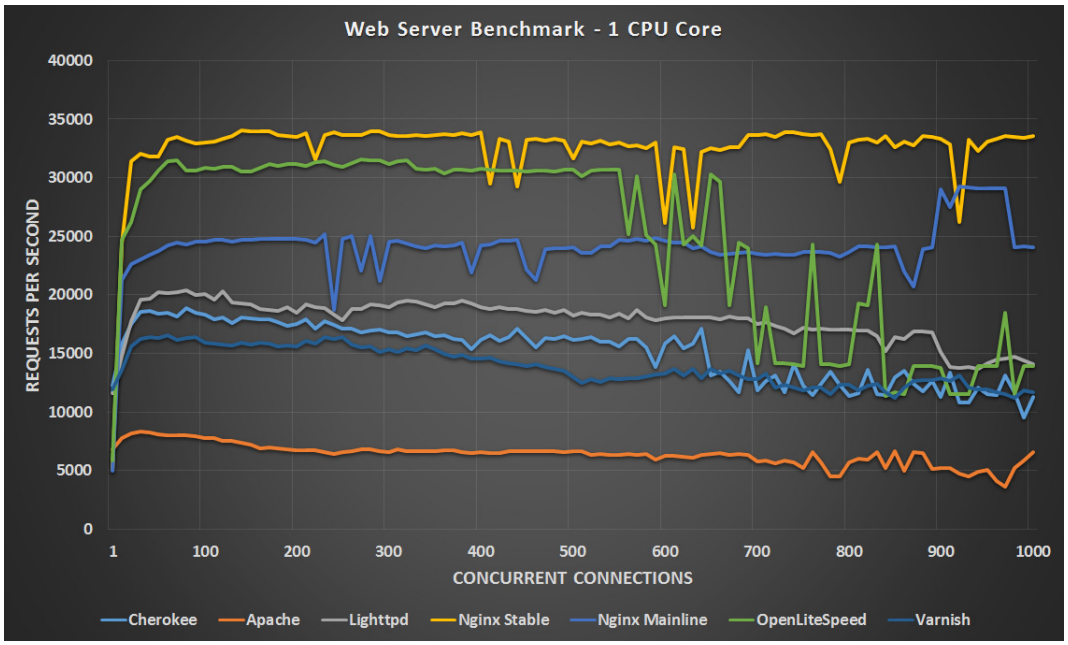
Drugim rodzajem systemu jaki można wykorzystać to serwer aplikacji działający na protokole HTTP. Najpopularniejsze[17] aplikacje to: Tomcat[18], WildFly[19] (dawny JBoss) oraz Jetty[20]. Za ich pomocą można napisać aplikację np. w języku Java lub C++, która później służy do obsługi zapytań sieciowych. W ten sposób można stworzyć zaawansowane serwery sieciowe odpowiadające potrzebom implementowanego systemu.

Porównanie wyżej wymienionych serwerów aplikacji można znaleźć w artykule Lightweight Java servers and developer view on the App Server[21].

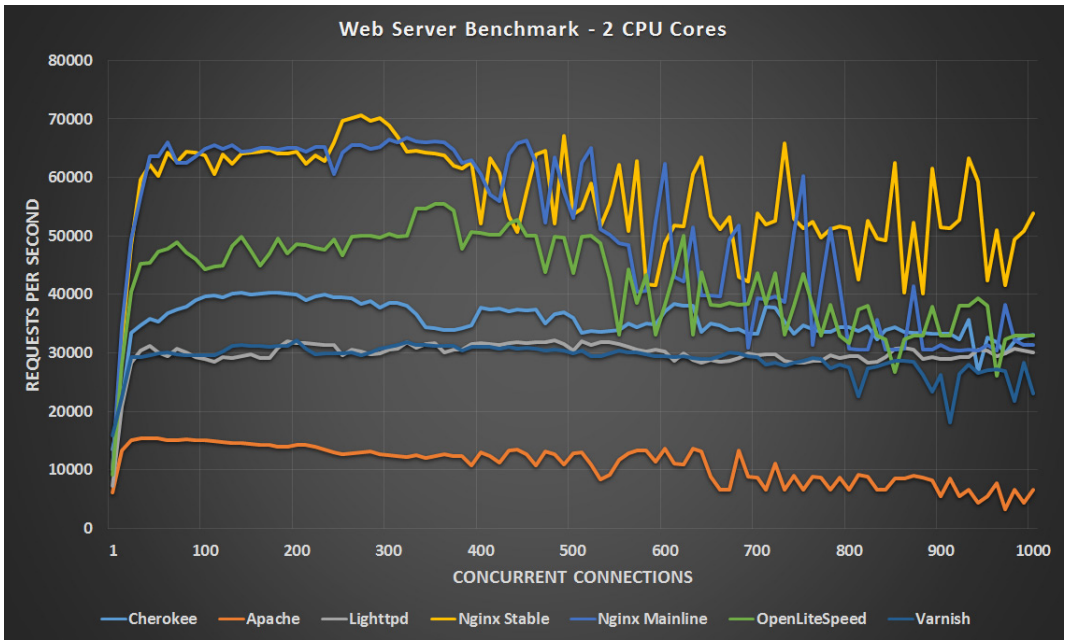
[17] https://plumbr.eu/uncategorized/most-popular-java-ee-servers-2016-edition
[18] https://tomcat.apache.org/
[19] http://wildfly.org/
[20] https://eclipse.org/jetty/
[21] https://advantage.ibm.com/2015/09/22/lightweight-java-servers-and-developer-view-on-the-app-server-update/



