Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Projekt inżynierski

Tomasz Pawlicki

Jacek Rajda

Kierunek studiów: **Elektronika i Telekomunikacja**

Optymalizacja kosztów usług telefonii komórkowej za pomocą aplikacji mobilnej

Optimizing the cost of mobile services by means of mobile application

Promotor pracy: dr inż. Jarosław Bułat

Kraków, styczeń 2012

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie (w zakresie wyszczególnionym we wstępie) i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

……

……

Spis treści

Wprowadzenie

Człowiek dąży do minimalizacji kosztów, szukając ekonomicznych rozwiązań. Nasz świat właśnie tak funkcjonuje, by korzystać z możliwie prostych i tanich rozwiązań. Zatem możemy tutaj postawić pytanie. Dlaczego mamy płacić więcej za tą samą rzecz lub usługę, skoro możemy zapłacić mniej?

Zaprojektowany system służy do minimalizacji kosztów za usługi płacone operatorom telefonii komórkowej. Pod pojęciem usług znajdują się wszystkie koszty, jakie ponosimy podczas korzystania z telefonu. W skład nich wchodzą: opłaty za rozmowy telefoniczne, SMS-y, transfer internetowy czy inne dodatkowe usługi oferowane przez operatorów tj. „darmowe numery", „nielimitowany transfer internetowy" i inne.

Bardzo ważne, wstępne prace polegały na zebraniu i analizowaniu najważniejszych informacji na temat istniejących taryf, ofert, usług i cenników operatorów oferujących swoje usługi w Polsce. Podstawowym źródłem informacji stały się oferty głównych operatorów strukturalnych telefonii komórkowej, czyli Polkomtel S.A., P4 Sp. z o.o., PTK Centertel i Polska Telefonia Cyfrowa S.A. (załączniki A-D).

Zaprojektowany system zbudowany jest z trzech głównych części. Bazy danych, zawierającej informacje od operatorów (podrozdział 2.1), interfejsu administratora, wykorzystującego techniki HTML, CSS i PHP (podrozdział 2.2.) oraz aplikacji na platformę Android, gdzie wykonywane są obliczenia optymalizacyjne. Aplikacja posiada interfejs użytkownika opisany w podrozdziale 2.3.

W celu zmniejszenia rachunków za wykonane usługi telefoniczne, zaprojektowany system potrzebuje kilku informacji. Przede wszystkim: zbiór wszystkich taryf od różnych operatorów oraz przesłanie ich do aplikacji mobilnej. Dane od użytkownika, czyli taryfa jaką używa, jak długo jest w sieci, albo kiedy zaczyna się jego okres rozliczeniowy. Następnie analizowane są koszty poszczególnych usług z jakich użytkownik skorzystał, zaś w międzyczasie sprawdzane jest jak długo dany użytkownik rozmawiał z konkretnym operatorem. Następnie po wykonaniu działań optymalizacyjnych określony zostaje odpowiedni tańszy abonament wraz ze zbiorem

konkretnych usług. Wynik ten będzie dostępny w aplikacji na telefon, gdzie będzie można zapoznać się z proponowaną ofertą.

Rozdział 2 będzie stanowić podstawę teoretyczną uzasadniającą wybór narzędzi programistycznych. Część praktyczna przedstawiona w rozdziale 3, prezentuje w postaci schematu wspólne zależności między elementami składowymi projektu, takimi jak strona internetowa, baza danych czy aplikacja na platformę Android. W podrozdziałach 3.1-3 przedstawiono poszczególne elementy projektowanego systemu, zaś w podrozdziale 3.4 wskazano plany jego dalszego rozwoju.

1. Założenia projektu

Projektowanie aplikacji powinno być oparte na właściwym planie. Plan taki musi zawierać podstawowe założenia, jak na przykład: zbiór technik projektowych oraz wzajemne ich współistnienie w osiągnięciu z góry określonego celu. W tym rozdziale przedstawiono podstawowe informacje na temat aplikacji oraz strony internetowej będących tematem tej pracy a następnie wskazano różnice względem produktów konkurencji.

1.1. Cel

Projekt polegał na napisaniu na platformę Android programu do analizy i optymalizacji kosztów usług w telefonii ponoszonych przez każdego użytkownika telefonu komórkowego. Program ten miał za zadanie wykonywać cały algorytm optymalizacji na telefonie, żeby dane poufne użytkownika nie były zapisywane i dostępne na zewnątrz urządzenia. Oprócz aplikacji na urządzenie mobilne powstała również baza danych zawierająca zbiór ofert operatorów komórkowych oraz panel administracyjny w postaci strony internetowej, służący zarządzaniu informacjami zebranymi w tej bazie.

1.2. Pozyskiwanie informacji o wykonanych połączeniach

Wymagania dotyczące wyliczanej taryfy oparte są na historii połączeń znajdującej się w telefonie. Aplikacja nie wymusza więc, żeby użytkownik sam wprowadzał liczbę wysłanych SMS czy długość połączeń do poszczególnych operatorów. Wymagane jest jedynie od użytkownika wskazanie aktualnej taryfy i usług, z których korzysta. Dzięki temu aplikacja potrafi wyliczyć koszt za połączenia w ramach bieżącej taryfy u danego operatora.

1.3. Polityka poufności

Wspomniane źródło informacji (historia połączeń) oraz fakt wykonywania algorytmu optymalizującego na telefonie są ważne z punktu widzenia poufnych danych użytkownika. W rezultacie aplikacja do swojego poprawnego działania nie potrzebuje przesyłać ich na żadne serwery, na których miałoby nastąpić wykonanie procesu

optymalizacji. W skrócie, w algorytmie następuje odczytanie długości połączeń wychodzących, przeliczenie wybranych opcji z bazy danych a na końcu wyświetlenie najlepiej dobranej taryfy do telefonicznych predyspozycji danego użytkownika aplikacji.

1.4. Produkt konkurencyjny

Telefonia komórkowa z roku na rok ustanawia coraz to nowsze rekordy popularności. Już w roku 2007 według Głównego Urzędu Statystycznego [1] w Polsce znajdowało się tyle samo aktywnych kart SIM, co wszystkich mieszkańców co nie oznacza, że każdy obywatel ma telefon komórkowy. Liczbę tę zawyżają osoby o większej liczbie aktywnych kart SIM oraz operatorzy komórkowi, którzy w terminie nie dezaktywują wymaganych kart. Jednak liczba ta odzwierciedla we właściwy sposób skalę popularyzacji telefonii komórkowej.

Skoro telefonia dotyczy prawie każdego z nas i każdy kto z niej korzysta, musi za nią płacić, wymyślono produkt, który sprawi że zapłacimy mniej za rachunki telefoniczne. Jest nim serwis *Killbill*. Jego działanie polega na podobnej zasadzie, co w zaprojektowanej aplikacji, czyli w wyniku działania algorytmu zostaną wyświetlone najlepiej dopasowane taryfy operatorów do zapotrzebowań użytkownika produktu.

W tabeli 1.1 przedstawiono zalety i wady zaprojektowanej aplikacji względem aplikacji konkurencyjnej. Głównym źródłem różnic między tymi dwoma produktami są kolejno: inna forma dostępu do danych bilingowych, całkiem odmienna struktura budowy i różny cel powstania obydwóch produktów.

Tab. 1.1. Porównanie zaprojektowanej aplikacji z produktem konkurencji

Zaprojektowana aplikacja	**Killbill**
aplikacja na telefon	portal internetowy
dane użytkownika użyte w celu optymalizacji, pobierane są z historii telefonu	wymagane podanie przez użytkownika loginu i hasła, w celu zalogowania się na stronę operatora
dane poufne nie wychodzą poza telefon	operowanie danymi użytkowników na serwerach
aplikacja bezpłatna	za jedno skorzystanie pobrana opłata rzędu kilkunastu/kilkudziesięciu złotych
możliwość przeglądania pełnych ofert operatorów w czytelnej postaci	———————
produkt skierowany do grupy użytkowników telefonów z system operacyjnym Android	produkt, z którego może skorzystać prawie każdy, o ile zgadza się na powyższe warunki

Zaprojektowany produkt, oferowany będzie w całości bezpłatnie oraz bez konieczności rejestracji. Żeby wzbudzić zaufanie wśród potencjalnego użytkownika, cały algorytm wykonywany jest na poziomie telefonu. Tym samym w wyniku działania aplikacji, nikt nie ma dostępu do danych poufnych użytkownika takich jak: loginy, hasła, jak i również informacji na temat wykonanych przez użytkownika połączeń.

Dodatkowo oferujemy bezpłatną możliwość przeglądania na zaprojektowanej stronie internetowej aktualnych ofert operatorów komórkowych. Podobne rozwiązanie dotychczas nie powstało. Strona internetowa umożliwi użytkownikowi szybkie i łatwe poznanie obecnie obowiązujących taryf, usług i ofert operatorów. Zostanie on odciążony od czytania niezrozumiałych regulaminów, różniących się czasem od ofert opisanych na stronach internetowych operatorów. Uważamy, że aplikacja na platformę Android wraz ze stroną internetową zmniejszą problem niezorientowania użytkownika telefonu komórkowego wśród taryf i usług operatorów komórkowych.

W produkcie konkurencji źródło wiedzy na temat wydatków danego użytkownika brane jest ze strony operatora komórkowego, poprzez wcześniejsze podanie loginu i hasła na konto na stronie operatora. Należy podkreślić, że użytkownik nie po to na co dzień chroni się przed nieuczciwymi ludźmi, stosując na każdym kroku specjalne zestawy loginów i haseł, żeby je odtajniać w innych miejscach. Drugim momentem, kiedy od użytkownika wymagane jest podanie loginu i hasła jest moment rejestracji na stronie *www.killbill.pl*. Rejestracja nie zwalnia jednak od opłaty za skorzystanie z usług serwisu. Jednorazowy koszt skorzystania z aplikacji waha się w granicy kilkunastu do kilkudziesięciu złotych.

Aplikacja *Killbill* pomimo dostępności z poziomu przeglądarki internetowej, nie jest skierowana do każdego użytkownika posiadającego telefon. Ograniczeniem jest brak w aplikacji algorytmu dla osób będących posiadaczem telefonu z taryfą *mix*. Oznacza to, że jedynie użytkownicy telefonów na kartę i abonament mogą korzystać z produktu konkurencji.

Podsumowując, produkt konkurencji wymaga od użytkownika w swej obecnej wersji zaufania, że dane poufne, które służą mu do logowania na konto na stronie operatora oraz informacje na temat wykonanych połączeń, nie zostaną wykorzystane w nieznany sposób.

2. Wykorzystane narzędzia i techniki programowania

W tym rozdziale przedstawiono języki/techniki programistyczne, na których został oparty ten projekt inżynierski. Zebrane informacje przedstawiają podstawową strukturę wybranych języków oraz ideę wykorzystania ich w projekcie.

2.1. PostgreSQL

SQL (z ang. *Structured Query Language*) jest językiem programowania służącym do zarządzania danymi w relacyjnej bazie danych. SQL jest narzędziem do bezpośredniego operowania danymi w bazie danych, między innymi możliwe jest tworzenie tabel i zarządzanie danymi w nich zawartych, np.: dodawanie/modyfikacja wpisów, oraz wyciąganie potrzebnych zależności.

PostgreSQL, nazywany również Postgres-em, jest zaawansowanym systemem zarządzania relacyjnymi bazami danych typu Open Source (oprogramowanie otwarte zgodnie z warunkami licencji BSD). System ten w obecnej wersji implementuje standard SQL w wersji szóstej (SQL:2008). Poniżej przedstawiono różnice pomiędzy PostgreSQL a konkurencyjnym MySQL, powody realizacji bazy danych w oparciu o system PostgreSQL oraz podstawową architekturę systemu.

Pomimo ogromnych możliwości oferowanych przez PostgreSQL, takich jak znaczne zwiększenie zabezpieczeń, niezawodności i integralności danych; największą część uwagi wśród systemów relacyjnych baz danych Open Source otrzymuje nieustannie MySQL. Wynika to przede wszystkim z łatwości obsługi, z większej szybkości, jak i ukierunkowania kodów źródłowych wielu aplikacji Web na ten właśnie system. Łatwość obsługi systemu MySQL wiąże się jednak ze zmniejszoną funkcjonalnością oraz niższym poziomem bezpieczeństwa niż w PostgreSQL. Można jednak zauważyć, że pomiędzy nowszymi implementacjami tych dwóch systemów istnieje coraz więcej podobieństw. Wiąże się to ze zwiększaniem szybkości Postgresa, jak i rozszerzaniem funkcjonalności w MySQL. W projektowanym projekcie inżynierskim wybór systemu bazodanowego ograniczał się jedynie do wyboru spośród konkretnej wersji obydwu wymienionych systemów: MySQL 5.0.84 [2] albo PostgreSQL 8.1.23 [3], czyli większa szybkość lub większe możliwości. Powody,

dla których zaimplementowano PostgreSQL w ramach projektu, a nie system MySQL zostały wypisane w tabeli 2.1. W tym pojedynku wygrała funkcjonalność.

Tab. 2.1. Porównanie dwóch popularnych systemów zarządzania bazami danych

	MySQL 5.0.84	**PostgreSQL 8.1.23**
Wyzwalacze, funkcje	Brak	Wyzwalacz może zostać wykonany przez każdą z funkcji zdefiniowaną przez jeden z języków proceduralnych (nie tylko domyślny PL/plSQL, lecz również np.: język c). Dodatkowo do jednego zdarzenia na tabeli można przypisać kilka funkcji.
Zrzut bazy	Brak możliwości szczegółowego manipulowania formatem zrzutu. Zrzut bazy ogranicza się tylko do kilku opcji (mysqldump).	Wiele opcji (polecenie pg_dump), w tym również możliwość zrzutu zawartości tabel w postaci pełnych insertów, co umożliwia łatwiejszą konwersję bazy do formatu SQLite – format bazy danych w aplikacji na urządzenia mobilne.

Podstawowym powodem wyboru relacyjnej bazy danych w Postgresie, jest brak w MySQL 5.0.84 funkcji i wyzwalaczy (*triggerów*). Funkcje i wyzwalacze (podrozdział 2.1.3) zostały napisane w celu ograniczenia błędów, mogących powstać w wyniku pośredniego manipulowania danymi w bazie (ochrona przed możliwymi błędami w kodzie źródłowym serwisu WWW). Zaś odpowiedni zrzut bazy danych, ułatwia konwersję bazy z formatu SQL do formatu wymaganego przez aplikację na platformie Android.

2.1.1. Podstawy architektury PostgreSQL [4]

PostgreSQL korzysta z modelu klient-serwer. Sesja pomiędzy klientem a serwerem zawiera trzy współpracujące ze sobą Unix-owe procesy:

- *postmaster* – proces administratora;
- *frontend* – aplikacja klienta (np. program *psql*);
- *backend* – jeden lub więcej serwerów (proces *postgres*).

Administrator (*postmaster*) zarządza zbiorem baz danych (*klastrem*). Aplikacja klienta (*frontend*) tworzy zapytania do biblioteki *libpq* w celu otrzymania dostępu do bazy danych. Biblioteka po otrzymaniu zapytań przesyła je przez sieć do *postmaster-a*, który tworzy nowy proces serwera i łączy *frontend* z *backend-em*. W ten sposób dalsza komunikacja klient-serwer odbywa się bez udziału *postmaster-a*. Ten fakt nie oznacza jednak, że proces administratora jest „ubijany", wręcz przeciwnie, *postmaster* jest uruchomiony zawsze, w celu oczekiwania na żądania przychodzące poprzez bibliotekę z aplikacji klienta.

Biblioteka *libpq* zezwala pojedynczej aplikacji klienta na nawiązanie wielu połączeń do procesu serwera. Aplikacja klienta pomimo takiego zabiegu w dalszym ciągu będzie procesem jednowątkowym. Powodem jest brak wsparcia połączeń wielowątkowych frontend/backend przez bibliotekę *libpq*. W typowej aplikacji klient/serwer, klient i serwer mogą być różnymi hostami. Należy o tym pamiętać, ponieważ pliki dostępne na komputerze klienta, mogą nie być dostępne na serwerze bazy danych.

2.1.2. PL/pgSQL

PL/pgSQL jest językiem proceduralnym zainstalowanym domyślnie wraz z systemem PostgreSQL. Został on w projekcie wykorzystany w celu walidacji wprowadzanych, zmienianych oraz usuwanych danych w ramach funkcjonalności panelu administracyjnego bazą danych (serwisu WWW). Umożliwia on m.in. stosowanie struktur sterujących oraz pętli. Programy napisane w PL/pgSQL nazywane są funkcjami. Można je stosować jako część poleceń SQL albo jako wyzwalacze. Wyzwalacze (ang. *Triggers*) wykonują z góry określoną przez programistę czynność na tabeli lub grupie tabel w momencie wywołania konkretnego typu zapytania

w języku SQL. Przykładowo, można sprawdzać, czy dane wprowadzone w zapytaniu SQL nie są powtórzeniem istniejących już danych w bazie, oraz czy wprowadzane dane są zgodne z właściwym formatem przypisanym do odpowiednich pól w tabeli. W momencie gdy wprowadzone dane są poprawne, można również we właściwy sposób przeedytować dane w innych tabelach. Dzięki przedstawionej walidacji baza jest chroniona przed błędnymi danymi wpisywanymi do tabel a w przypadku poprawnie wykonanych zapytań SQL można dodatkowo zdefiniować konkretne akcje na pozostałych tabelach w bazie danych.