Rysunek 1.3: Schemat działania FCGI ze skryptem do generowania odpowiedzi napisanym w Python. Źródło: Apache, FastCGI and Python



Rysunek 1.4: Ilość obsługiwanych zapytań przy danej ilości otwartych połączeń. Źródło: Linux Web Server Performance Benchmark

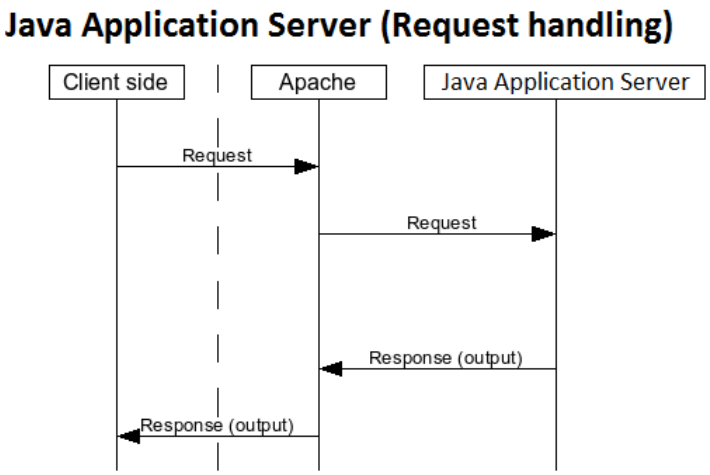


Rysunek 1.5: Ilość obsługiwanych zapytań przy danej ilości otwartych połączeń. Źródło: Linux Web Server Performance Benchmark

Excellent / Good / Limited / Very Limited / No support

	IBM WAS Liberty 8.5.5.7	IBM WAS full 8.5.5.6	Tomcat 8.0.26	TomEE+ 1.7.2[4]	Jetty 9.3.2	Glass Fish 4.1	Web Logic 12.1.3[3]	WildFly 9.0.1	JBoss EAP 6.4
Server stop+start[5]	4.9 sec	34.1 sec	5.5 sec	11.2 sec	3.1 sec	9.4 sec		10.2 sec	9.2 sec
App redeploy[5]	1.2 sec	6.1 sec	2.3 sec	2.5 sec	2.2 sec	2.5 sec		1.2 sec	1.2 sec
RAM[5]	59 MB	175 MB	125 MB	236 MB	102 MB	376 MB		269 MB	430 MB
Download size[1]	11 to 94 MB	3 GB	10 MB	48 MB	10 MB	103 MB		127 MB	158 MB
Size installed[1]	15-123 MB	2.6 GB	17 MB	52 MB	12 MB	214 MB		159 MB	174 MB
Size per instance	0.5 MB	40 MB	0.4 MB	0.4 MB	0.4 MB	96 MB		1.5 MB	1.2 MB
Dev. Install[6]	5 sec	30 min	2 sec	3 sec	1 sec	5 sec		5 sec	5 sec
# of config files	1+	100+	8+	12+	20+	14+		16+	16+
Dynamic config[2]	99%	80%	20%	20%	20%	20%	80%	60%	60%
IDE	Eclipse, IntelliJ IDEA, NetBeans are supported with some minor differences								
Configuration Editor	Eclipse UI	Browser UI	None	None	None	Browser UI	Browser UI	Browser UI	Browser UI
DevOps	Maven, Jenkins, Ant, Chef and other DevOps tools are supported with some minor differences								
Java EE	Java EE 7	Java EE 6+	JSP/ Servlet	Java EE 6 Web Prof.	JSP/ Servlet	Java EE 7	Java EE 6	Java EE 7	Java EE 6
Free Dev. License	IBM	IBM	Apache 2.0	Apache 2.0	EPL 1.0	CDDL 1.1	Oracle	LGPL 2.1	LGPL 2.1
Free Dev. Support	IBM[7]	IBM[7]	Self	Self	Self	Self	$	Self	Red Hat[8]

Rysunek 1.6: Porównanie popularnych serwerów aplikacji. Źródło: Lightweight Java servers and developer view on the App



Rysunek 1.7: Schemat działania serwera aplikacji.

Rozdział 2

Specyfikacja

2.1 Oryginalna specyfikacja

2.1.1 Aplikacja mobilna

Podstawowe wymagania jakie aplikacja ma spełniać są następujące:

- zgodność z systemami Android w wersji 2.2 - 7.0
- komunikacja z serwerem przez HTTP
- szyfrowanie odpowiedzi z testu i przechowywanie ich w pamięci wewnętrznej telefonu
- bezpieczny sposób przechowywania odpowiedzi na telefonie
- ochrona oszukiwaniem poprzez nieuprawnione wyjście i powrót do wykonywanego testu
- logiczny i przyjazny dla użytkownika interfejs

2.1.2 Aplikacja serwerowa

Wymagania aplikacji serwerowej są następujące:

- komunikacja przez HTTP
- duża wydajność przetwarzania zapytań sieciowych
- przetwarzanie i zapisywanie zapytań przychodzących z aplikacji mobilnych
- zachowanie możliwie wysokiej prostoty aplikacji serwerowej

2.1.3 Dodatkowe programy

- deszyfrator plików z odpowiedziami tworzonych przez aplikację mobilną

2.2 Dodatki i zmiany

2.2.1 Aplikacja mobilna

W trakcie implementacji i testowania systemu były wprowadzane zmiany w specyfikacji aplikacji mobilnej. Następujące zmiany to:

- komunikacja z serwerem poprzez HTTPS
- minimalna wersja obsługiwanego systemu Android zmieniona na 4.0
- lista z pytaniami zmieniona ze statycznej na dynamiczną

Dodatki to:

- odrzucanie połączeń przychodzących do użytkownika w trakcie testu
- długotrwałe przechowywanie parametrów konfiguracji takich jak imię, nazwisko, itd., w aplikacji
- wyświetlanie informacji dla użytkownika o procesach zachodzących w aplikacji
- nadanie aplikacji cyfrowego podpisu
- obfuskacja skompilowanego kodu źródłowego aplikacji
- testowanie osiągalności serwera HTTP
- weryfikacja aplikacji za pomocą SafetyNet
- automatyczna konfiguracja parametrów testu na podstawie danych otrzymanych z serwera

2.2.2 Aplikacja serwerowa

Dodatki w specyfikacji aplikacji serwerowej wyglądają następująco:

- walidacja aplikacji mobilnych poprzez SafetyNet
- przesyłanie konfiguracji testu do aplikacji mobilnej

2.2.3 Dodatkowe programy

- konfigurator tworzący pliki konfiguracyjne dla aplikacji mobilnych

Rozdział 3

Implementacja

Biorąc pod uwagę przeprowadzoną analizę problemu zdecydowano, że system do rozwiązywania testów musi być napisany od podstaw. Nie znaleziono przykładów otwartych systemów na tyle użytecznych, żeby można było użyć ich przy tworzeniu systemu. Systemy takie jak Quizowanie mają zamknięty kod i można się nimi sugerować jedynie w kwestii tworzenia interfejsu użytkownika.

3.1 Elementy aplikacji

3.1.1 Widoki aplikacji

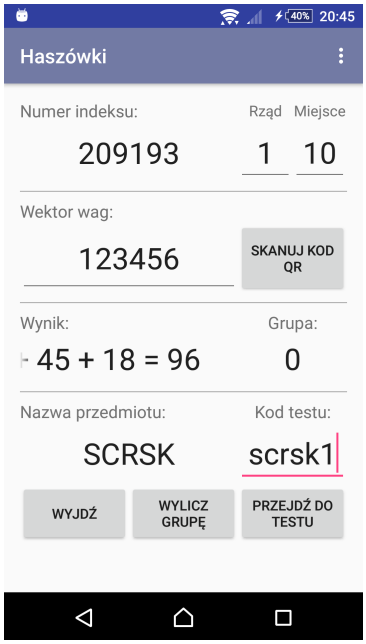
Wygląd aplikacji stanowi ważny element aplikacji. W początkowych fazach projektu rozmieszczenie elementów na ekranie telefonu komórkowego jak też ich estetyka wielokrotnie była zmieniana. Potrzebne było wypracowanie przejrzystego oraz przyjaznego dla studenta interfejsu aplikacji. W trakcie tworzenia aplikacji zauważono, że istotnym elementem jest zachowanie kompatybilności z różnymi wersjami systemu Android realizując jednolite i przewidywalne zachowanie się interfejsu. Na przykład nie można było zastosować kolorowania przycisków korzystając z metody setColorFilter obecnej we wszystkich wersjach systemu od Android 4.0 do Android 7.0. Efekt wywołania tej metody w systemach Android 4.0 do 4.4 był inny od efektu uzyskiwanego w systemach Android 5.0 i wyższych. Powodowało to brak kompatybilności pomiędzy tymi wersjami.

Widoki jakie zostały zaimplementowane w aplikacji to:

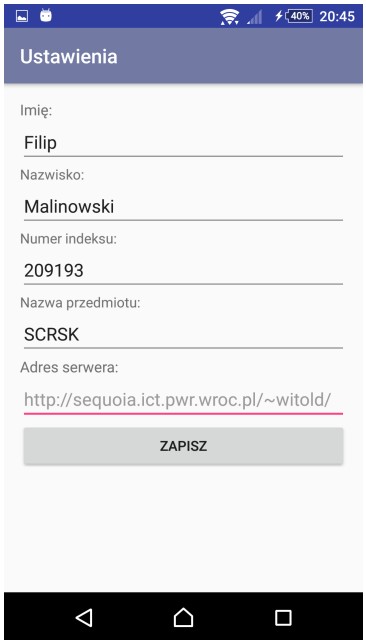
- Widok główny
  Za jego pomocą użytkownik może wprowadzić swoje miejsce i rząd w jakim się znajduje, wektor wag służący do wyliczenia grupy na teście oraz kod testu. Na tym widoku można uruchomić również skaner kodów QR do automatycznego pobrania wektora wag i kodu testu oraz wyliczenia grupy na teście. Trzy dolne przyciski służą do wyjścia z aplikacji, ręcznego wyliczenia numeru grupy oraz przejścia do testu. Z tego widoku można również otworzyć menu, z którego można dostać się do widoku ustawień i widoku informacji.

- Widok ustawień
  Tutaj użytkownik może wprowadzić i zapisać w aplikacji swoje dane takie jak: imię, nazwisko, numer indeksu oraz nazwa przedmiotu. Opcjonalnie może wprowadzić inny adres serwera, na który aplikacja będzie przesyłać odpowiedzi użytkownika.