Niepożądane wpisy w bazie powstają zazwyczaj poprzez wielokrotne odświeżanie podstron w serwisie WWW, np.: w momencie kiedy ze wcześniejszej podstrony zostały wysłane dane za pomocą metody POST lub GET zawierające identyfikatory pól o określonej wartości jednej z tabel w bazie, a strona odświeżana odpowiedzialna jest za modyfikację danych skojarzonych z przesłanymi identyfikatorami. W ten sposób poprzez odświeżanie podstrony może dojść do nieplanowego wielokrotnego zmodyfikowania zawartości pól. Taka sytuacja jest zazwyczaj czynnikiem błędu projektowania aplikacji. Im większa strona internetowa, tym większe prawdopodobieństwo na popełnienie błędu w kodzie PHP, czyli większa szansa na błędny zapis w bazie, jak i na usunięcie ważnych danych. Najprostszym rozwiązaniem są właśnie odpowiednie funkcje i wyzwalacze sterujące przepływem danych pomiędzy użytkownikiem końcowym a bazą danych. Dają one pewność, że gdy nastąpi nieprzewidziane w kodzie zachowanie użytkownika serwisu, to dane w bazie nie zostaną zmodyfikowane w niewłaściwy sposób.

2.2. Języki: HTML, CSS, PHP

Zaprojektowany serwis WWW, oprócz tworzenia i modyfikowania bazy danych, umożliwia również przeglądanie ofert operatorów. Funkcja ta jest dostępna, dla każdego internauty, a ponieważ jest on osobą o nielimitowanym czasie, ceniącą profesjonalizm oraz osobą wymagającą dużych możliwości od sprzętu i aplikacji, którymi posługuje się na co dzień, dlatego ważna jest estetyka i funkcjonalność zaprojektowanej strony internetowej, i zarazem prostota w celu zyskiwania przez serwis na popularności. W celu osiągnięcia takich zamierzeń, przy projektowaniu serwisu, posłużyliśmy się kilkoma technikami opisanymi w kolejnych podrozdziałach.

2.2.1. HTML

HTML (ang. *HyperText Markup Language*) jest językiem pozwalającym na podstawową prezentację (wygląd) informacji zawartych wewnątrz strony internetowej, dzięki stosowanym znacznikom (tzw. *tagi*). Poprzez swoją niezależność od systemu operacyjnego oraz sprzętu komputerowego HTML przyczynił się do popularności systemu WWW oraz Internetu.

2.2.2. CSS

CSS (ang. *Cascading Style Sheets*) jest językiem arkuszy stylów używanym do szczegółowej prezentacji elementów dokumentu napisanego w języku znaczników (w tej pracy HTML). Dokument (stronę internetową) możemy podzielić na bloki i przypisać je kolejno do odpowiednich klas. W oddzielnym(-ych) pliku(-ach) definiujemy dla elementów tych klas odpowiedni wygląd w celu uzyskania właściwego wyglądu dla całej strony internetowej. Dzięki takim zabiegom nie musimy kreślić styli dla elementów strony bezpośrednio w języku znaczników, przez co upraszczamy pliki z kodem źródłowym strony, oraz dajemy możliwość łatwiejszych/szybszych zmian w jej wyglądzie.

2.2.3. PHP

PHP (akronim rekursywny „*PHP: Hypertext Preprocessor*”) jest skryptowym językiem programowania pozwalającym na pisanie dynamicznych stron internetowych po stronie serwera HTTP. Dynamiczne strony można pisać za pomocą innych języków skryptowych (np. Perl), jak i również pisząc na platformie ASP.NET posługując się językami takimi jak C# oraz VB. Wybraliśmy jednak PHP, ze względu na jego skalowalność. PHP jest językiem interpretowanym (we wykorzystanej funkcjonalności w ramach projektu), czyli nie następuje kompilacja kodu do postaci kodu maszynowego, lecz wykonywany jest on przez specjalną aplikację zwaną interpreterem PHP. Istnieją również kompilatory PHP, jednak nie zostały wykorzystane w projekcie, ponieważ nie było potrzeby „pakowania” kodu PHP do pliku wykonywalnego. Głównymi zaletami PHP są:

- prosta składnia (oparta w dużej mierze na składni języków C oraz Perl),
- możliwość zagnieżdżania w języku HTML,
- sterowanie logiką na stronie (odpowiedzi serwera na akcje użytkownika),
- kod wykonywany po stronie serwera (użytkownik nie może podglądnąć kodu źródłowego),
- obsługa większości baz danych, w tym wykorzystanej w projekcie PostgreSQL,
- bardzo duża liczba bibliotek oraz funkcji, w tym biblioteka libcurl z funkcją cURL, którą się posłużyliśmy w nakładce, w celu zdobycia informacji na temat nazwy sieci powiązanej z danym numerem telefonu.

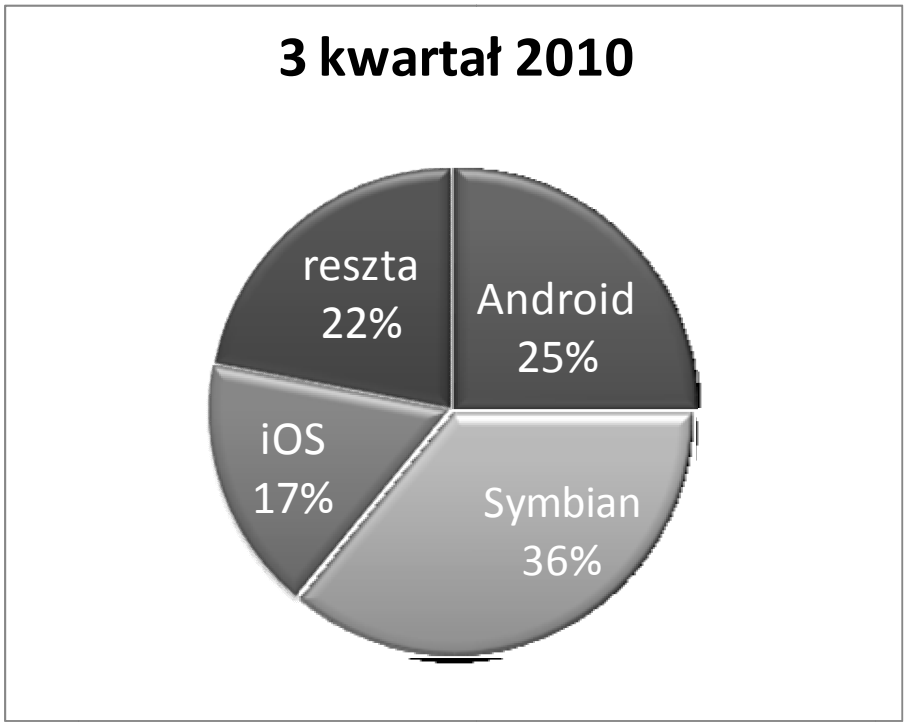
Podsumowując powyższe zalety, można trafnie określić PHP, jako szkielet łączący wszystkie pozostałe techniki w ramach serwisu WWW. PHP został wybrany przez nas jako język programowania strony WWW również ze względu na dużą popularność, jak i dobrą znajomość składni i funkcji dostępnych w ramach języka. Należy nadmienić, że podstawową wadą języka PHP jest stosunkowo niska wydajność aplikacji o dużej ilości kodu źródłowego, ponieważ skrypty PHP są za każdym razem interpretowane, w przeciwieństwie na przykład do języka ASP.

2.3. System operacyjny Android

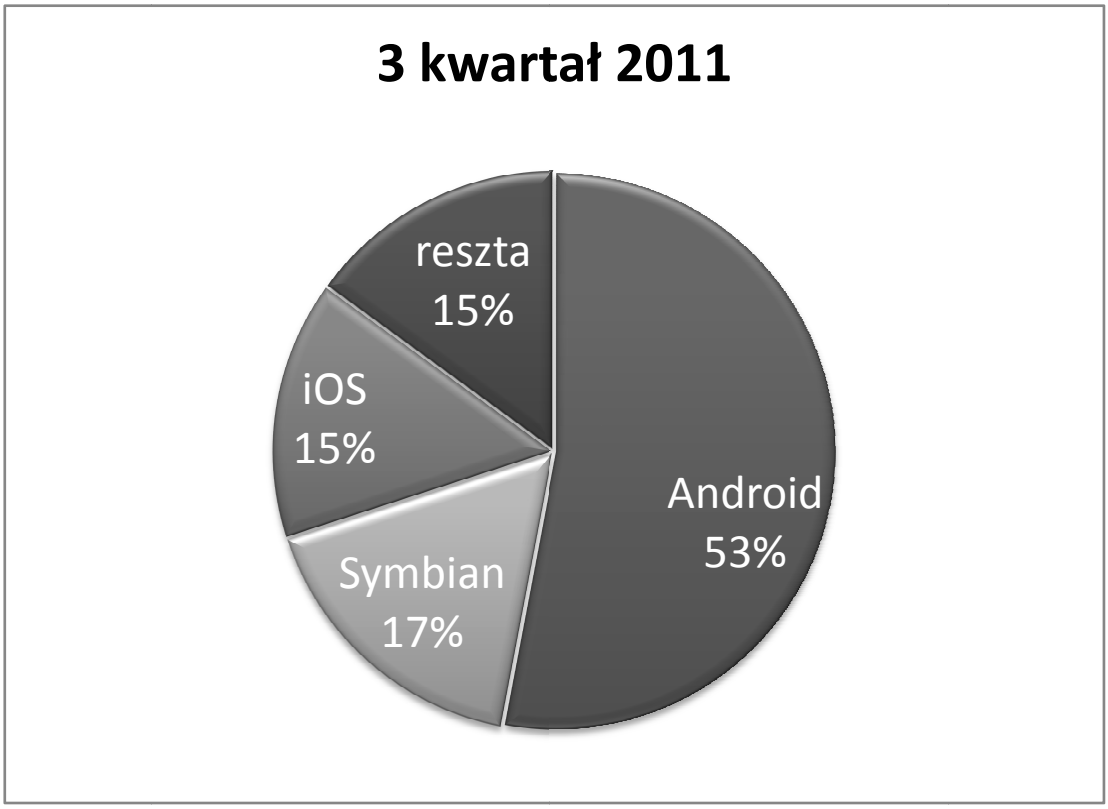
Android jest to otwarty system operacyjny oraz platforma programowa dla urządzeń przenośnych. System ten opiera się głównie o jądro Linuksa 2.6. Wpływa to na jego stabilność i niezawodność. Należy zwrócić również uwagę na język programowania, ponieważ aplikacje na platformę są pisane za pomocą dostępnych narzędzi programistycznych SDK (*Software Development Kit* [5]) oraz języka Java. Wykorzystywany jest *DalvikVirutal Machine* – oprogramowanie za pomocą, którego aplikacje są uruchamiane na urządzeniach. Pliki *.class* Javy są konwertowane na pliki w formacie *.dex* za pomocą *DVM*, wykorzystanie tego zmniejsza zużycie pamięci i szybkości procesora. System wraz z zestawem prostych aplikacji takich jak budzik, obsługa połączeń, wiadomości tekstowych, kalendarza czy Android Market jest powszechnie instalowany w telefonach marki HTC, Samsung, LG, Motorola i innych.

2.3.1. Konkurencyjność

Zaczynając projekt inżynierski trzeba było zdecydować, która platforma będzie bardziej odpowiednia dla aplikacji. Najważniejszym atutem były proste i przejrzyste rozwiązania oraz otwartości oprogramowania, a w szczególności narzędzia programistyczne. Uwzględniona została również popularność, wzrost udziałów i atrakcyjność dla konsumentów, a na tej podstawie została wybrana platforma , która jest najbardziej odpowiednia dla projektu. Poniżej przedstawiony został udział systemów operacyjnych w sprzedanych smartfonach za 3 kwartał 2010 roku oraz 3 kwartał 2011 roku na świecie.



. **Rys. 2.1.** Przedstawienie udziału systemów operacyjnych w sprzedanych smartfonach na świecie w 3 kwartale 2010 roku [6]



Rys. 2.2. Przedstawienie udziału systemów operacyjnych w sprzedanych smartfonach na świecie w 3 kwartale 2011 roku [6]

Z rysunków 2.1 i 2.2 wynika, że przyrost sprzedawanych smartfonów z systemem Android jest bardzo duży. Udziały Androida na rynku światowym zwiększyły się ponad dwukrotnie w ciągu roku. Dane pochodzą ze strony firmy doradczej Gartner zajmującej się strategicznym wykorzystaniem i zarządzaniem technikami w zakresie IT.

2.3.2. Cechy i architektura systemu Android

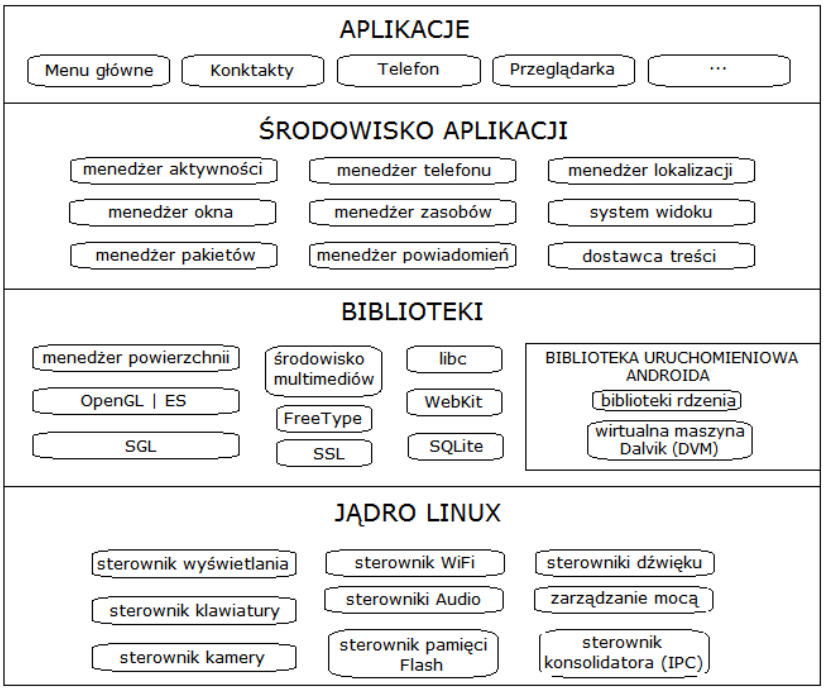
System Android to system otwarty, ze względu na ten fakt można mieć łatwy dostęp do kodu źródłowego. Dodatkowym atutem jest funkcjonalny zestaw narzędzi programistycznych (*Software Development Kit*). Na stronie SDK [5] znajduje się dokumentacja, która jest aktualizowana podczas pojawiania się na rynku nowszych wersji systemu. Znajdziemy tam również przykładowe kody źródłowe aplikacji, na których można się wzorować. Jednak najcenniejszą rzeczą jaką tam znajdziemy, są różnorodne, otwarte biblioteki, które po instalacji wraz ze środowiskiem Eclipse są przydatne w procesie projektowania programu.

Firma Google nie tylko udostępnia możliwość pisania oprogramowania w języku Java. Możemy również skorzystać z języka C++ poprzez pakiet *Native Development Kit* [7] umieszczony na stronie SDK. Jego główną zaletą jest integracja języka C++ z Javą, dlatego NDK jest chętnie wykorzystywany do tworzenia gier lub przyspieszenia obliczeń.

Podstawowa architektura Androida opiera się o jądro systemu operacyjnego Linux 2.6. Istotnym faktem jest to, iż każda aplikacja platformy Android działa we własnej maszynie wirtualnej (DALVIK). Takie rozwiązanie zmniejsza prawdopodobieństwo awarii całego telefonu, jeżeli jakaś aplikacja będzie źle skonstruowana. Dodatkowo wykorzystanie jej zwiększa stabilność i bezpieczeństwo całego systemu poprzez poprawne i zrozumiałe segregowanie, każdej aplikacji osobno. Poniżej przedstawione zostały składowe całkowitej architektury aplikacji [8].

Android składa się z pięciu warstw:

1. Aplikacje.
2. Środowisko Aplikacji.
3. Biblioteki.
4. Biblioteka uruchomieniowa Androida.
5. Jądro Linux.



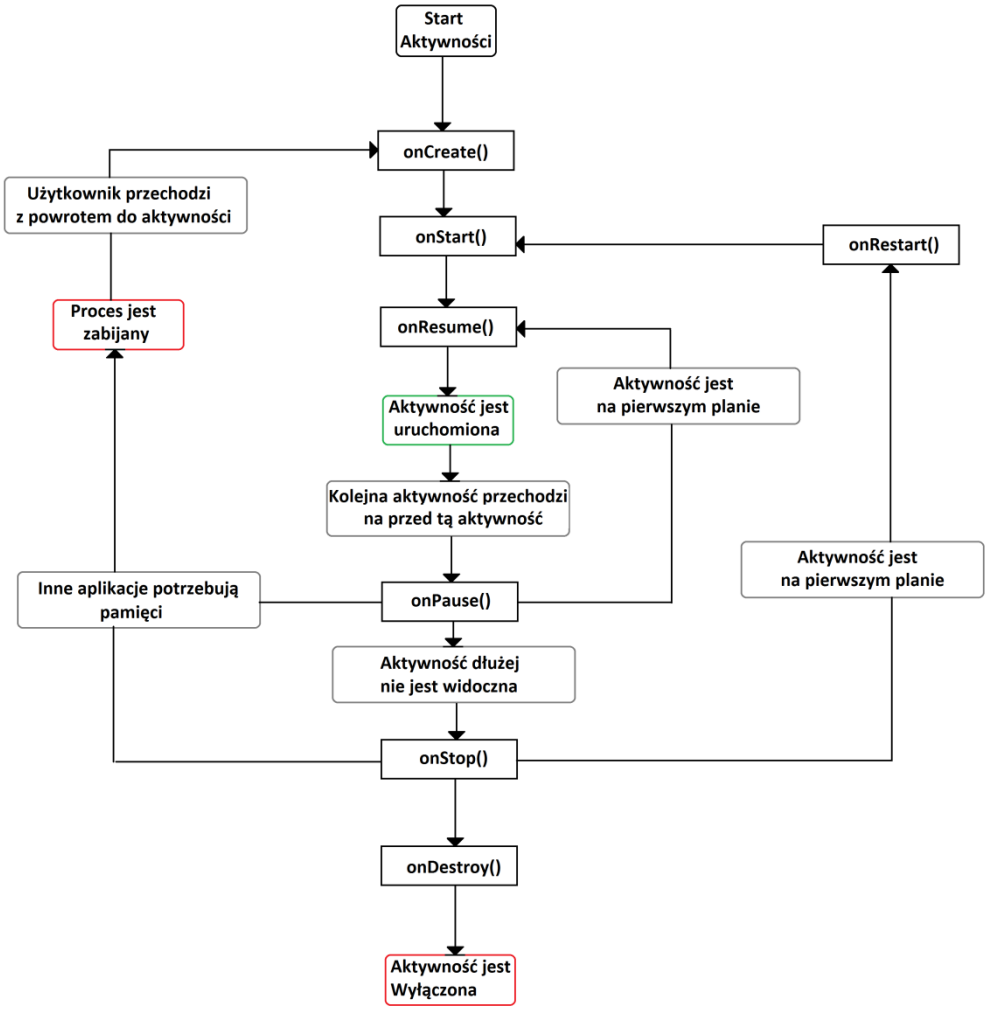
Rys. 3.1. Architektura Androida [9]

W Androidzie obsługiwana głównie jest platforma *Java Standard Edition* (Java SE), ale bez *AbstractWindowToolkita* oraz *Swinga*. Zamiast Java *Virtual Machine* (JVM) do interpretowania uruchomionego kodu bajtowego zastosowano wspomniany wcześniej *Dalvik Virtual Machine*. Różni się on od JVM, ponieważ plik wykonawczy oparty jest na plikach .dex, które są bardziej skompresowane od plików .jar. Takie rozwiązanie pozwala na określenie ograniczeń urządzenia takich jak pamięć, szybkość procesora czy moc. Ważnym faktem jest to, że głównie patenty i licencja wykluczyły wykorzystanie JVM i zmotywowały do stworzenia *DVM* [10].

Istotną informacją dla osoby programującej aplikację na platformę Android jest cykl życia aplikacji. Co oznacza, specyficzny sposób implementacji. Interesujące jest

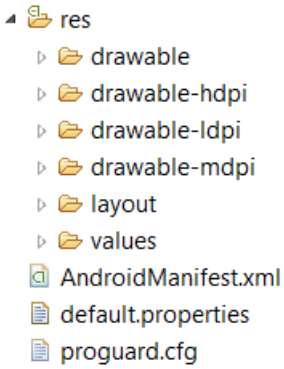
to, że nie można zatrzymać wątku głównego, ponieważ utrzymuje on interfejs użytkownika i jeżeli zostałoby to zrobione, to aplikacja nie funkcjonowałaby poprawnie, prawdopodobnie zakończyłaby się błędem. Rozwiązaniem tego jest stworzenie osobnego wątek do obsługi takich czynności jak pobieranie informacji z bazy danych czy wykonywanie obliczeń. Chodzi tutaj głównie o te operacje, na które program powinien poczekać aż zostaną skończone, by poprawnie funkcjonował.

Dlatego ważne jest zachowanie odpowiednich reguł przy projektowaniu aplikacji. Implementowanie aplikacji powinno przebiegać w taki sposób, by aktywności współgrały ze sobą. Każda jedna może wywołać kolejną, która będzie wykonywała inną określoną akcję. Nowa aktywność wypycha starą na stos i jest przywracana po przyciśnięciu przycisku *BACK*. Każda aktywność, która została złożona na stos, jest powiadamiana o zmianie stanu za pomocą tak zwanej metody *activity'slifecyclecallback.* Mogą być one różne, w zależności od tego w jaki stan przechodzi aplikacja (wznawianie, tworzenie, zatrzymanie, niszczenie). Przykładowo, gdy aktywność się zatrzymuje to powinna zwolnić duże obiekty, takie jak połączenia sieciowe bądź bazodanowe. Za pomocą schematu 2.4 przedstawione zostały relacje cyklu życia aktywności.



Rys. 2.4. Relacje metod w cyklu życia aktywności

Dużą rolę na platformie Android odgrywa rozszerzalny język znaczników, czyli XML. Za jego pomocą kreowany jest układ graficzny dla użytkownika, zapisywane są dłuższe ciągi znaków oraz konfiguracja znajdująca się w pliku *AndroidManifest.xml.* Większość plików *.xml* jest plikami źródłowymi (ang. *resources*), które przetrzymują informacje wykorzystywane w budowie interfejsu użytkownika. Na rysunku 2.5 przedstawiono hierarchiczną strukturę folderów źródeł w każdej aplikacji.



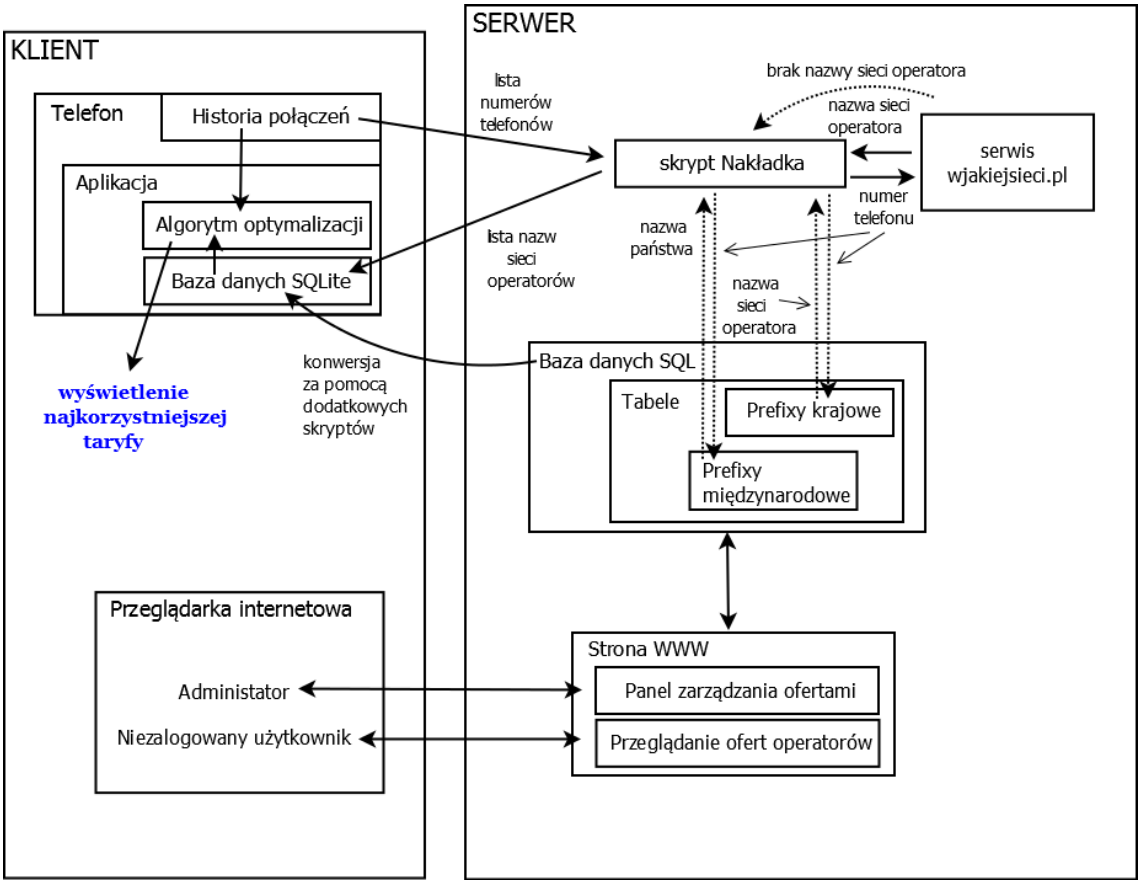
Rys. 2.5. Struktura XML w aplikacji

W pliku *AndroidManifest.xml* znajdują się informacje opisujące wersje platformy Android, prawa dostępu do czynności wykonywanych przez aplikację, wygląd, nazwę w systemie, czy wreszcie deklaracje aktywności.

Platforma Android jest bardzo funkcjonalna, otwarta i łatwo rozwijalna. Charakterystyczna struktura oparta o jądro Linuksa oraz wykorzystanie cyklu życia aktywności, pozwala na zwiększenie niezawodności działania aplikacji. Wykorzystanie języka XML ułatwia proces projektowanie aplikacji poprzez zorganizowaną strukturę.

3. Implementacja

W tym rozdziale przedstawiony zostanie opis funkcjonalnych bloków utworzonych w ramach projektu. Napisane one zostały przy wykorzystaniu technik opisanych w rozdziale 3. Na zaprojektowaną aplikację składa się kilka elementów. Na rysunku numer 3.1 przedstawiamy zależności pomiędzy nimi, w celu pokazania zadań, które zostały im przydzielone, a są opisane w kolejnych podrozdziałach.



Rys. 3.1. Relacje między elementami projektu

Na zaprezentowanym schemacie (rys. 3.1) zostały przedstawione informacje przesyłane między wyszczególnionymi blokami funkcjonalnymi. W centrum znajduje się baza danych na serwerze. Łączy się z nią zaprojektowana strona internetowa, w celu dodawania/modyfikacji lub przeglądania ofert operatorów komórkowych. Aplikacja mobilna korzysta ze wspomnianej bazy skonwertowanej do formatu SQLite. W ten sposób algorytm optymalizacji ma dostęp do wszystkich aktualnych taryf i usług operatorów. Informacje na temat wykonanych połączeń brane są z historii połączeń znajdującej się w telefonie. Następnie w bazie SQLite w wyniku działania aplikacji

tworzona jest dodatkowa tabela zawierająca powiązania każdego z numerów telefonów, do których zostały wykonane połączenia, z odpowiadającymi im nazwami sieci operatorów. Do znajdowania nazwy sieci operatora skojarzonej z danym numerem służy skrypt *Nakładka*. Domyślnie łączy się on w tym celu ze stroną www.wjakiejsieci.pl. Jednak może się okazać, że danego numeru telefonu nie ma w bazach operatorów. Wtedy skrypt odwołuje się do tabel w bazie danych na serwerze, w których zapisane się prefiksy krajowe i międzynarodowe. W zależności od numeru telefonu zwracana jest nazwa sieci operatora lub nazwa państwa, z którego pochodzi ten numer. Algorytm optymalizacji mając zbiór informacji na temat wszystkich ofert i taryf operatorów, długości trwania wykonanych połączeń z każdym z numerów telefonicznych (historia połączeń), oraz powiązanie danego numeru z nazwą sieci (nowopowstała tablica), może wskazać sumy długości połączeń do każdego z operatorów, by następnie wyliczyć najbardziej pasującą (najkorzystniejszą) taryfę wraz z przypisanymi usługami dla danego użytkownika.

3.1. Baza danych

Zanim jednak zostanie przedstawiona strona WWW i aplikacja androidowa, należy opisać „serce projektu”, czyli bazę danych. Jak już napisano, zaprojektowana aplikacja służy do wskazywania najkorzystniejszych ofert wśród operatorów. Tak więc podstawowym zagadnieniem jest odpowiednie przechowywanie zbioru takich informacji, aby zarazem ten zbiór był łatwy do modyfikacji jak i poszerzania o coraz to nowsze usługi czy oferty. Dołączenie takiej funkcjonalności (możliwości manipulowania zbiorem ofert) do aplikacji na urządzenia mobilne ma kilka wad. Po pierwsze: operowania na wielkim zbiorze danych, oraz dodawanie nowych wpisów na ekranie telefonu jest zajęciem mało komfortowym. Aplikacja potrzebuje do poprawnego działania jedynie pliku będącego bazą danych, stosownego interfejsu graficznego i zaimplementowanego algorytmu matematycznego optymalizującego oferty zależnie od predyspozycji użytkownika aplikacji.

Tak więc pozyskanie odpowiedniej bazy w ramach aplikacji na telefon, powinno nastąpić poprzez wykonanie jej innymi narzędziami, niż te oferowane przez telefon, w celu poprawy komfortu/szybkości pracy administratorów. Najprostszym rozwiązaniem jest zaprojektowanie bazy danych na serwerze i skorzystanie z narzędzi

do przekonwertowania jej do odpowiedniego formatu obsługiwanego przez platformę Android. Dostęp do takiej bazy będzie odbywał się poprzez zaprojektowany i opisany w podrozdziale 3.2.5 panel administracyjny.

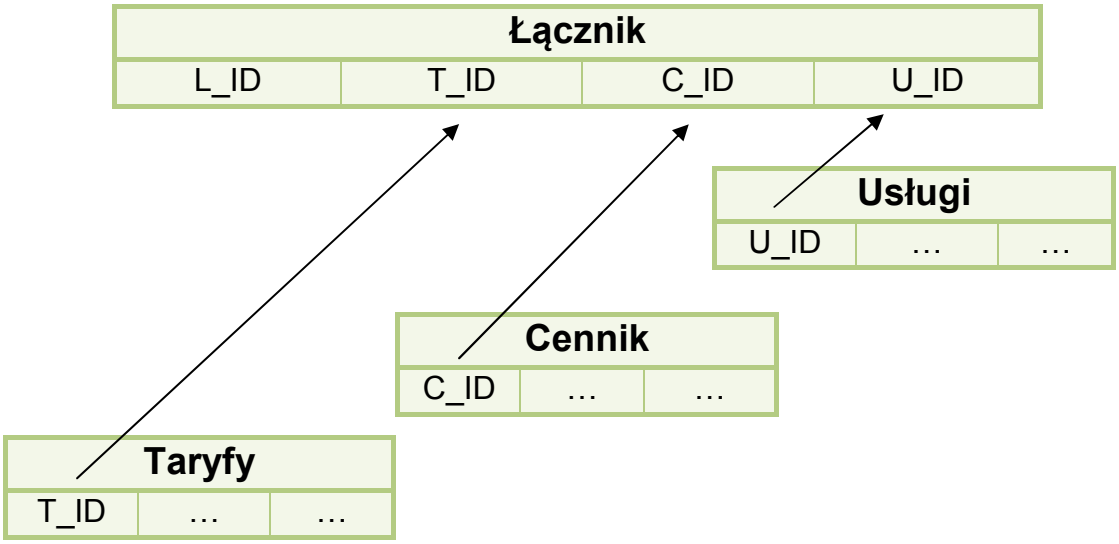
3.1.1. Struktura bazy danych

Zaprojektowana baza danych na serwerze wykorzystuje PostgreSQL (patrz 2.1). Służy do przechowywania usług i cenników operatorów komórkowych. Zawiera szczegółowe informacje na temat ofert powiązanych z taryfami operatorów, takich jak na przykład: okres ważności usługi, do kogo jest skierowana, ceny kosztów za SMS, MMS, minutę połączenia, transfer danych, połączenie międzynarodowe do osób znajdujących się poza granicami kraju, liczba darmowych: SMS, MMS, minut w ramach taryfy lub usługi i wiele innych (zależnie od taryfy lub usługi). Taryfy wraz z usługami zostały uzależnione ponadto od sposobu zawarcia umowy przez użytkownika z siecią danego operatora: przedłużenie umowy, nowy w sieci, przeniesienie numeru.