- Widok konfigurowania testu
  Tutaj użytkownik jest informowany o przebiegu konfiguracji testu. Aplikacja



(a) Menu główne



(b) Ustawienia

Rysunek 3.1: Przykładowo skonfigurowane widoki menu głównego i ustawień

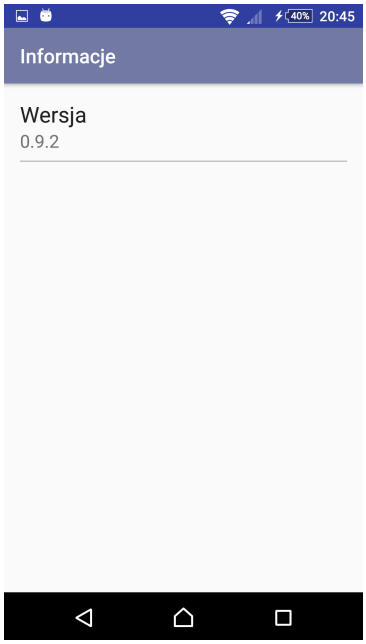
sprawdza osiągalność serwera, do którego wysyła odpowiedzi oraz pobiera plik konfiguracyjny zawierający takie dane jak: minimalna i maksymalna wersja aplikacji dopuszczona do testu oraz klucz do szyfrowania danych w plikach testu przechowywanych na telefonie użytkownika. Następnie porównuje dopuszczalną wersję aplikacji z aktualną wersją oraz zapisuje klucz szyfrowania.

- Widok testu
  W tym miejscu wyświetlany jest zestaw zakładek, po którym użytkownik może się poruszać i wchodzić w interakcje. UI użytkownika pozwala na przesuwanie ekranu w celu wyświetlenia innych pytań. Początkowo wyświetlana jest tylko jedna zakładka. Na zakładce pytania znajduje się: nr grupy użytkownika, nr pytania, przyciski: tak, nie, nie wiem do wysłania odpowiedzi, przyciski do dodawania pytań oraz do podsumowania testu. Na dole zakładki wyświetlany jest unikalny identyfikator sesji wygenerowany dla aktualnej sesji testowej otwartej na telefonie. Dodanie pytania wymusza przejście do następnego pytania.

- Widok podsumowania
  Tutaj wyświetlana jest ilość poprawnie przesłanych odpowiedzi "tak", "nie" oraz "nie wiem" do serwera, oraz ilość odpowiedzi zapisanych w pliku testowym. W razie różnicy w odpowiedziach przesłanych do serwera, a zapisanych w pliku aplikacja wyświetla odpowiednie ostrzeżenie sugerujące użytkownikowi powrót do testu i ponowne przesłanie odpowiedzi. Na ekranie podsumowania wyświetlany jest przycisk pozwalający powrót do testu oraz przycisk kończący test powodujący powrót do menu głównego aplikacji.

- Widok informacji
  W tym widoku użytkownik może się dowiedzieć z jakiej wersji aplikacji korzysta.

(a) Podsumowanie testu



(b) Informacje

Rysunek 3.2: Możliwe informacje wyświetlane na podsumowaniu i widoku informacji

3.1.2 Interaktywne elementy na widokach

Rozwijane listy zostały utworzone w widoku ustawień i widoku głównym. W widoku ustawień po kliknięciu na pole do edycji nazwy testu otwiera się okienko z przesuwalną listą, z której można wybrać nazwę kursu. W widoku głównym taka sama wizualnie lista wyświetla propozycje ID testu wygenerowane na podstawie nazwy kursu.

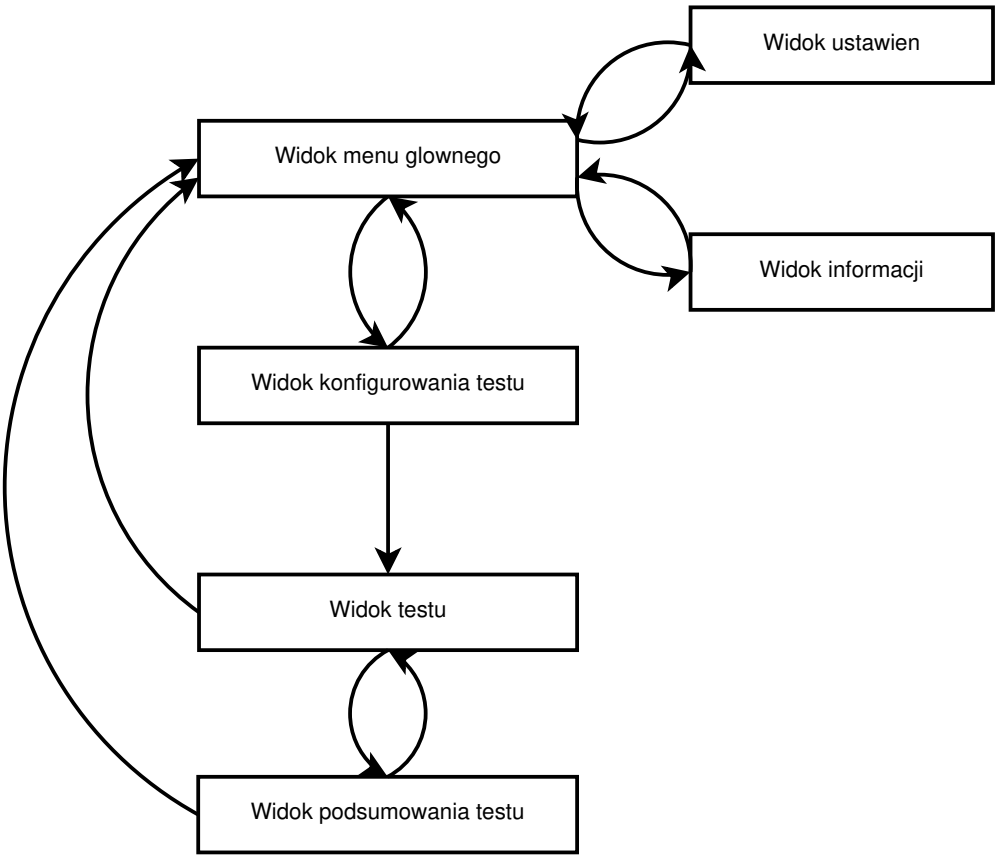
Zmiennokształtne przyciski służą do sygnalizowania asynchronicznych operacji zachodzących w trakcie konfiguracji i weryfikacji aplikacji. Za podstawę służy Android Circular Progress Button[1], który został umieszczony na widoku ustawień. Przycisk ten w aplikacji zmienia swój kolor oraz wygląd sygnalizując:

- bezczynność - niebieski pusty przycisk
- przetwarzanie operacji - wirujące kółko
- powodzenie przetwarzania operacji - zielony przycisk z checkmark
- niepowodzenie przetwarzania operacji - czerwony przycisk

Sygnalizowanie bezczynności zostało początkowo użyte dla jednej nieaktywnej funkcji, którą była walidacja aplikacji. Zielony kolor sygnalizuje powodzenie testowego połączenia z serwerem, poprawne pobranie pliku konfiguracyjnego oraz walidację klucza szyfrowania i dopuszczalny zakres wersji aplikacji. Czerwony kolor sygnalizuje niepowodzenie wyżej wymienionych operacji. Wirujące kółko trwa tak długo jak te asynchronicznie wykonywane operacje nie zakończą swojego działania.

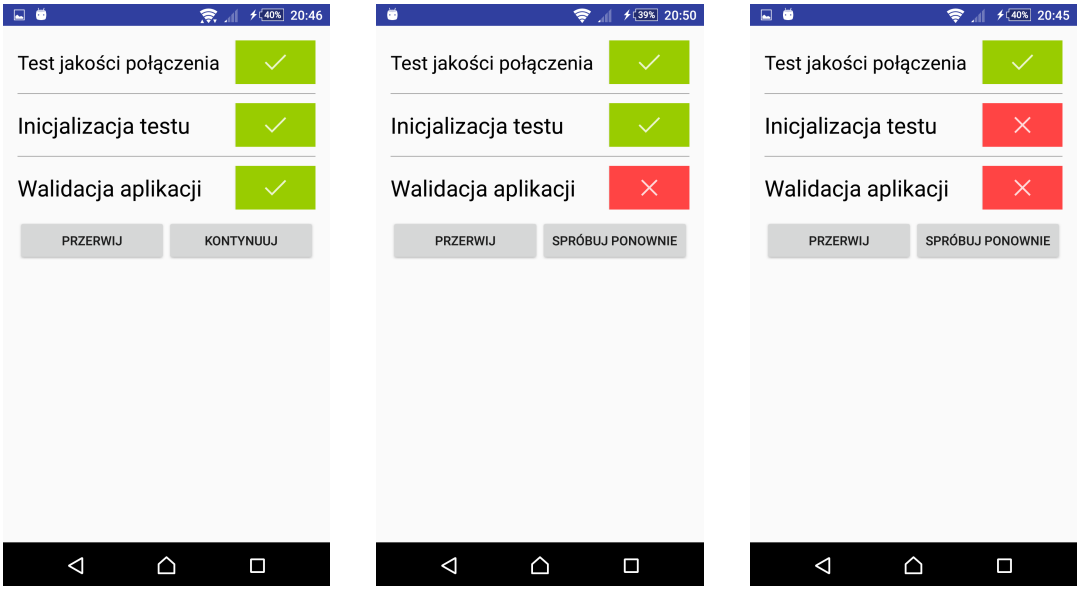
Przyciski zmieniające kolor umieszczone są na kartach z odpowiedziami w aplikacji. Sygnalizują swoimi kolorami stan przetwarzania odpowiedzi użytkownika. Biały przycisk informuje o braku wybranej do tej pory odpowiedzi. Szary przycisk

[1]https://github.com/flavioarfaria/circular-progress-button



Rysunek 3.3: Zależności pomiędzy widokami informujące o tym jak użytkownik może się po nich poruszać

sygnalizuje poprawnie zapisaną odpowiedź do pliku testowego. Niebieski przycisk sygnalizuje poprawnie wysłaną odpowiedź na serwer. Jeżeli operacja zapisywania do pliku się nie powiedzie to przycisk pozostaje biały. Jeżeli operacja wysyłania do serwera się nie powiedzie to przycisk pozostaje szary. Użytkownik wtedy wie, że jego aktualna odpowiedź przechowywana jest jedynie lokalnie.