Początkowa struktura bazy danych zakładała osobne tabele dla każdego z operatorów. Nie było to dobrym rozwiązaniem, ponieważ i tak należałoby, co najmniej na etapie algorytmu w androidowej aplikacji, określić zależności między tymi tabelami. Jednak takie podejście komplikuje strukturę danych przechowywanych w bazie oraz wydłuża działanie algorytmu optymalizacji. Poszukiwana struktura bazy danych powinna zawierać następujące własności:

- uniwersalne tabele (możliwość zapisu zróżnicowanych ofert – opis usług w postaci zbioru liczb),
- właściwe relacje między tabelami,
- brak powielania informacji w różnych tabelach.

Zaprojektowano łącznie 9 tabel. Trzy z nich służą do zapisania kolejno: taryf, cenników oraz usług. Utworzono również tabelę łącznikową, w której w kolejnych rekordach zapisywane są powiązania konkretnych wierszy z trzech powyższych tabel, identyfikowanych poprzez klucze główne. Zależność ta została przedstawiona schematycznie na rysunku 3.2.



Rys. 3.2. Podstawowa relacja między tabelami

Poniżej przedstawiono opis trzech tabel: *Taryfy, Cennik, Usługi*. Kreskowaniem i czcionką pogrubioną będą oznaczone kolumny pełniące rolę kluczy głównych poszczególnych tabel, zaś liniami ukośnymi i kursywą kolumny, które są kluczami obcymi. Pod każdą tabelą zostanie ponadto dodana specyfikacja zawartych w niej pól.

Tab. 3.1. *Taryfy* – informacje podstawowe dotyczące taryf

Taryfy		
Kolumna	**Typ**	**Modyfikator**
t_id	Smallint	not null
o_id	Smallint	not null
Taryfa	character varying (15)	not null
podtaryfa	character varying (45)	not null
kwota_abon_dolad	numeric (9,2)	not null
okres_umowy	Smallint	not null
liczba_darm_uslug	Smallint	default 0

t_id – klucz główny tabeli *Taryfy*;

o_id – klucz obcy tabeli *Operatorzy*, identyfikuje konkretnego operatora;

taryfa – łańcuch znaków ze zbioru {*Abonament*, *Mix*, *Karta*}, określenie sposobu przynależności klienta do operatora;

podtaryfa – pełna nazwa taryfy, przykładowo: *Mixtura darmo wieczory 29*, typ został zdefiniowany jako ciąg znaków o zmiennej długości o liczbie nieprzekraczającej 45;

kwota_abon_dolad – kwota, którą zobowiązał się regularnie płacić użytkownik w ramach podpisania umowy;

okres_umowy – minimalny okres trwania umowy dla danej taryfy oraz podtaryfy, ponieważ długość podpisanej umowy nie wiąże się z różnymi usługami, czy cennikami, lecz różną ceną telefonu branego wraz z podpisaniem umowy;

liczba_darm_uslug – ilość darmowych usług przypisanych do danej podtaryfy ,nie dotyczy wszystkich darmowych usług, ale jedynie darmowych opcjonalnych usług dodatkowych - czyli takich, z których możemy bezpłatnie korzystać w ramach podtaryfy (przykładowo dana taryfa pozwala na bezpłatne korzystanie z dwóch usług z określonego zbioru trzech usług).

Tab. 3.2. *Cennik* – cena za podstawową usługę do użytkownika w sieci operatora *o_id*

Cennik		
Kolumna	**Typ**	**Modyfikator**
c_id	smallint	not null
o_id	smallint	not null
cena_minuta	numeric (9,2)	not null
cena_sms	numeric (9,2)	not null
cena_mms	numeric (9,2)	not null
cena_transfer	numeric (9,2)	not null
Nowy	smallint	default 0
Przejscie	smallint	default 0
przedl_mniej_4	smallint	default 0
przedl_wiecej_4	smallint	default 0
Telefon	smallint	default 0

c_id – klucz główny tabeli *Cennik*;

o_id – klucz obcy tabeli *Operatorzy* (tabela 3.5), identyfikuje operatora, u którego znajduje się osoba, do której wykonano połączenie, wysłany SMS, itp.;

cena_minuta – koszt rozmowy do sieci operatora *o_id* trwającej równą minutę, ponieważ obecnie wszyscy operatorzy krajowi stosują naliczanie sekundowe, więc koszt ceny za minutę jest zawsze na stałym poziomie niezależnie od liczby zestawionych połączeń;

cena_sms – cena za jedną wiadomość tekstową;

cena_mms – cena za jedną wiadomość multimedialną, przeważnie każde rozpoczęte 100kB wiadomości multimedialnej równa się nowej wiadomości MMS;

cena_transfer – cena za transfer 100 kB danych z Internetu;

nowy – wartość ‘1’ oznacza, że użytkownik jest nowy w sieci, bez przenoszenia numeru z sieci od innego operatora;

przejscie – wartość ‘1’ oznacza nowego użytkownika w sieci wraz z przeniesieniem numeru;

przedl_mniej_4 – wartość ‘1’ oznacza, że użytkownik przedłużył umowę, ale w sieci danego operatora jest krócej niż 4 lata;

przedl_wiecej_4 – wartość ‘1’ oznacza, że użytkownik przy podpisywaniu ostatniej umowy znajdował się w sieci obecnego operatora dłużej niż 4 lata;

telefon – wartość ‘1’ oznacza, że użytkownik wraz z podpisywaniem umowy wchodzi w posiadanie nowego telefonu.

Dla danego cennika możliwe są jednocześnie jedynki w kilku polach ze zbioru {*nowy,przejscie,przedl_mniej_4,przedl_wiecej_4*}, np.: *nowy*=’1’, *przejscie*=’1’ - oznacza to, że użytkownik ma dostęp do takich samych usług jeżeli znajdzie się w sieci, jako nowy użytkownik bez przeniesienia jak i z przeniesieniem numeru. Zazwyczaj takie wielokrotne przypisania mają miejsce, jeżeli użytkownik jest posiadaczem telefonu na kartę, gdzie ma dostęp do tego samego zbioru usług i cen za usługi, bez względu na to w jaki sposób stał się posiadaczem taryfy pre-paid.

Tab. 3.3. *Usługi* – specyfikacja konkretnej usługi

Usługi		
Kolumna	**Typ**	**Modyfikator**
u_id	smallint	not null
Nazwa	character varying (20)	not null
Koszt	numeric (9,2)	not null
min_do_all	smallint	not null
min_do_sieci	smallint	not null
sms_do_all	smallint	not null
sms_do_sieci	smallint	not null
internet_mb	integer	not null
Ważność	smallint	not null
ile_numerow	smallint	not null
stan_darmowy	smallint	not null
Hotspoty	smallint	default 0
czy_bezplatna	smallint	default 0
czy_zamienna	smallint	default 0
is_checked	smallint	default 0

u_id – klucz główny tabeli *Usługi;*

nazwa – nazwa usługi zgodna z nazewnictwem danego operatora;

koszt – określa koszt usługi, koszt ten wynosi ‘0’ w momencie gdy usługa jest bezpłatna oraz jest nierozłączną częścią taryfy;

min_do_all – liczba minut do sieci wszystkich operatorów (w ramach usługi);

sms_do_all – liczba SMS do sieci wszystkich operatorów komórkowych;

min_do_sieci – liczba minut w sieci operatora, w której znajduje się użytkownik;

sms_do_sieci – liczba SMS w sieci operatora, w której znajduje się użytkownik;

internet_mb – wielkość pakietu internetowego w MB;

waznosc – liczba dni obowiązywania danej usługi;

ile_numerow – liczba numerów telefonów, których dotyczy dana usługa;

stan_darmowy – liczba SMS w sieci operatora, w której znajduje się użytkownik;

hotspoty – wartość ‘1’, gdy darmowe są hotspoty;

czy_bezplatna – wartość ‘1’, gdy dana usługa jest ze zbioru opcjonalnych usług bezpłatnych;

czy_zamienna – wartość ‘1’, gdy przykładowo darmowe minuty w ramach usługi są zamienne na SMSy;

is_checked – pole dodatkowe na potrzeby aplikacji mobilnej.

Powyżej przedstawione tabele nie są jednak całością prezentującą oferty operatorów. Jeżeli do danego cennika jest przypisanych wiele różnych usług, należy określić, która z nich ma pierwszeństwo. Załóżmy, że dysponujemy w ramach taryfy dwoma usługami: darmowymi minutami do wszystkich i darmowy minutami w sieci. Może się zdarzyć tak, że po wykorzystaniu pewnej liczby minut w ramach własnej sieci, minuty te zostaną odjęte od liczby minut jednej, lub drugiej usługi. Dlatego aby aplikacja (algorytm optymalizacji) wiedziała w jakiej kolejności ma rozpatrywać przypisane do danej taryfy usługi, zdefiniowano tabele *Hierar_uslug*, w której zapisano priorytet usług.

Na tabelę *Hierar_uslug* składają się dwie kolumny, w pierwszej znajduje się identyfikator priorytetu usługi *h_id*, będący zarazem kluczem głównym tabeli. W drugiej kolumnie: klucz obcy tabeli *Usługi* (3.4), czyli identyfikator konkretnej usługi.

Tab. 3.4. *Hierar_uslug* – specyfikuje kolejność wykorzystania usług

Hierar_usług		
Kolumna	**Typ**	**Modyfikator**
h_id	smallint	not null
u_id	smallint	not null

Kolejną podstawową tabelą jest lista operatorów (tabela 3.5). Zawiera identyfikator *o_id* (klucz główny tabeli), powiązany z nazwą operatora *nazwa*. Klucz główny tabeli *Operatorzy* w tabelach *Taryfy*, *Cennik*, *Państwa* pełni rolę klucza obcego, co zabezpiecza przed zdefiniowaniem taryfy lub cenników dla nieistniejących operatorów. Klucz ten oznacza w tych tabelach kolejno: nadawcę, odbiorcę w kraju i odbiorcę poza granicami zestawionego połączenia. Pozostałe pola w danych tabelach wskazują na typ połączenia i koszty z nim związane.

Tab. 3.5. *Operatorzy* – definiuje nazwa operatora oraz właściwy dla niego identyfikator liczbowy

Operatorzy		
Kolumna	**Typ**	**Modyfikator**
o_id	smallint	not null
nazwa	character varying (20)	not null

Czasem się zdarza, że wykonywane połączenia są kierowane poza granice kraju. Ceny połączeń z numerami telefonów stacjonarnych i komórkowych znajdujących się w sieciach zagranicznych operatorów są zwykle znacznie droższe, niż połączenia krajowe. Należy również pamiętać, że przy takich zestawieniach ma miejsce naliczanie minutowe. Oznacza to, że nawet krótkie rozmowy z siecią operatora zagranicznego mogą być kosztowne. W celu umożliwienia rozróżniania numerów, a co za tym również idzie – cen połączeń, z sieciami operatorów zagranicznych została zdefiniowana tabela *Państwa* (tabela 3.6). Korzysta z niej nakładka napisana w PHP.

Tab. 3.6. *Państwa* –koszt każdej rozpoczętej minuty połączenia z danej taryfy danego operatora z numerem zagranicznej sieci telefonicznej

Taryfy		
Kolumna	**Typ**	**Modyfikator**
p_id	Smallint	not null
o_id	smallint	not null
num_kier	smallint	not null
taryfa	character varying (15)	not null
nazwa_panst	character varying (30)	not null
koszt_za_min	numeric (9,2)	not null

p_id – klucz główny tabeli *Państwa*;

o_id – klucz obcy określa sieć operatora, z którego wykonane zostało połączenie;

num_kier – numer kierunkowy danego państwa, umożliwiający rozpoznanie numeru zagranicznego;

taryfa – sposób przynależności danego użytkownika do operatora *o_id*, wartość ze zbioru {*Abonament*, *Mix*, *Karta*}, nie jest to klucz obcy, ponieważ parametru *taryfa* nie określono w żadnej z tabel jako klucz główny;

nazwa_panst – nazwa państwa, na terenie którego znajduje się docelowa (w zestawionym połączeniu) sieć operatora;

koszt_za_min – koszt rozpoczęcia każdej następnej minuty.

Ostatnią tabela utworzoną w ramach struktury bazy danych jest tabela *Numery* (tabela 3.7). Powstała ona w celu zwiększenia skuteczności działania nakładki napisanej w PHP opartej na bibliotece libcurl. Związane jest to z ograniczoną skutecznością znajdowania sieci operatora powiązanego z danym numerem telefonu. Jest to spowodowane niewystępowaniem danego numeru w bazach numerów operatorów, a czasem także z niewłaściwych zakresów zdefiniowanych w *Nakładce* określających czas przeznaczony na wyszukanie nazwy operatora dla jednego z numerów. W wyniku takiego zdarzenia zwracana jest pusta wartość, a nazwa sieci jest przyporządkowana do numeru w oparciu o tabelę *Numery,* oraz tabelę *Państwa* – jeżeli numer zostanie rozpoznany jako numer zagraniczny.

Tab. 3.7. *Numery* – prefiksy numerów wraz z odpowiadającymi im nazwami operatorów sieci komórkowych

Numery		
Kolumna	**Typ**	**Modyfikator**
n_id	integer	not null
nazwa	character varying (30)	not null

n_id – klucz główny tabeli *Numery*, jego wartość odpowiada oryginalnym wartościom prefiksów operatorów;
nazwa – nazwa sieci operatora związana z danym prefiksem numeru.

3.1.2. Funkcje i wyzwalacze PL/pgSQL

Czasem może się zdarzyć, że wprowadzane dane do tabeli w bazie danych są prawidłowe odnośnie typów kolejnych pól. Jednak po dłuższym czasie możliwe jest znalezienie błędów w tabeli, w niektórych polach są liczby w przedziale 2-3, a w innych 2000-3000, pomimo że wartości tych pól powinny się zawierać w przedziale 20-30. Problem powstał, ponieważ administrator przy uzupełnianiu tabeli nie zwrócił uwagi na właściwą jednostkę występującą przy danej zmiennej. Najprostszym sposobem na uniknięcie takich błędów jest napisanie odpowiednich funkcji i wyzwalaczy w bazie danych (przykładowa funkcja i wyzwalacz, użyte zarazem w kodzie zostaną przedstawione poniżej w tabeli 3.8, a wszystkie znajdują się w załączniku E. W funkcjach można zdefiniować warunki dla wprowadzanych danych: określić węższe zakresy, niż te określone przez typ danego pola, jak także zadeklarować konkretny

format wprowadzonych danych dla danego pola w tabeli. W wyzwalaczach określić można sposób i tabele, dla których mają się te funkcje wykonać. Jest to dobre zabezpieczenie danych, ponieważ pozwala ustrzec bazę przed błędami po stronie kodu PHP lub Java Script w przypadku zmiany lub dodawania nowych wartości do tabel w bazie danych. Walidacja w PHP lub Java Script także jest ważna, ponieważ pozwala na pełną kontrolę formatu zwracanych błędów. Jeden nieobsłużony błąd może świadczyć o możliwości niewzięcia pod uwagę przez administratora wystąpienia kolejnych błędów, co nie wpływa korzystnie na jakość i bezpieczeństwo serwisu.

Tab. 3.8. Przykładowa funkcja (1-19) i wyzwalacz (20-23) wykorzystane w ramach projektu

```
1.  CREATE FUNCTION taryfy_val() RETURNS "trigger"
2.   AS $$
3.   begin
4.     if (new.kwota_abon_dolad > 500) then
5.       raise exception 'Niepoprawna wartosc kolumny: Kwota abonamentu';
6.       return null;
7.     end if;
8.     if (new.okres_umowy > 48) then
9.       raise exception 'Niepoprawna wartosc kolumny: Okres umowy';
10.      return null;
11.    end if;
12.    if (new.liczba_darm_uslug > 15) then
13.      raise exception 'Niepoprawna wartosc kolumny: Liczba darm uslug';
14.      return null;
15.    end if;
16.    return new;
17.  end;
18.  $$
19.  LANGUAGE plpgsql;

20.  CREATE TRIGGER taryfy_val_t
21.  BEFORE INSERT OR UPDATE ON taryfy
22.  FOR EACH ROW
23.  EXECUTE PROCEDURE taryfy_val();
```

Przedstawiona funkcja w tab. 3.8 o nazwie *taryfy_val()* służy do sprawdzenia poprawności trzech pól tabeli *Taryfy*: *kwota_abon_dolad*, *okres_umowy i liczba_darm_uslug*.

W linii 20 został stworzony wyzwalacz *taryfy_val_t*, który powoduje wykonywanie się funkcji *taryfy_val()* (linia 23) przed operacją dopisywania nowych lub modyfikacji danych w tabeli *Taryfy* (linia 21).

W bloku funkcji *taryfy_val()* następuje sprawdzenie, czy każda wartość dla trzech zdefiniowanych pól nie przekracza poprawnego zakresu (linie 4,8,12). Jeżeli choć jedna z wartości nie jest poprawna, to wtedy następuje wypisanie adekwatnego wyjątku. Jednocześnie funkcja kończy swe działanie zwracając wartość *null*, tym samym przerwana zostaje również modyfikacja wartości tabeli.

W przypadku, gdy każda z wartości jest poprawna, nie wykonują się instrukcje w wyrażeniach warunkowych *if*, co oznacza, ze funkcja zamiast *null* zwraca *new*. Taka sytuacja odpowiada poprawnemu wykonaniu się funkcji, co kończy się modyfikacją tabeli nowymi poprawnymi wartościami.

Funkcje i wyzwalacze o budowie podobnej do tej przedstawionej w tabeli 3.8, znacznie ograniczają możliwość popełnienia błędów przy wstawianiu lub modyfikacji danych. Klucz główny tabeli *Taryfy* ogranicza ponadto możliwość dodania dwukrotnie tego samego wpisu, ponieważ jest we właściwy sposób inkrementowany po stronie kodu PHP, a nie następuje jego automatyczna inkrementacja po stronie bazy danych. Wartość pola *o_id* powinna być zaś zgodna z wartościami klucza głównego tabeli *Operatorzy*, inaczej modyfikacja lub dodanie nowego wpisu do tabeli także zakończy się niepowodzeniem.

Oprócz dodawania nowych wartości lub modyfikacji już istniejących, można także usuwać niektóre wiersze z tabeli. Niekiedy wiąże się to z usunięciem wartości również z innych tabel. Niepełne usunięcie wszystkie właściwych wartości z tabel w bazie danych, może zaburzyć działanie programu. Problem ten można rozwiązać wykorzystując także funkcje i wyzwalacze, jednak zastosowano w tym celu transakcje (albo zostaną usunięte wszystkie skorelowane wiersze/pola, albo żadne – w przypadku błędu). Przykład zastosowanej transakcji zostanie przedstawiony w podrozdziale dotyczącym opisu ciekawszych fragmentów kodu PHP.

3.2. Serwis WWW

Narzędzie pełniące rolę wymiany informacji pomiędzy administratorem ofertami a bazą danych to strona internetowa napisana przy wykorzystaniu technik HTML/CSS/PHP (podrozdział 2.2).

Stronę internetową została napisana od podstaw, bez korzystania z gotowych wzorców czy systemów zarządzania treścią (z ang. CMS – *Content Management System*). Pozwoliło to na ukształtowanie całej struktury, funkcjonalności i wyglądu strony w ścisły, zdefiniowany sposób. A także pozwala na dalszą nieograniczoną rozbudowę każdego z bloków funkcjonalnych kodu.

Ważnym czynnikiem w budowaniu strony było zoptymalizowanie działań, pozwalające na szybkie zarządzanie całymi blokami danych. Przedstawienie funkcjonalności strony internetowej wraz z interfejsem graficznym pokazano w podrozdziale 3.2.5. Wcześniej opisana zostanie pokrótce struktura strony, następnie fragmenty kodu odpowiedzialne za bezpieczeństwo danych w bazie danych oraz przydatne skrypty, w tym skrypt *Nakładka*.

3.2.1. Struktura panelu administracyjnego

Kod składający się na cały panel administracyjny został podzielony na pliki. W momencie, gdy administrator się zaloguje na swoje konto, jego identyfikator jest zapisywany w zmiennych sesji i od tej chwili może przeglądać podstrony znajdujące się w panelu administracyjnym. Zawartość danej podstrony ładowana jest poprzez plik indeksowy.. Oznacza to, że plik indeksowy pełni kontrolę nad dostępem użytkowników serwisu do innych podstron. Jest to rozsądne rozwiązanie, ponieważ nie wymaga stosowania zabezpieczeń na każdej z podstron z osobna.

Zwykli internauci również mogą korzystać z zaprojektowanego serwisu. Przygotowane dla nich zostały podstrony do przeglądania ofert operatorów komórkowych. Warto zaznaczyć, że takie rozwiązanie nie istnieje jeszcze na rynku. Związane jest to z rozbudowaną i z różnicowaną bazą ofert operatorów, która jest bardzo trudna do zebrania i przedstawienia w uniwersalny sposób.

3.2.2. Ważniejsze fragmenty kodu PHP

W tym podrozdziale postaramy się opisać ważniejsze fragmenty kodu PHP wykorzystane w projekcie strony internetowej. Skupimy się przede wszystkim na aspektach bezpieczeństwa. Przy pisaniu kodu, korzystano z dokumentacji internetowej języka PHP [11].

Mówiąc o bezpieczeństwie mamy na myśli w głównej mierze bezpieczeństwo bazy danych. W tym celu pokazana zostanie walidacja danych (stosowanie wyrażeń regularnych) po stronie PHP, oraz ochrona przed wstrzyknięciem do bazy danych złośliwego kodu (z ang. *SQL Injection*). Jeżeli strona internetowa w swoim kodzie wstawia tekst bezpośrednio z pola tekstowego lub listy rozwijanej do zapytania SQL, można wtedy wykonać na danej bazie, której tyczy się to zapytanie, określonej przez nas operacji. Można przykładowo usunąć wszystkie tabele w danej bazie, albo zmodyfikować lub odczytać wskazane przez nas dane w tabelach. Najprostszym okazuje się jednak zalogowanie na taką stronę, poprzez wpisanie w pola logowania przykładowego tekstu: *‘ or 1=1*. Spowoduje on zamknięcie warunku sprawdzającego dany login czy hasło. Następnie zostaje dopisany do instrukcji *where* danego zapytania SQL fragment, który jest zawsze spełniony, tym samym całe zapytanie SQL wykona się poprawnie, a użytkownik zaloguje się, bez znajomości hasła, na niezabezpieczoną stronę internetową [12].

Kolejno zostanie pokazana obsługa transakcji, które chronią bazę przed błędami mogącymi wystąpić w zapytaniach SQL. Poniżej zaprezentowano fragment kodu (tabela 3.9), w którym wykonywanych jest kilka następujących po sobie zapytań skierowanych do bazy danych wynikających z usuwania danej usługi ze zbioru usług operatorów. Oprócz usunięcia danej usługi z tabeli *Usługi*, trzeba również uporządkować pozostałe tabele: w tabeli *Hierar_uslug* należy usunąć powiązanie usuniętej usługi wraz z danym jej identyfikatorem w hierarchii usług. Transakcje spowodują, że wykonają się wszystkie zapytania SQL w oznaczonym bloku, lub w przypadku jakiegokolwiek błędu – nie wykona się żadne zapytanie. Dzięki temu administrator nie zostanie zaskoczony wpisami w niektórych tabelach dotyczącymi nieistniejących już obiektów.

Walidacja danych z wykorzystaniem PHP

Walidacja odbywa się w momencie modyfikowania i dodawania nowych ofert, usług i innych, korzystając ze strony WWW (panelu administracyjnego). Poniżej w tabelach znajdują się przykładowe dwa fragmenty kodu z pliku *wstawusluge.php*, w celu przybliżenia procesu walidacji danych w technice PHP (wspomniany plik, jak i pozostałe pliki składające się na stronę WWW, znajdują się w załączniku F). Walidacja po stronie bazy danych omówiono w podrozdziale 3.1.2.

Tab. 3.9. Walidacja przy wprowadzaniu nowych danych do bazy

```
1.      if($_POST['nazwa']==NULL) echo "<h3>Wpisz nazwę usługi!</h3>";
2.      else{
3.             if(!sprawdz_nazwe($_POST['nazwa'])){
4.                    echo "<h3>Niepoprawna nazwa usługi!</h3>";
5.                    $booolean = 0;
6.             }else $post_nazwa_u=pg_escape_string($_POST['nazwa']);
7.      }

8.      if($_POST['koszt']==NULL) echo "<h3>Podaj koszt usługi!</h3>";
9.      else{
10.            if(!sprawdz_koszt_uslugi($_POST['koszt'])){
11.                   echo "<h3>Niepoprawna koszt usługi, zakres 0-50!</h3>";
12.                   $booolean = 0;
13.            }else $post_koszt_u=pg_escape_string($_POST['koszt']);
14.     }
```

W tabeli 3.9 zaprezentowano walidację dwóch pól tekstowych. W pierwszej linii następuje sprawdzenie, czy dane pole zostało uzupełnione. Jeżeli nie, to „wyrzucany jest" stosowny komunikat, odpowiednio sformatowany w arkuszach styli (CSS). W przeciwnym razie (linie 2 i 9), jeżeli pole tekstowe zostało uzupełnione, to sprawdzane jest czy wpisana wartość pasuje do wzorca zdefiniowanego w odpowiednich wyrażeniach regularnych. W naszym przypadku odpowiadają za to funkcje: *sprawdz_nazwe(...)* i *sprawdz_koszt_uslugi(...)* przedstawione poniżej. Jeżeli wartość jest prawidłowa, to następuje przypisanie do zmiennych *$post_nazwa_u* i *$post_koszt_u* wartości funkcji *pg_escape_string(...)* z tych pól. Funkcja *pg_escape_string(...)* stawia przed znakami specjalnymi takimi jak na przykład apostrofy i cudzysłowy znaki backslash, dzięki czemu nie są one traktowane od tej pory przez język SQL jako znaki specjalne, tylko jako zwykły tekst. Tym samym wpisanie właściwego tekstu ze znakami specjalnymi w polu tekstowym na stronie, zapobiega atakowi *SQL Injection*. Oznacza to, że system staje się bardziej bezpieczny.

Poniżej przedstawiono wyrażenia regularne wykorzystane we fragmencie kodu z tabeli 3.9. Funkcje te zwracają jako wynik liczbę wystąpień argumentu funkcji w danych wyrażeniach regularnych.

Tab. 3.10. Wyrażenia regularne

```
function sprawdz nazwe($nazwa uslugi)
        {
                $nod1 = strtolower($nazwa uslugi);
                return preg_match("/^[a-ząćęłńóśźż]{1}[0-9a-
                                ząćęłńóśźż_\.-\s]+$/i", $nod1);
        }


function sprawdz_koszt_uslugi($liczba)
        {
                return preg_match('/^(([1-4]?[0-9](\.[0-9]{1,2})?)|(50))$/', $liczba);
        }
```

W bloku pierwszej funkcji w tabeli 3.10, wykorzystanej do sprawdzenia poprawności nazwy usługi, posłużono się dwoma podstawowymi funkcjami bibliotek PHP. Funkcja *strtolower(...)* służy zamienieniu wszystkich dużych liter na małe w argumencie wywołania funkcji *sprawdz_nazwe(...)*. Zaś funkcja *preg_match(...)* porównuje wzorce z ciągiem znaków zwróconych przez funkcję *strtolower(...)*. Budowa obydwóch wzorców została przedstawiona poniżej.

Wzorzec znajdujący się w funkcji *sprawdz_nazwe(argument)* sprawdza czy *argument* jest co najmniej dwuznakową nazwą, w której pierwszym znakiem jest litera. Zaś funkcja *sprawdz_koszt_uslugi(argument)* zwróci pozytywny wynik, w momencie gdy *argument* będzie liczbą z przedziału domkniętego od 0 do 50.

Podsumowując, wykonane walidacje pól tekstowych na stronie chronią przed atakiem *SQL Injection*, ponieważ napisane funkcje nie akceptują znaków specjalnych. Można było zatem nie stosować w powyższych przypadkach funkcji *pg_escape_string(...)*. Jednak może się zdarzyć sytuacja konieczności umieszczenia w którymś z pól tekstowych cudzysłowu lub apostrofu. Wtedy funkcja spełni swoje zadanie, ponieważ baza nie będzie chroniona wyrażeniem regularnym. W takim wypadku uzasadnione jest stosowanie podwójnego zabezpieczenia.

Transakcja jest typowym narzędziem języka SQL. Jest to zbiór zapytań skierowany do bazy, który wykona się w całości lub wcale. Dzięki czemu pewne jest, że jeśli z dwóch skorelowanych zapytań wykona się pierwsze, to na pewno wykona się

i drugie. W tabeli 3.11 umieszczony został fragment pliku *wstawusluge.php* z przykładem transakcji.

Tab. 3.11. Przykładowa transakcja SQL w kodzie PHP

```
441. pg query("BEGIN WORK");

443. if($dont_refresh==1){

     /****************** BLOK 1 - AKTUALIZACJA TABELI HIERAR_USLUG ******************/
444.   for($mm=0;$mm<count($hid_hierar);$mm++){
445.      pg_query("update hierar_uslug set u_id=".$uid_hierar[$mm]." where
                                                  h id=".$get hid for uid[0]."");
446.      $get_hid_for_uid[0]++;
447.   }
448.   pg_query("delete from hierar_uslug where h_id=".$get_hid_for_uid[0]."");
449.   pg query("delete from hierar uslug where u id=$i");
     /*****************************************************************************/


     /** wyciagam amp_id, dla ktorych zostana usuniete uslugi w tabeli lacznik */
454.   $get_before = pg_query("select amp_id from lacznik
                                              where u_id=$i group by amp_id");
455.   while($row get=pg fetch row($get before)){
456.      $get amp[] = $row get[0];
457.   }
458.   if(!isset($get_amp[0])) $get_amp = array();


     /** usuwam konkretny wiersz z tabeli uslugi, oraz nastepnie uaktualniam tabele
       * Lacznik, wstawiajac w miejsce usunietej uslugi 0 */
461.   pg_query("delete from uslugi where u_id=$i");
462.   pg_query("update Lacznik set u_id=0, hierar=0 where u_id=$i");


467.   for($ile amp=0;$ile amp<count($get amp);$ile amp++){
       /** kod, ktory znajduje sie domyslnie w tym bloku sluzy do usuniecia
         * powtarzajacych sie wierszy w tabeli lacznik */
484.   }
485. }//KONIEC if($dont_refresh==1)
486. pg_query("COMMIT");
```

W tabeli 3.11 w liniach 441 i 486 zostały określone granice transakcji. Linie o numerach 444-484 prezentują blok uaktualniający stan kilku tabel. Następuje to po naciśnięciu przez administratora przycisku usuwającego konkretną usługę. Zmienna *$dont_refresh* została zdefiniowana w bloku poprzedzającym przedstawioną transakcję. Jej wartość opiera się na stanie bazy danych w momencie naciskania przycisku i stanie aktualnym, dlatego nie nastąpi ponowne usunięcie którejś z wartości w chwili odświeżania strony. Mogłaby taka sytuacja nastąpić, ponieważ identyfikatory hierarchizujące usługi zmieniają swoje wartości dynamicznie wraz z usuwanymi usługami. Wielokrotne odświeżanie skutkowałoby usunięciem wszystkich wpisów z tabeli *Hierar_usług*.

Transakcja dotyczy wszystkich linii, w których występuje zapytanie SQL (*pg_query*). Czyli, jak można odczytać z kodu, modyfikacji wymaga zawartość trzech tabel: *Usługi, Łącznik, Hierar_usług*. W bloku oznaczonym numerem pierwszym,

modyfikowane są wiersze w tabeli *Hierar_usług*, dla wszystkich usług o przypisanym kluczu głównym *h_id* większym niż klucz usuwanej usługi. Ma to miejsce w przedstawionym fragmencie kodu w linii 445. Nowe przypisania są tworzone w ten sposób, żeby pomiędzy kolejnymi wartościami *h_id* różnica wynosiła 1. W liniach 448 i 449 znajdują się dwa zapytania SQL. Czyszczą one tabelę z informacji zostawionych po usuwanej usłudze. Pierwsze zapytanie:

```
pg_query("delete from hierar_uslug where h_id=$get_hid_for_uid[0]");
```

usuwa z tabeli *Hierar_usług* wiersz, w którym znajduje się największa wartość identyfikatora *h_id*. Wynika to z faktu, że usuwając jedną usługę z hierarchii usług, zmniejszana jest liczba zhierarchizowanych usług o jeden, tym samym mniej jest również wykorzystanych identyfikatorów *h_id*. Drugie zapytanie:

```
pg_query("delete from hierar_uslug where u_id=$i");
```

usuwa zaś wiersz, w którym znajduje się identyfikator usuwanej usługi. Usuwając jedną usługę, musimy usunąć dwa wpisy z tabeli, w której istniał tylko jeden wpis z identyfikatorem usuwanej usługi. Wynika to jedynie z utworzenia nowych przypisań w linii 445. Jeżeli usuwana usługa była najniżej w hierarchii ważności usług (największa wartość *h_id*), to przypisania nie będą zmieniane, a całość informacji o danej usłudze zostanie usunięta w pierwszym zapytaniu.