(a) Walidacja przeszła poprawnie (b) Niepoprawna wersja lub klucz (c) Brak pliku konfiguracyjnego na serwerze

Rysunek 3.4: Przykładowe wyniki etapu konfiguracji

3.1.3 Powiadomienia

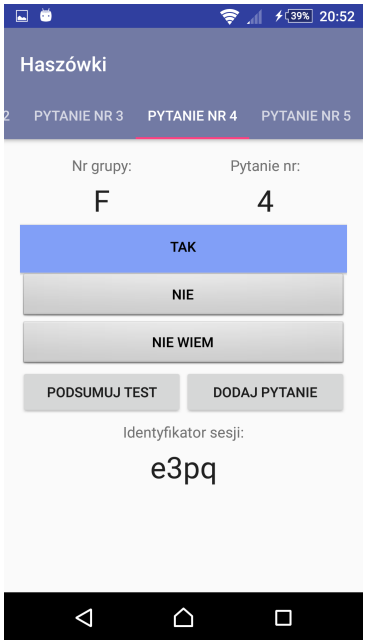
Istotnym elementem aplikacji okazały się być powiadomienia pozwalające na zrozumienie użytkownikowi zdarzeń występujących w aplikacji. Dzięki wypracowaniu zestawu chmurek oraz monitów użytkownik jest na przykład informowany o:

- wprowadzeniu nr indeksu o niepoprawnej długości,
- nie wprowadzeniu swojego imienia, nazwiska, kursu, indeksu,
- wprowadzeniu niepoprawnego wektora wag (równego 0),
- nie wprowadzeniu swojego miejsca i rzędu,
- braku dostępu do sieci,
- braku łączności z serwerem,
- niedotarciu do serwera danych odpowiedzi, itd.

3.1.4 Komunikacja internetowa

Z tego powodu, że w niezabezpieczonej sieci WiFi Politechniki Wrocławskiej odblokowane są jedynie porty służące do obsługi poczty i protokołu HTTP to do komunikacji internetowej wykorzystano protokół HTTP. Odpowiedzi są wysyłane na zasadzie pobierania nagłówka pliku konfiguracyjnego XML z odpowiedziami umieszczonymi w query string. Kod otrzymany po wykonaniu zapytania jest następnie interpretowany, a jego wartość decyduje o powiadomieniu użytkownika o poprawnym lub niepoprawnym wysłaniu odpowiedzi do serwera.

(a) Wybrana odpowiedź "tak"



(b) Brak wybranej odpowiedzi

Rysunek 3.5: Widok testu

3.1.5 Szyfrowanie plików

Początkowo pliki były szyfrowane algorytmem DES z kluczem przechowywanym w aplikacji. Było to jednak rozwiązanie niebezpieczne, ponieważ klucz szyfrowania można było uzyskać po zdekompilowaniu aplikacji. Pierwszym ulepszeniem zaimplementowanym w aplikacji było zastosowanie metody "security by obscurity", gdzie klucz szyfrowania zmieniał się w trakcie działania programu. W ten sposób sprawa jego uzyskania została utrudniona. Następnie w aplikacji zaimplementowano pobieranie dokumentu XML z ustawieniami testu. W tym dokumencie znajdował się klucz szyfrowania wykorzystywany później w algorytmie DES. W dalszym etapie wykorzystano szyfrowanie algorytmem RSA, którym zastąpiono algorytm DES, a klucz w dokumencie XML został zamieniony na 4096 bitowy klucz publiczny algorytmu RSA.

3.1.6 Zapisywanie plików na telefonie

Po szyfrowaniu odpowiedzi aplikacja musi je zapisać w pamięci telefonu w pliku tekstowym. Android nie umożliwia w prosty sposób pisania strumieniem bitowym do pliku. Zapisywanie i czytanie z plików opiera się na zapisywaniu i czytaniu stringów. Pojawiła się więc potrzeba zastosowania kodowania transportowego po to, żeby w trakcie operacji zapisu i odczytu dane nie mogły ulec uszkodzeniu. Wykorzystać do tego można base64, base16 (Hex) lub też kodowanie będące jednobajtowym kodowaniem znaków jak np. ISO-8859-1. Dzięki zastosowaniu jednego z powyższych metod kodowania bajtów można w bezpieczny sposób zapisywać i odczytywać ciągi bajtów będące zakodowanymi odpowiedziami.

3.1.7 Skaner QR

W aplikacji wykorzystano również skaner kodów QR. Wykorzystano do tego bibliotekę zxing[2] (Zebra Crossing). Pozwala ona nie tylko na skanowanie kodów QR

[2]https://github.com/zxing/zxing

ale również kodów kreskowych, kodów Aztec i innych. Skaner skanuje kod QR wyświetlany na ekranie przez rzutnik, automatyczne zapisując w aplikacji kodu testu i wektor wag. Po operacji skanowania wymusza wyliczenie nowego numeru grupy. Ułatwia to pracę z aplikacją potencjalnemu użytkownikowi oraz pozwala na zmniejszenie prawdopodobieństwa popełnienia błędu przez użytkownika przy wyliczaniu grupy na teście.

3.1.8 Baza danych

Do długotrwałego przechowywania danych w aplikacji wykorzystano bazę danych MySQL. W tym celu stworzono klasę imitującą bazę danych MySQL oraz klasę imitującą interfejs dostępowy do tej klasy. Interfejs został następnie wykorzystany do zapisywania informacji zawartych w widoku ustawień. Baza danych dodatkowo służy jako narzędzie do wymiany danych między widokami aplikacji. Baza danych służy do przechowywania danych użytkownika takich jak: imię, nazwisko, numer indeksu oraz inne parametry właściwe dla aktualnej sesji testu. Przy tworzeniu bazy danych oparto się na przykładzie opisanym na portalu Vogella[3].

3.1.9 Odrzucanie połączeń w trakcie testu

W trakcie testowania aplikacji na grupie studentów pojawiła się potrzeba automatycznego odrzucania przychodzących połączeń telefonicznych. W tym celu stworzono klasę na bazie wbudowanej BroadcastReceiver, która to odrzuca wszystkie przychodzące połączenia jeśli użytkownik jest w trakcie pisania testu. Do odrzucania połączeń wykorzystano kod z portalu Stack Overflow[4].