W zapytaniu z linii 461 usuwany jest wpis o danej usłudze z tabeli *Usługi*:

```
pg_query("delete from uslugi where u_id=$i");
```

W następnej linii modyfikacji ulega wpis w tabeli *Łącznik* powiązany z usuwaną usługą:

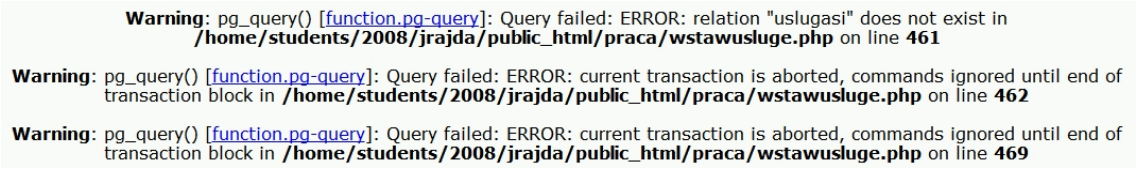
```
pg_query("update Lacznik set u_id=0, hierar=0 where u_id=$i");
```

Można zauważyć, że w tym wierszu zerowany jest identyfikator usuwanej usługi *u_id* oraz identyfikator *hierar* – zmienna utworzona do przyszłych wykorzystań na potrzeby aplikacji mobilnej na platformę Android. Nie następuje od razu usuwanie danych wierszy, tylko zerowanie, ponieważ w czasie usuwania powiązania z daną usługą, nie powinny jednocześnie usunąć się powiązania danej taryfy z danym cennikiem. Przypominając, wiersz tabeli *Łącznik* zawiera identyfikator główny *l_id,* oraz identyfikatory podstawowe *t_id, u_id i c_id.* My w pierwszej fazie jedynie zerujemy *c_id.*

Druga faza modyfikacji tabeli Łącznik ma miejsce w liniach 454-458 i bloku 467-484, jednak ze względu na obszerność kodu zapis w tabeli 3.11 został skrócony. Cały fragment kodu znajduje się w skrypcie *wstawusluge.php* w załączniku F.

Kod ten sprawdza istnienie powtarzających się wierszy w tabeli *Łącznik*, w których nie istnieje powiązanie taryfy i cennika z usługą. W przypadku znalezienia takich wierszy, są one usuwane. Sytuacja taka następuje w momencie usuwania kilku usług, które przynależą do jednej pary: taryfa, cennik. W chwili dowiązania do danej taryfy i cennika innej nowej usługi, zera z pierwszej fazy modyfikacji zostają nadpisane na identyfikator nowej usługi *u_id*. Jeżeli puste przypisania nie istnieją w tabeli *Łącznik*, ponieważ nie były wcześniej usuwane powiązane usługi, to dodawane są nowe wiersze do tabeli *Łącznik* – wiążące daną taryfę, cennik oraz usługę.

We fragmencie kodu przedstawionym w tabeli 3.11 znajduje się kilka zapytań SQL. W momencie jakby jedno z nich się nie wykonało z powodu nieoczekiwanego błędu, mogłoby to skutkować niespójnością informacji w bazie danych. Dlatego konieczne jest posłużenie się transakcją, która dzięki swemu działaniu uchroni bazę danych przed możliwymi błędami. Na rysunku 3.3 został zaprezentowany komunikat zwrócony w wyniku błędnie podanej nazwy tabeli w jednym z zapytań SQL objętym omawianym blokiem transakcji. Można zauważyć, że pozostałe zapytania w danej transakcji są przerwane. Również nie wykonają się zapytania poprzedzające błędne zapytanie z linii 461.



Rys. 3.3. Odpowiedź na błąd w zapytaniu SQL objętym transakcją

Jeżeli nie zastosowano by w tym przypadku transakcji, to zostałyby usunięte wszystkie powiązania danej usługi, prócz usunięcia jej z tabeli *Usługi*.

Właściwa walidacja pól tekstowych (oraz co ważne także pól typu *select*) może uchronić przed usunięciem, modyfikacją, lub podglądem bazy danych przez osoby niepowołane. Stosowanie transakcji uchroni bazę przed błędami mogącymi wystąpić w wyniku nieprawidłowej konstrukcji zapytań objętych transakcją.

3.2.3. Dodatkowe skrypty

Kod źródłowy omówionych poniżej skryptów znajduje się w załączniku G.

Zamiana bazy z formatu SQL na format SQLite

W celu przekonwertowania bazy danych na format umożliwiający, korzystanie z niej z poziomu telefonu, zostały napisane trzy skrypty. Wykorzystano również aplikację o nazwie *Sqlitebrowser*.

Etap 1:

Należy zapisać bazę danych do pliku, wykonując polecenie:

```
pg_dump –d nazwa_bazy_danych > nazwa_pliku.sql
```

Parametr *–d* służy do zapisania bazy z pełnymi insertami, co upraszcza dalsze etapy konwertowania bazy.

Etap 2:

- należy wkleić w obszarze tekstowym w oknie przeglądarki internetowej pod adresem skryptu *wyciaganie_tabel_ze_schematu_bazy.php* zawartość pliku *plik.sql*, w wyniku działania skryptu zwrócone zostaną linie, które odpowiadają za tworzenie tabel (*Create table ...(...);*) w formacie SQL,

- otrzymane linie należy zaś wkleić w obszarze tekstowym w oknie przeglądarki pod adresem skryptu o nazwie *sql_pod_db.php*, dostając w rezultacie wiersze do tworzenia tabel o odpowiednim formacie i typie poszczególnych pól zgodnych ze składnią SQLite, wynik należy zapisać do pliku o nazwie *dane.sql*,

- w oknie przeglądarki internetowej w obszarze tekstowym skryptu *wyciaganie_danych_z_bazy_w_postaci_insertow.php* również należy wkleić zawartość pliku *plik.sql*, skrypt w odpowiedzi wyświetli wszystkie wstawione wiersze do tabel w bazie danych, które należy dopisać na koniec pliku *dane.sql*,

- po skorzystaniu z programu *Sqlbrowser*, importując do nowoutworzonego w nim pliku o rozszerzeniu .db zawartość pliku *dane.sql*, otrzymuje się właściwą strukturę bazy danych wraz ze wszystkimi wpisami potrzebnymi dla naszej aplikacji na platformie Android.

Import danych do bazy

Skrypt o nazwie *import_danych_do_bazy.php* pozwala na odtworzenie tabel w bazie danych, zmodyfikowanych testami lub błędami administratora. Potrzebna jest do tego jedynie kopia pełnego stanu bazy danych zapisana wcześniej do pliku. Jest to bardzo szybkie rozwiązanie, ponieważ wystarczy w oknie przeglądarki wkleić odpowiednie wpisy z pliku, a po kilku sekundach zostaną one odtworzone w bazie. Obszar, w którym są wklejane wiersze nie ma ograniczeń co do liczby znaków, więc wklejenie tysięcy wierszy nie stanowi problemu. Wykonanie podobnego zadania w konsoli wymaga cierpliwości, ponieważ przy wklejaniu w okno konsoli większej liczby rozbudowanych wierszy powstają błędy. Oczywiście można w konsoli przekierować plik tekstowy, jednak wymagane jest uruchomienie konsoli oraz utworzenie odpowiedniego pliku (nowy plik, ponieważ wymagane może być odtworzenie rekordów tylko w jednej wybranej tabeli). Wklejenie zwartości całej tabeli do okna skryptu, pomimo brakujących w tabeli w bazie tylko kilku wpisów, spowoduje uzupełnienie jedynie tych kilku brakujących wpisów. Pozostałe zostaną zignorowane.

Skrypty napisane w celu rozszerzenia przyszłej funkcjonalności projektu

Skrypty o nazwach: *dodawanie_minut_cz1.php, dodawanie_minut_cz2.php, ostatnie_cyfry.php* i skrypt *gNN.php* (załącznik G – odpowiednio zmodyfikowana nakładka z rozdziału 3.2.4) służą do wyciągnięcia z bilingów telefonicznych (załącznik H) informacji na temat czasu połączeń/wysłanych SMS do każdej z sieci operatorów komórkowych. Obliczane zostają również informacje na temat wielkości pobranych danych z Internetu. Posiadając takie informacje użytkownik mógłby skorzystać z naszej aplikacji na telefonie innego użytkownika, uzupełniając jedynie konkretne pola tekstowe w aplikacji zgodnie z ich opisem. Skrypty nie funkcjonują obecnie na stronie internetowej, ponieważ nie implementują jeszcze obsługi bilingów od wszystkich podstawowych operatorów komórkowych.

3.2.4. Skrypt Nakładka

Algorytm optymalizacji w zaprojektowanej aplikacji na platformę Android wymaga poprawnego skojarzenia nazwy sieci operatora komórkowego z każdym numerem telefonu znajdującym się w historii telefonów. W tym celu powstał skrypt o nazwie *Nakładka*.

Połączenie internetowe aplikacji z podstroną *Nakładki* odbywa się poprzez rozbudowany adres URL składający się z pewnej sekwencji numerów telefonów. W odpowiedzi (*HTTP Request*) *Nakładka* wyświetla na danej podstronie nazwy sieci skojarzone z numerami telefonów w adresie URL. Aplikacja na telefonie analizuje te informacje, zapisując je do nowoutworzonej tabeli w bazie SQLite.

Nakładka musi rozpoznać nazwy sieci operatorów komórkowych powiązanych z polskimi numerami oraz określić państwo w przypadku numerów zagranicznych. Pierwsza czynność odbywa się z wykorzystaniem serwisu *Wjakiejsieci* oraz tabeli *Numery* w bazie danych. Zaś do określenia nazwy państwa, do którego zostało wykonane połączenie, posłużono się tabelą *Państwa*.

Nakładka składa się trzech podstawowych części. W pierwszej następuje modyfikacja formatu numerów podanych w adresie URL, oraz zapisanie ich do tablicy. W kolejnym etapie *Nakładka* łącząc się ze stroną *www.wjakiejsieci.pl* zapisuje odpowiedź tej strony na przesłanie właściwego numeru telefonu. Wyrażenie regularne wyznacza z zapisanej odpowiedzi nazwę sieci operatora dla danego numeru. Operacja ta powtarzana jest dla każdego z numerów telefonów podanych w adresie URL. Jeżeli numer wysłany przez *Nakładkę* na stronę nie istnieje w bazach danych operatorów oznacza to, że dana osoba nie przenosiła numeru do innej sieci. W tym przypadku wykonuje się trzecia część skryptu. Pełny kod drugiej części algorytmu został przedstawiony w tabeli 3.14.

W ostatniej części przyporządkowywane są nazwy sieci operatorów dla wszystkich numerów, które pozostały bez powiązania z odpowiednią siecią z części drugiej. Są to numery, których nie ma w bazach polskich operatorów komórkowych oraz wszystkie numery sieci operatorów zagranicznych. W następujących po sobie warunkach ma miejsce sprawdzenie liczby znaków, prefiksu międzynarodowego

(+… lub 00…), oraz prefiksów sieci krajowych operatorów komórkowych. W ten sposób dany numer zostanie przypisany do którejś z sieci krajowych operatorów komórkowych, do któregoś z państw, albo będzie po prostu krajowym numerem stacjonarnym lub ostatecznie błędnym numerem – dla takiego numeru zostanie wyświetlone słowo *brak*.

W tabeli 3.13 przedstawiono wyniki działania *Nakładki* dla przykładowych kilkunastu numerów telefonów.

Tab. 3.13. Poprawność działania *Nakładki*

Id	Numer	Nazwa sieci
1	788522527	Play Mobile
2	48788522527	brak
3	+48788522527	Play Mobile
4	0048788522527	Play Mobile
5	8752472	brak
6	338752472	Stacjonarne
7	48338752472	brak
8	+48338752472	Stacjonarne
9	0048338752472	Stacjonarne
10	002532345664	Dżibuti
11	0018093345664	Dominikana
12	+14434343434	Kanada

Numery o identyfikatorach 2, 5 i 7 są niepoprawne, ponieważ składają się z nieprawidłowej liczby znaków. Numery telefoniczne w Polsce, nie biorąc pod uwagę numerów telefonów alarmowych czy informacyjnych, mają nie mniej niż 9 znaków. Nie istnieją również w Polsce numery o liczbie znaków równej 11 (błędne numery 2 i 7). Kod źródłowy *Nakładki* znajduje się pod nazwą *getNetworkName.php* w załączniku I.

W *Nakładce* korzystaliśmy z narzędzia o nawie cURL [13] (*Client URL Request Library),* które oparte jest na bibliotece *libcurl.* CURL pozwala na pobranie lub przesłanie podstron/plików przy użyciu składni URL, co zostało wykorzystane w celu pobrania informacji z serwisu *Wjakiejsieci*.

Tab. 3.14. Przykład użycia biblioteki cURL

```
1    $url = "http://www.wjakiejsieci.pl/";
2    $ch = curl_init($url);
3    curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
4    curl_setopt($ch, CURLOPT_TIMEOUT, 3);
5    curl_setopt($ch, CURLOPT_USERAGENT, "Mozilla/5.0 (Windows NT
           6.1; WOW64; rv:7.0.1) Gecko/20100101 Firefox/7.0.1");
6    curl_setopt($ch, CURLOPT_REFERER,$url);
7    curl_setopt($ch, CURLOPT_POST, 4);
8    curl_setopt($ch, CURLOPT_POSTFIELDS,array('phone'=>$num_phone,
                                          'prefix'=>'%2B4'));
9    $result = strip_tags(curl_exec($ch));
10   curl_close($ch);
11   $wzorzec = '/(?<=or:)[a-zA-Z\-\s:]*(?= Dob)/';
12   if(preg_match_all($wzorzec,$result,$matches)!=0)
       echo "<p id=$i>".$matches[0][0]."</p>";
```

W drugiej linii kodu źródłowego z tabeli 3.14 następuje rozpoczęcie sesji z jednoczesnym połączeniem się z serwisem *Wjakiejsieci.* Następnie w liniach od 3 do 8 przy użyciu funkcji *curl_setopt* zdefiniowano podstawowe parametry połączenia umożliwiające uzyskanie nazwy sieci skojarzonej z konkretnym numerem telefonu (*$num_phone*). W linii 9 następuje wykonanie bloku funkcji *curl_setopt* wraz ze zwróceniem wyniku w postaci strony internetowej. Wykorzystując wyrażenie regularne „wyciągnięto” z otrzymanej strony nazwę sieci skojarzoną z numerem telefonu.

Pozostałe skrypty powstałe w ramach pracy oraz skrypty przedstawione powyżej zostały zamieszczone w załączniku G. Skrypt *Nakładka* dodany został jako załącznik I.

3.2.5. Interfejs użytkownika panelu administracyjnego

W kolejnych akapitach przedstawiono od strony praktycznej powstały serwis internetowy. Na początku zostanie pokazany w swojej funkcjonalności panel administracyjny, a następnie podstrony umożliwiające niezalogowanym użytkownikom przeglądanie ofert operatorów.

Panel administracyjny

Administrator po zalogowaniu się na swoje konto ma możliwość edycji wszystkich informacji dotyczących taryf, usług i ofert operatorów komórkowych zebranych w bazie danych opisanej w rozdziale 3.1.

1. Menu nawigacyjne

Po lewej stronie ekranu znajduje się menu nawigacyjne (rysunek 3.4 a)) pomiędzy kolejnymi podstronami panelu administracyjnego.

(a) (b)

Rys. 3.4. a) Menu z hiperłączami do danych podstron,
b) Dodawanie do bazy nowego operatora

W menu znajduje się osiem stałych przycisków oraz przycisk *Cofnij* – pojawiający się w chwili zagłębiania administratora w widok podstron dostępnych jedynie z konkretnej podstrony. Pod słowem *Międzynarodowe* kryje się w graficzny sposób przedstawiona zawartość tabeli *Państwa*. Można przechodzić pomiędzy taryfami, odczytując przypisane koszty, które można również edytować.

2. Podstrona *Operatorzy*

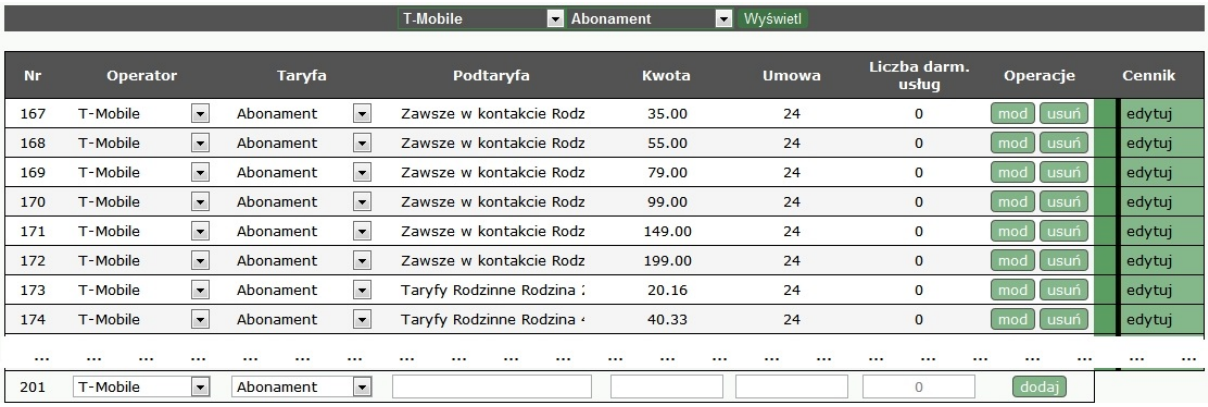
Klikając przycisk *Operatorzy* znajdziemy się na podstronie umożliwiającej edycję istniejących już nazw sieci operatorów w bazie danych (rysunek 3.4 b)). W przypadku błędnej nazwy zostanie zwrócony komunikat przedstawiony na rysunku 3.5. Komunikaty błędów będą wyświetlane zawsze wtedy, gdy do któregoś z pól w całym panelu administracyjnym zostanie wpisana błędna wartość. Więcej informacji na temat walidacji pól można znaleźć w rozdziałach 3.1.2 i 3.2.2. W przypadku usunięcia sieci danego operatora (*usuń*), wyświetlony zostanie komunikat o tabelach, w których są przypisania taryf/usług/innych z daną siecią.

Po naciśnięciu *pokaż taryfy* nastąpi przejście do zakładki *Taryfy,* pod którą znajdą się wyszczególnione taryfy dla wskazanego operatora.

Rys. 3.5. Komunikat w przypadku błędnej nazwy sieci operatora

3. Podstrona *Taryfy*

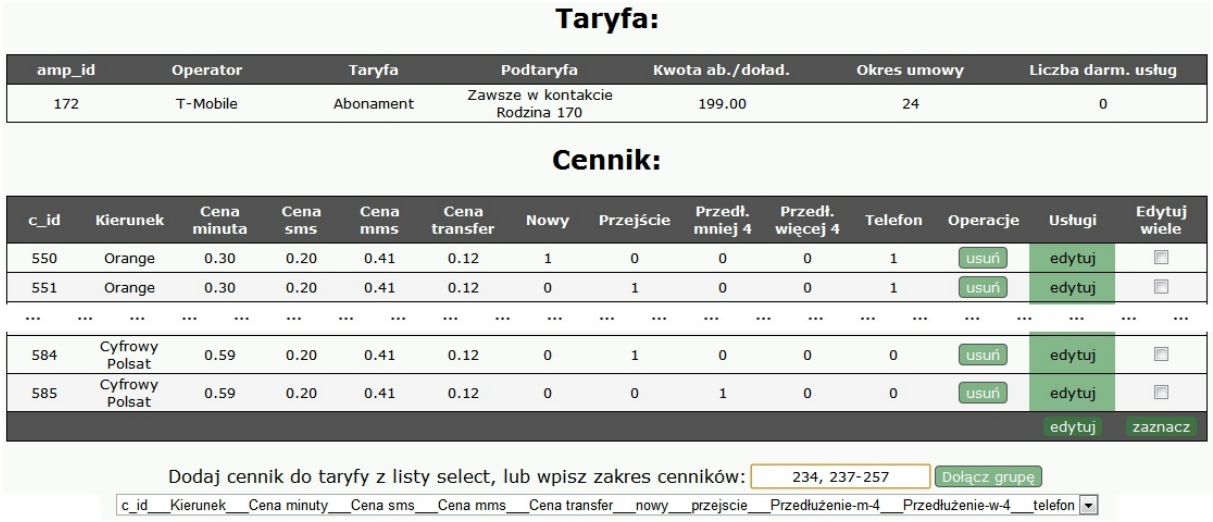
Na tej podstronie można utworzyć nowe taryfy lub zmodyfikować już istniejące. Możliwe jest również skojarzenie danej taryfy z właściwymi cennikami i usługami, o ile wcześniej zostały już takie utworzone w zakładkach *Cennik* i *Usługi*.



Rys. 3.6. Fragment tabeli *Taryfy*

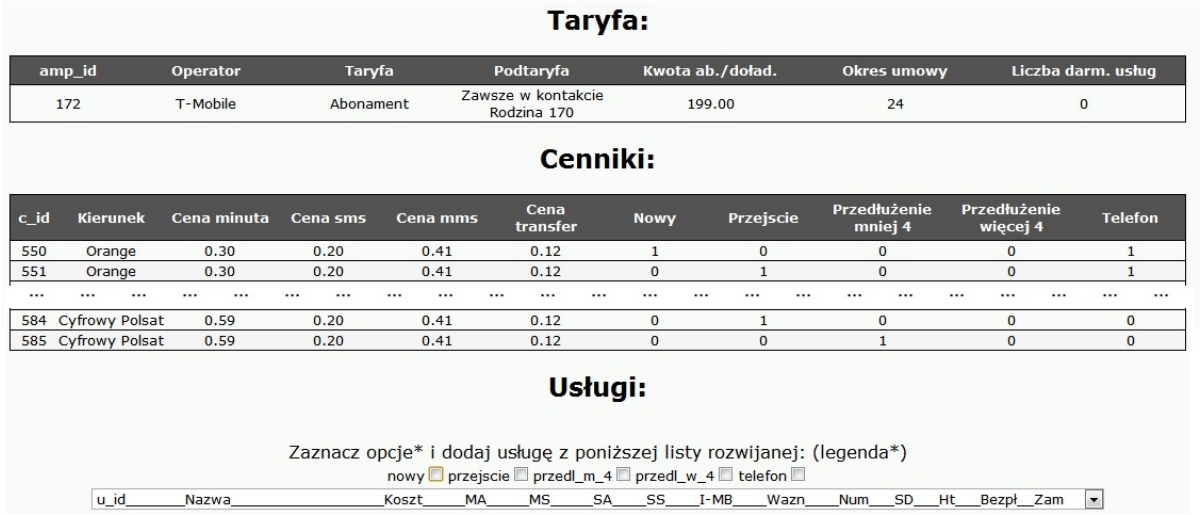
Tak jak we wszystkich tabelach na stronie panelu administracyjnego, tak i w tabeli *Taryfy* z rysunku 3.6 czynności, które można wykonać na danym wierszu znajdują się w kolumnie *Operacje*. Zawartość danej podstrony składa się z trzech części. Pierwsza z nich określa sieć operatora, którego taryfy mają zostać wyświetlone. Na drugą część składają się: wypisane taryfy z możliwością wykonania na nich zadanych operacji i pola o nazwie *edytuj* służące do tworzenia powiązań wskazanej taryfy z określonymi w późniejszych etapach cennikami i usługami. Trzecia część powala na dodawanie nowych taryf.

Jeżeli zostanie naciśnięty przycisk *edytuj* znajdujący się w wierszu z daną taryfą, nastąpi wtedy załadowanie podstrony o nazwie *Taryfa-Cennik* (rys. 3.7).



Rys. 3.7. Dodawanie cenników do jednej z taryf z rysunku 3.6

Na rysunku nr 3.7 pokazano taryfę o identyfikatorze 172 do której przypisany został zbiór cenników (550-585). W celu usunięcia któregoś z tych powiązań, należy kliknąć przycisk *usuń*. Można również dołączyć brakujące cenniki, służy do tego rozwijana lista wyboru, oraz znajdujące się bezpośrednio nad nim pole tekstowe. W chęci dodania usługi dla podzbioru cenników, trzeba nacisnąć odpowiednie pola wielokrotnego wyboru w kolumnie *edytuj wiele* lub nacisnąć przycisk *zaznacz*, który oznacza kolumny każdego z wierszy. Po wybraniu cenników, do których mają zostać dowiązane usługi, kliknąć należy przycisk *edytuj* znajdujący się obok przycisku *zaznacz*, w celu załadowania się podstrony o nazwie *taryfa_usługi*, która jest zaprezentowana na rysunku numer 3.8.



Rys. 3.8. Dodawanie usług do wybranych cenników

Nie widać na rysunku 3.8 żadnych usług powiązanych z daną taryfą i cennikami. Oznacza to, że dla całego zbioru zaznaczonych cenników nie została określona chociaż jedna wspólna usługa.

Przypisywanie usług do cenników odbywa się przez zaznaczenie właściwej kombinacji pól ze zbioru {*nowy, przejscie, przedl_m_4, przedl_w_4, telefon*} oraz wybór usługi z listy rozwijanej (opisane skróty znajdujące się w liście rozwijanej można sprawdzić po najechaniu kursorem myszy na słowo *legenda**). Kombinacja tych pól definiuje podzbiór cenników z listy wyświetlonych na podstronie, do których zostanie przypisana wybrana usługa. Jeżeli nie zaznaczono żadnej z opcji a wybrano usługę, to taka usługa zostanie przypisana do każdego z wyświetlonych cenników. Jeżeli planowane są operacje na mniejszym zbiorze cenników niż zbiór cenników o numerach 550-585, należy nacisnąć przycisk *Cofnij* i wybrać odpowiadający podzbiór.

3. Podstrony *Cennik* i *Usługi*

Żeby można było przypisywać cenniki i usługi do taryf należy je wcześniej zapisać w bazie danych, co można wykonać na podstronie zaprezentowanej na rysunkach 3.9 i 3.10.

Nr	Kierunek	Cena za minutę	Cena za SMS	Cena za MMS	Cena za transfer	Nowy	Przejście	Przedłuż. < 4	Przedłuż. > 4	Telefon	Operacje
1	Stacjonarne	0.49	1.01	0.00	0.00	1	1	0	0	0	mod usuń
2	Stacjonarne	0.49	1.01	0.00	0.00	1	1	0	0	1	mod usuń
3	Stacjonarne	0.49	1.01	0.00	0.00	0	0	1	0	1	mod usuń
4	Stacjonarne	0.49	1.01	0.00	0.00	0	0	1	0	0	mod usuń
...	...	...	...	...	...	...	...	...	...	...	...
856	Orange	0.29	0.09	0.19	0.12	1	0	0	0	0	mod usuń
857	Orange	0.29	0.09	0.19	0.12	0	1	1	0	0	mod usuń
858	Orange					0	0	0	0	0	dodaj

Rys. 3.9. Wpisywanie nowych cenników

Nr	Hier.	Nazwa usługi	Koszt	Min. do All	Min. w sieci	SMS do All	SMS w sieci	Inter. MB	Ważn.	Ile num.	Stan darm.	Hotsp.	Bezpł.	Zam.	Operacje
1	0	Minut do wszystkich	0.00	30	0	30	0	0	30	0	0	0	0	1	mod usuń
2	1	Minut do wszystkich	0.00	40	0	40	0	0	30	0	0	0	0	1	mod usuń
3	2	Minut do wszystkich	0.00	60	0	60	0	0	30	0	0	0	0	1	mod usuń
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69	68	Stan darmowy	0.00	0	10000	0	0	0	30	0	100	0	0	0	mod usuń
70	69	Pakiet Internet	0.00	0	0	0	0	250	30	0	0	0	0	0	mod usuń
71	70									0	0	0	0	0	dodaj

Rys. 3.10. Wpisywanie nowych usług

Przeglądanie ofert operatorów

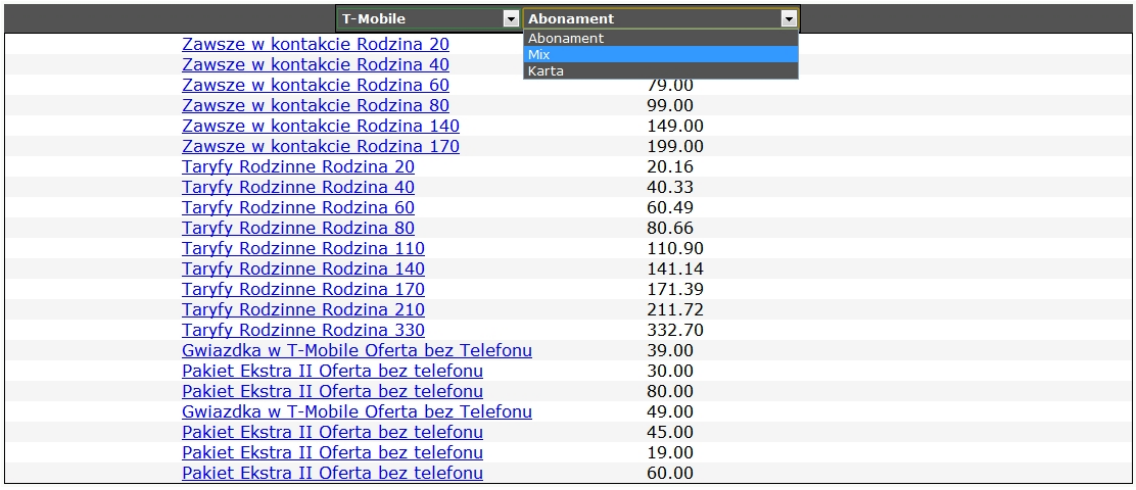
Jest to dodatkowa funkcjonalność systemu. Pozwala na przeglądanie stanu bazy danych przez niezalogowanego użytkownika zaprojektowanej strony. Jest to bardzo przydatne, ponieważ znalezienie potrzebnych cen na stronach operatorów nie jest prostym zadaniem.

Podstrona *Pokaż Oferty*

Na rysunku numer 3.11 zaprezentowano pierwsze okno, które widzi użytkownik na podstronie służącej wyświetlaniu ofert. W pierwszym etapie następuje wybór sieci operatora.

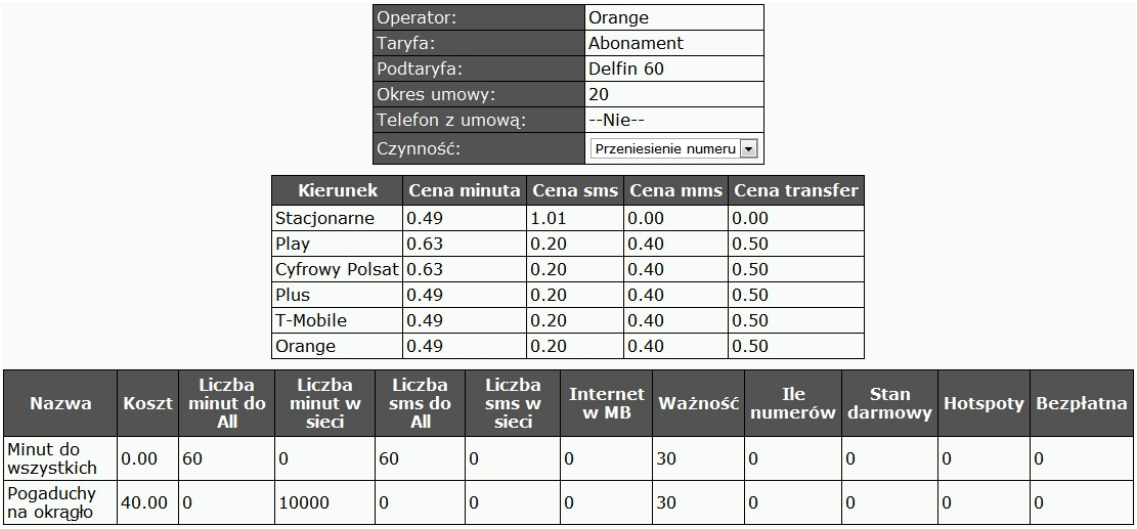
Rys. 3.11. Wybór sieci operatorskiej

W kolejnej fazie użytkownik musi dokonać wyboru interesującej go podtaryfy ze zbioru taryf {*Abonament, Mix, Karta*}. Na rysunku 3.12 został przedstawiony przykład wyboru taryf.



Rys. 3.12. Wybór taryfy i podtaryfy

Po wyborze taryfy należy wybrać właściwą opcję z listy rozwijanej przedstawiającej *Czynność* użytkownika, czyli wartość spośród: *przeniesienie numeru, nowy w sieci, przedłużenie umowy*. Po dokonaniu wyboru wyświetlają się tabele określające ceny podstawowych usług, takich jak połączenia, SMS-y oraz lista usług przypisanych do danej taryfy. Jeżeli koszt dla danej usługi wynosi zero (jak w przykładzie z rysunku 3.13), to oznacza, że dana usługa jest dołączona do danej taryfy bezpłatnie w ramach abonamentu.



Operator:	Orange
Taryfa:	Abonament
Podtaryfa:	Delfin 60
Okres umowy:	20
Telefon z umową:	--Nie--
Czynność:	Przeniesienie numeru

Kierunek	Cena minuta	Cena sms	Cena mms	Cena transfer
Stacjonarne	0.49	1.01	0.00	0.00
Play	0.63	0.20	0.40	0.50
Cyfrowy Polsat	0.63	0.20	0.40	0.50
Plus	0.49	0.20	0.40	0.50
T-Mobile	0.49	0.20	0.40	0.50
Orange	0.49	0.20	0.40	0.50

Nazwa	Koszt	Liczba minut do All	Liczba minut w sieci	Liczba sms do All	Liczba sms w sieci	Internet w MB	Ważność	Ile numerów	Stan darmowy	Hotspoty	Bezpłatna
Minut do wszystkich	0.00	60	0	60	0	0	30	0	0	0	0
Pogaduchy na okrągło	40.00	0	10000	0	0	0	30	0	0	0	0

Rys. 3.13. Informacja wyświetlana dla jednej z podtaryf

3.3. Aplikacja

W tym podrozdziale zostanie w sposób szczegółowy przedstawiona aplikacja na platformę Android (załączniku J). Opisywany projekt służy do wskazywania najkorzystniejszych ofert wśród operatorów, zatem potrzebujemy aplikacji wygodnej dla użytkownika.