3.2 Programy dodatkowe

3.2.1 Generator konfiguracji

Do tworzenia plików konfiguracyjnych napisany został konfigurator w języku Java. Pozwala na uruchomienie się tylko w linii poleceń. W parametrach wywołania tego programu podaje się kod testu, minimalną wersję aplikacji oraz maksymalną wersję aplikacji dopuszczoną do testu. Program generuje dwa pliki. Pierwszy jest dokumentem XML zawierającym klucz publiczny RSA plus dodatkowe parametry testu przeznaczone dla aplikacji Android. Drugi plik zawiera klucz prywatny RSA służący do rozszyfrowywania odpowiedzi zaszyfrowanych wcześniej kluczem publicznym.

3.2.2 Deszyfrator

Deszyfrator również został napisany w języku Java i umożliwia uruchomienie się jedynie w linii poleceń. Początkowa wersja deszyfratora korzystała z algorytmu DES, najpierw ze stałym kluczem zapisanym w aplikacji, potem z kluczem odczytywanym z pliku. Deszyfrowanie odbywało się za pomocą klucza symetrycznego. Ostatecznie stosuje algorytm RSA, a klucz prywatny do rozszyfrowywania pobiera z pliku. Korzysta przy tym z szyfrowania asymetrycznego.

[3]http://www.vogella.com/tutorials/AndroidSQLite/article.html
[4]https://stackoverflow.com/questions/7347871/how-to-reject-a-call-programatically-in-android

3.2.3 Parser logów

Do przetwarzania logów serwera Apache wykorzystano skrypt napisany w języku AWK. Skanuje on logi serwera Apache w poszukiwaniu zarejestrowanych zapytań przysłanych z aplikacji mobilnych i umożliwia zapisanie ich w formacie użytecznym dla zarządcy systemu. Skrypt został napisany przez Promotora dr Witolda Paluszyńskiego.

Rozdział 4

Praktyczne wykorzystanie systemu

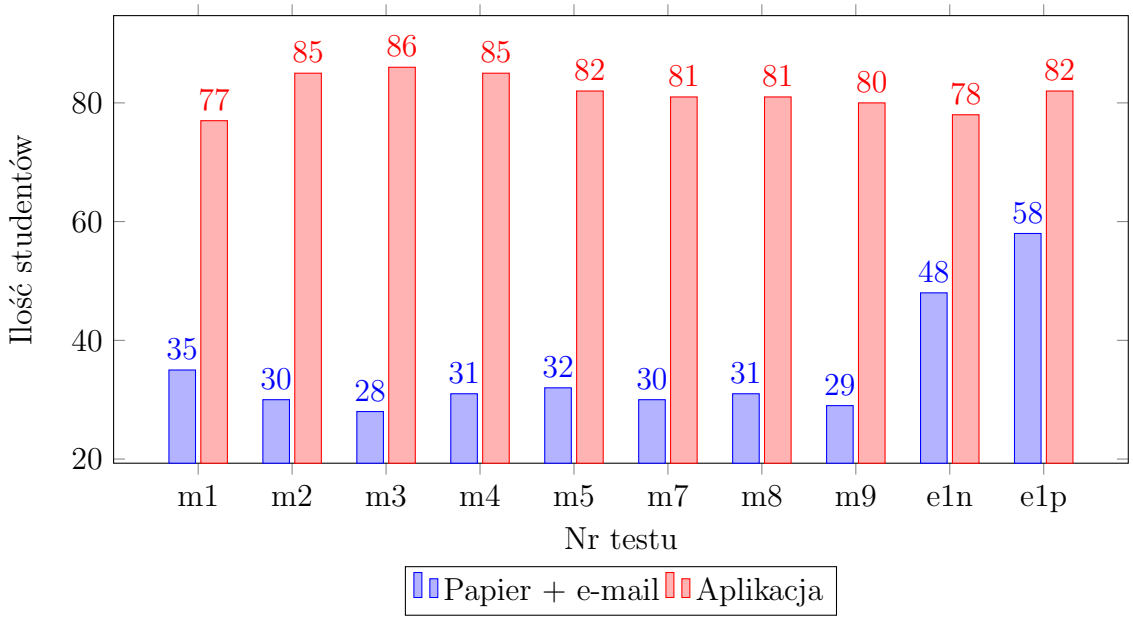
System został pomyślnie wykorzystany na Wydziale Mechanicznym (W-10), kierunku Automatyce i Robotyce na kursie Systemy Czasu Rzeczywistego i Sieci Komputerowe (SCRSK). Średnio z aplikacji korzystało około 90 osób na każdym z siedmiu wykonanych testów. Ponadto aplikacja została wykorzystana na testach na Wydziale Elektroniki (W-4), kierunku Automatyce i Robotyce na kursie SCR Systemy Operacyjne.

4.1 Wdrożenie aplikacji

Przygotowywanie aplikacji zostało rozpoczęte w lipcu 2016 roku. Tak duże wyprzedzenie wynikało z możliwości przetestowania aplikacji na wykładach jako platformy testowej, sprawdzenia jej przydatności i w ten sposób zweryfikowania idei takiego systemu. Pierwszy test odbył się 4 listopada 2016 roku.

4.2 Statystyki użycia aplikacji

Studenci w trakcie testów mieli do wyboru dwie formy pisania testów. Pierwszą było pisanie testu za pomocą telefonu z systemem Android i zainstalowaną aplikacją. Drugą było pisanie testów na papierze i dostarczanie ich do wykładowcy drogą e-mailową.
Wykorzystanie obu form dostarczania odpowiedzi z testu wyglądają następująco:

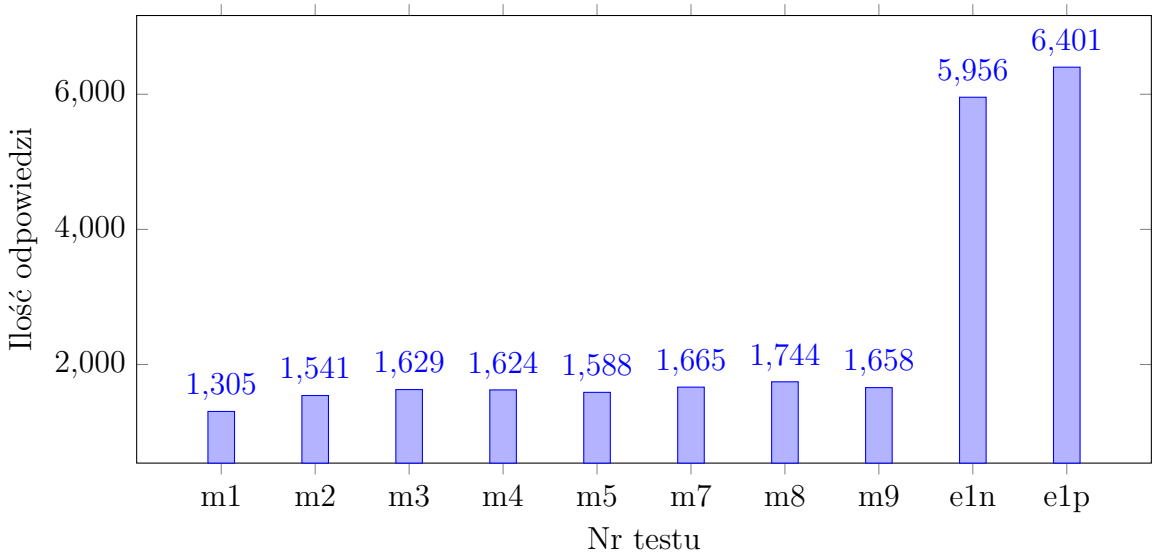


O możliwości pisania testów studenci zostali powiadomieni 2 dni przed wykonaniem 1 testu na Wydziale Mechanicznym. Stąd można zaobserwować wzrost ilości użytkowników aplikacji na 2 teście, między innymi kosztem papierowej formy pisania testu. W dalszych testach można zaobserwować powolny spadek użytkowników aplikacji jak i formy papierowej. Wynika to z rezygnacji studentów z pisania testów cząstkowych w celu przystąpienia do testu końcowego będącego alternatywnym sposobem zaliczenia kursu.

gdzie:

- od m1 do m9 - testy wykonane na Wydziale Mechanicznym
- e1n - test wykonany na Wydziale Elektroniki w tygodniu nieparzystym
- e1p - test wykonany na Wydziale Elektroniki w tygodniu parzystym

Suma wszystkich otrzymanych zapytań na serwerze dla każdego z testów wygląda następująco:



Średnia szczytowa ilość odpowiedzi w trakcie pisania testów oscylowała między 10 a 16 odpowiedziami na sekundę. Najwyższa zarejestrowana wartość szczytowa to było 29 odpowiedzi na sekundę.
Na wykresie można zauważyć jak zmienia się obłożenie serwera w trakcie rozwijania aplikacji Android. Wraz z dokładaniem kolejnych pakietów informacji przesyłanych do serwera w zauważalny sposób zwiększa się sumaryczna ich ilość otrzymana na serwerze. Porównując ostatni test do pierwszego testu wykonanego na Wydziale Mechanicznym zauważono wzrost o 27% sumarycznej ilości odpowiedzi.
Dodatkowo po wprowadzeniu informowania serwera o braku wybranej odpowiedzi przez studenta szczytowa ilość odpowiedzi wzrosła średnio o 3 odpowiedzi na sekundę.

Statystyki błędów popełnianych przez studentów na Wydziale Mechanicznym oraz Wydziale Elektroniki na przestrzeni lat wyglądają następująco:

Wydział	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017
Elektroniki	7	24	22	14	7
Mechaniczny	5	6	6	10	1

Każdemu rocznikowi odpowiada suma wszystkich jednostkowych błędów popełnionych w trakcie semestru. Na każdy semestr Elektroniki składa się od 8 do 9 testów

16 pytaniowych. W przypadku Mechanicznego każdy semestr odpowiada dwóm testom 64 pytaniowym.

Można zauważyć, że dzięki wykorzystaniu aplikacji udało się zauważalnie zmniejszyć ilość błędów popełnianych przez studentów. W przypadku AiR na W-10 ilość błędów spadła do poziomu z okresu 2012-2013. Zaś w przypadku AiR na W-4 udało się uzyskać mniejszą ilość błędów niż w jakimkolwiek poprzednim okresie.

Rozdział 5

Podsumowanie

W tej pracy stworzono system pozwalający na wykonywanie testów na aplikacjach Android. Aplikacje komunikują się z serwerem przez Internet przesyłając wszystkie informacje o wykonanych testach.

Aplikacja jest stabilna i niezawodna. Znacząco ułatwia pisanie testu wyliczając dla użytkownika wszystkie potrzebne informacje.

Pisanie systemu tej wielkości w znacznym stopniu zależy od odpowiedniego wyspecyfikowania elementów takiego systemu. Ich początkowy opis decyduje o strukturze oprogramowania systemu w etapie weryfikacji i wdrażania.