Na początku jednak aplikacja powinna określić jaką taryfę oraz z jakich dodatkowych usług korzysta użytkownik. Do tego potrzeba kilku elementów, gdzie podstawowym jest stworzenie czterech tabel w bazie danych oraz pobranie z serwera przekonwertowanej bazy danych z PostgreSQL do SQLite'a. Wykorzystywanym narzędziem był analizator składni, który wraz z dostępem do historii połączeń udostępniał informacje o nazwie operatora dla danego połączenia wychodzącego. Najważniejszym fragmentem aplikacji jest algorytm optymalizujący, który zostanie opisany w podrozdziale 3.3.3. Ostatnią częścią jest zaś interfejs użytkownika, który powinien być prosty, czytelny i funkcjonalny.

Podsumowując aplikacja potrzebuje do poprawnego działania pliku będącego bazą danych w SQLite, połączenia http do pobrania tego pliku, analizatora składniowego sprawdzającego nazwę operatora, prostego w obsłudze interfejsu graficznego i zaimplementowanego algorytmu matematycznego optymalizującego oferty zależnie od predyspozycji użytkownika aplikacji.

3.3.1. Struktura aplikacji

Baza danych

W aplikacji wykorzystano bazę SQLite ze względu na jej szybkość oraz dostępny interfejs w systemie Android. Do konwersji bazy danych z PostegreSQL na SQLite został napisany skrypt, który opisano w podrozdziale 3.2.3.

Bazę danych w aplikacji mobilnej rozszerzono o dodatkowe tabele przedstawione poniżej (tabele 3.15-3.18). Kolorem ciemno-zielonym oznaczono kolumny pełniące rolę kluczy głównych poszczególnych tabel.

Tabela *Operatormiuty* jest niezbędna dla alternatywnego sposobu wykorzystania aplikacji, w której użytkownik sam wpisuje ilość minut wydzwonionych do konkretnego operatora. Taki przypadek nie wymaga połączenia z internetem w celu wykorzystania skryptu *Nakładki* opisywanego w podrozdziale 3.2.4.

Tab. 3.15. Operatorminuty – tabela przechowująca czas połączeń do wszystkich operatorów

Operatorminuty	
Kolumna	**Typ**
opmin_id	integer
ilosc_minut	text
Nazwa_operatora	text

Opmin_id – klucz główny tabeli *Operatorminuty*,

Ilosc_minut – liczba sekund wykorzystana dla danego operatora,

Nazwa_operatora– nazwa operatora,

Tabela *podtaryfaiuslugi* powstała ze względu na możliwość dodania najlepszej podtaryfy konkurencyjnej dla abonamentu użytkownika, która zawiera dodatkowe

usługi. Ta tabela głównie wykorzystywana jest w algorytmie optymalizacyjnym opisanym w podrozdziale 3.3.3.

Tab.3.16.podtaryfaiuslugi - zawiera załączone usługi dla danej podtaryfy

podtaryfaiuslugi	
Kolumna	**Typ**
ptiu_id	integer
u_id	integer
amp_id	integer

ptiu_id – klucz główny tabeli *podtaryfaiuslugi*;

u_id – identyfikator usługi;

amp_id – identyfikator danej podtaryfy.

Tabela *wyniki* przechowuje wyniki działania algorytmu optymalizacyjnego. Ta tabela głównie wykorzystywana jest do wpisania oraz wyświetlania wyników w odpowiednim okienku interfejsu graficznego.

Tab.3.17. wyniki - rezultat działania algorytmu optymalizacyjnego

wyniki	
Kolumna	**Typ**
w_id	integer
amp_id	integer
calk_koszt	real

w_id – klucz główny tabeli wyniki;

amp_id – identyfikator podtaryfy;

calk_koszt – całkowity usługi.

Tabela *telefony* przechowuje darmowe numery dla abonamentu użytkownika. Numery z tej listy nie będą uwzględniane w obliczeniach.

Tab. 3.18. telefony - lista darmowych numerów w wybranych w usługach

telefony	
Kolumna	**Typ**
t_id	integer
u_id	integer
telefon	text

t_id – klucz główny tabeli telefony;

u_id – identyfikator usługi;

telefon – numer telefonu dla danej usługi.

Struktura aplikacji

Intenty (aktywności) określają możliwość przejścia pomiędzy poszczególnymi oknami aplikacji. Struktura aplikacji podzielona jest w ten sposób, że dla każdego okna napisana jest osobna klasa opisująca daną instancję. Każda aktywność zawiera informacje opisujące zasadę działania przycisków, sposób przedstawienia listy taryf, usług, czy ekranu początkowego. Tabela 3.19 przedstawia sposób wywołania kolejnej nowej aktywności. W tym przypadku jest to wyświetlenie wyników bardziej szczegółowych za pomocą klasy *BazadanychWynikiSzczegoloweActivity.class*.

Tab. 3.19. *Aktywności*

```
Intent intent = new
Intent(BazadanychWynikiActivity.this,
BazadanychWynikiSzczegoloweActivity.class);
startActivity(intent);
```

W projekcie prawie wszystkie aktywności dziedziczą z klasy BazadanychActivity (przykład przedstawiony poniżej paragrafu), gdzie zapisane są funkcje pomocne dla obsługi bazy danych oraz inne stałe wykorzystywane w późniejszych aktywnościach.

```
public class BazadanychListPodTaryfyActivity extends BazadanychActivity
```

XML

Jak zostało opisane w podrozdziale 2.3.2, platforma Android korzysta z języka XML do przechowywania układu graficznego, informacji o obsługiwanych aktywnościach oraz zezwoleń dla użytkownika.

Do poprawnego działania aplikacji potrzebne jest zapisanie każdej aktywności (*intentu*) w pliku *AndroidManifest.xml*. Poniżej został podany jeden przykład.

```
<activity android:name=".BazadanychActivity"></activity>
```

Ze względu na konieczność dostępu niektórych funkcji systemu Android należy odblokować je w manifeście. Poniżej znajdują się funkcje wykorzystywane w projekcie wraz z przykładem ich wprowadzenia do aplikacji.

- łączność z internetem (sprawdzenie operatora dla numerów telefonów osób, do których zostało wykonane połączenie);

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

- czytanie kontaktów (wyszukiwanie numerów osób, do których wykonano połączenie);

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

- przechowywanie informacji na zewnętrznym nośniku (baza danych);

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

- sprawdzenie czy urządzenie jest podłączone do internetu (w momencie pobierania danych o numerach wychodzących z telefonu i sprawdzeniu nazwy operatora tych numerów);

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

- utrzymywanie telefonu w stanie aktywnym (podczas czekania na skończenie obliczania optymalnej taryfy dla użytkownika).

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

Prawie dla każdej aktywności istnieje potrzeba napisania nowego układu graficznego, który odpowiadałby wizualnemu odwzorowaniu w interfejsie graficznym. W Androidzie występują rożne typy układów interfejsu: relatywne, liniowe, tabelaryczne i ramkowe. W projekcie głównie wykorzystywane były linowe a w nich zawarte relatywne.

Tab. 3.20. Zawartość pliku *main.xml* opisująca ekran wejściowy aplikacji

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
          android:background="#fff"
    android:layout_height="fill_parent" android:weightSum="1">
<TextView
android:background="@drawable/tlo"
android:gravity="center"  android:textSize="11pt"  android:text="J-
aTom"  android:layout_height="wrap_content"  android:textColor="#fff"
android:layout_gravity="center"
android:layout_width="fill_parent"></TextView>
<TextView  android:id="@+id/TextView01"  android:background="#535353"
android:textColor="#fff"        android:layout_height="wrap_content"
android:layout_width="fill_parent"            android:textSize="7pt"
android:text="Witamy w naszejaplikacji."></TextView>

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
    <TextView   android:id="@+id/wstep"   android:background="#ffff"
android:textColor="#000"         android:layout_width="wrap_content"
android:textSize="7pt"          android:layout_height="wrap_content"
android:text="@string/Wstep"    android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
android:layout_marginLeft="28dp"   android:layout_marginRight="28dp"
android:layout_marginTop="28dp"></TextView>
    <TextView   android:id="@+id/wstep2"   android:background="#ffff"
android:textColor="#000"         android:layout_width="wrap_content"
android:textSize="7pt"          android:layout_height="wrap_content"
android:text="@string/Pozdrowienia"
```

```
android:layout_below="@+id/wstep"
android:layout_alignParentRight="true"
android:layout_marginTop="34dp"></TextView>
    <Button                  android:background="@drawable/my_button"
android:id="@+id/Select"                  android:text="Przejdźdalej"
android:layout_width="fill_parent"     android:onClick="selfDestruct"
android:layout_height="50px"                 android:textColor="#fff"
android:layout_below="@+id/wstep2"
android:layout_alignParentLeft="true"
android:layout_marginTop="34dp"></Button>
</RelativeLayout>
</LinearLayout>
```

W tabeli 3.20 występują zagnieżdżenia, wprowadzone przyciski, a także napisy wyświetlane na ekranie aplikacji. Widoczne jest tu również wykorzystanie wartości ciągów znaków przechowywanych w pliku *values/string.xml*.

Poprzez zastosowane rozwiązania:

```
android:text="@string/Pozdrowienia"
```

nie zajmuje się dużo miejsca oraz tekst kodu jest bardziej czytelny, ponieważ cały tekst nie musi znajdować się w pliku układu graficznego (np. w pliku *main.xml*).

Preferencje

Do zapisu preferencji użytkownika, czyli wykorzystywanej przez niego taryfy wraz z usługami, posłużono się funkcją *SharedPreferences* (z ang. *preferencje dzielone*). Na początku prawie każdej aktywności czyszczone są preferencje aktualnie ustawiane w aktywności i wczytywane są inne, które będą wyświetlane na ekranie aplikacji.

Adapter

Do wyświetlenia listy z bazy danych została wykorzystana klasa *SimpleCursorAdapter* (tabela 3.21), która używa kursora opisanego w podrozdziale 3.3.2. Obiekt klasy *android.database.Cursor* zwany potocznie kursorem za pomocą zapytania SQL wyciąga odpowiednie wiersze z bazy danych SQLite. Następnie przy wykorzystaniu klasy*SimpleCursorAdapter* należy utworzyć obiekt klasy *ListAdaptera*, a później wyświetlić go za pomocą obiektu klasy *ListView* na ekranie interfejsu użytkownika.

Tab. 3.21. Przykład wykorzystania klasy *SimpleCursorAdapter*

```
  mCursor = mDB.rawQuery( "Select amp_id _id, taryfa from ab_mix_pp
  WHERE o_id LIKE ? GROUP BY taryfa", querystring);
  startManagingCursor(mCursor);
  ListAdapter adapter =
              new SimpleCursorAdapter(
                  this,
                  R.layout.database_item,
                  mCursor,
                  new String[] {"taryfa"},
              newint[] {R.id.text2 });
  ListViewav = (ListView)findViewById(R.id.databaseList);

av.setAdapter(adapter);
```

3.3.2. Biblioteki

W tym podrozdziale wyjaśniono wybór niektórych bibliotek wykorzystywanych w projektowanej aplikacji.

- **Kursor i baza danych SQLite**

Najczęściej wykorzystywaną była biblioteka SQLite. Dzięki niej zaprojektowano nowe tabele opisywane w podrozdziale 3.3.1. Kursor to jedna z najbardziej potrzebnych bibliotek, przy jej wykorzystaniu uzyskano dostęp do całej bazy danych. Za jej pomocą możliwe było odwołanie się do odpowiednich wierszy w zadanym zapytaniu SQL. W tabelach 3.22 i 3.23 przedstawiono przykład wykorzystania dwóch powyższych klas.

Tab. 3.22. Tworzenie tabeli za pomocą biblioteki *SQLiteDatabase*

```
mDB.execSQL("CREATE TABLE IF NOT EXISTS telefony(" +
          "t_id INTEGER PRIMARY KEY AUTOINCREMENT," +

          "u_id INTEGER," +

          "telefon TEXT);");
```

W tabeli 3.22 pokazano utworzenie tabeli *telefony*, która została opisana w rozdziale 3.3.1.

Tab. 3.23. Zapytanie SELECT za pomocą kursora

```
mCursor =  mDB.rawQuery( "Select o_id, taryfa from ab_mix_pp WHERE
amp_id LIKE ?", new String[] {abonamentid});
    mCursor.moveToFirst();
    String[] oiditaryfa = new String[2];
    if(mCursor.getCount() != 0 ){
          while(!mCursor.isAfterLast()){
                oiditaryfa[0] = mCursor.getString(0);
                oiditaryfa[1] = mCursor.getString(1);
                mCursor.moveToNext();
          }
    }
mCursor.close();
```

W tabeli 3.23 zostało zaprezentowane zapytanie SQL, które zwraca *o_id* (identyfikator operatora) oraz nazwę taryfy z tabeli *t_id,* ustawiając następnie kursor na początku nieuporządkowanej listy metodą *moveToFirst()*. Używając pętli *while* nastąpi przejście po wszystkich wierszach znajdujących się w kursorze.

CallLog

Kursor stosowany był również w klasie *android.provider.CallLog*. Za jej pomocą uzyskany został dostęp do historii połączeń. Dane otrzymano z URI (Uniform Resource Identifier) *content://call_log/call* poprzez odpowiednie zapytanie SQL. Do tego celu utworzono tablicę przedstawioną w tabeli 3.24.

Tab. 3.24. Pomocnicza tablica do pozyskiwania informacji z URI *content://call_log/calls*

```
    protected String[] requestColumns = {
            CallLog.Calls.NUMBER,

            CallLog.Calls.TYPE,

            CallLog.Calls.DURATION,

            CallLog.Calls.DATE

          };
```

Do pobrania danych z bazy danych należy określić informacje, które powinny zostać pobrane z podanego URI (*content://call_log/call)*. Dla opisywanej aplikacji potrzebny jest czas trwania, numer i określony rodzaju połączenia (wychodzące). Taki zestaw stałych przedstawiono w tabeli 3.24, który następnie wykorzystano w późniejszej części aplikacji.

Tab. 3.25. Typ połączenia

```
int type = Integer.parseInt(calls.getString(calls.getColumnIndex(CallLog.Calls.TYPE)));
```

Zmienna *type* przedstawiona w tabeli 3.25 przechowuje wartość określającą rodzaj połączenia. Jeżeli wartość ta równa się jeden to ten typ wiadomości był przychodzący, dwa to wychodzący, zaś trzy określa połączenia pominięte.

Chcąc wziąć pod uwagę okres rozliczeniowy zastosowane zostało odejmowanie od obecnej daty wartość 30 dni (przeliczonych na sekundy). Następnie otrzymaną wartość porównano z zawartością *CallLog.Calls.DATE* dla każdej wykonanej rozmowy. Jeżeli wartość okazała się mniejsza to zostało odrzucane znalezione połączenia, a uzyskane dane nie zostały zapisywane.

Bibliotekę *android.util.Log* użyto podczas testów do debugowania programu. Ułatwiało to rozpoznanie niepoprawnie wprowadzonych parametrów oraz sprawdzenie czy program działa poprawnie. W tabeli 3.26 zaprezentowano przykładowe wykorzystanie logowania.

Tab. 3.26. Logi

```
Log.d(Debug_TAG, "Czas trwania: " + Czas);
Log.d(Debug_TAG, "Numer: " + Numerek);
```

Pod stałą *Debug_TAG* określony był identyfikator, poprzez który można było rozróżnić wpis w debugerze. Zmienne Czas i Numerek wskazują informacje pobrane z biblioteki CallLog, które z łatwością można było przeglądać.

Analizator składni

Wyszukiwanie nazwy operatora przypisanego dla danego numeru wychodzącego zostało wykonane poprzez przesłanie zapytania z wszystkimi numerami telefonów wychodzących do nakładki opisanej w rozdziale 3.2.4 za pomocą *http request*. Następnie analiza składni strony została wykonana za pomocą biblioteki *Jsoup* otrzymano nazwy operatorów. Przykład przedstawiono w tabeli 3.27.

Tab. 3.27. Wykorzystanie analizatora składni

```
Elements newsHeadlines = doc.select("p");
for (Element div : newsHeadlines){

    Log.d(Debug_TAG, "Co jest w tablicy?Numer: " + div.text());

    Siec[i][2] = div.text(); // nazwa sieci użytkownika

    i++;

}
```

Napisana strona nakładki zwracała wartości operatorów w kolejnych znacznikach <*p*> języka HTML. Zatem wystarczyło z niej pobrać zawartość między znacznikami, a następnie w takiej kolejności zapisać do tabeli odpowiednio zwrócone nazwy.

Java

W programie głównie wykorzystywane były różne biblioteki *Javy (java.io.*, java.net.**) za pomocą której tworzyło się klasy, zmienne statyczne oraz różne metody tj. *FileReader*. Z reguły utworzone metody służyły do zmniejszenia ilości kodu oraz łatwiejszego zrozumienia całości procesów obliczeniowych.

3.3.3. Algorytm Optymalizacji

Poniższy podrozdział poświęcono opisowi algorytmu optymalizacyjnemu, jego danym wejściowym, sposobie działania oraz osiągniętemu rezultatowi.

Dane wejściowe algorytmu

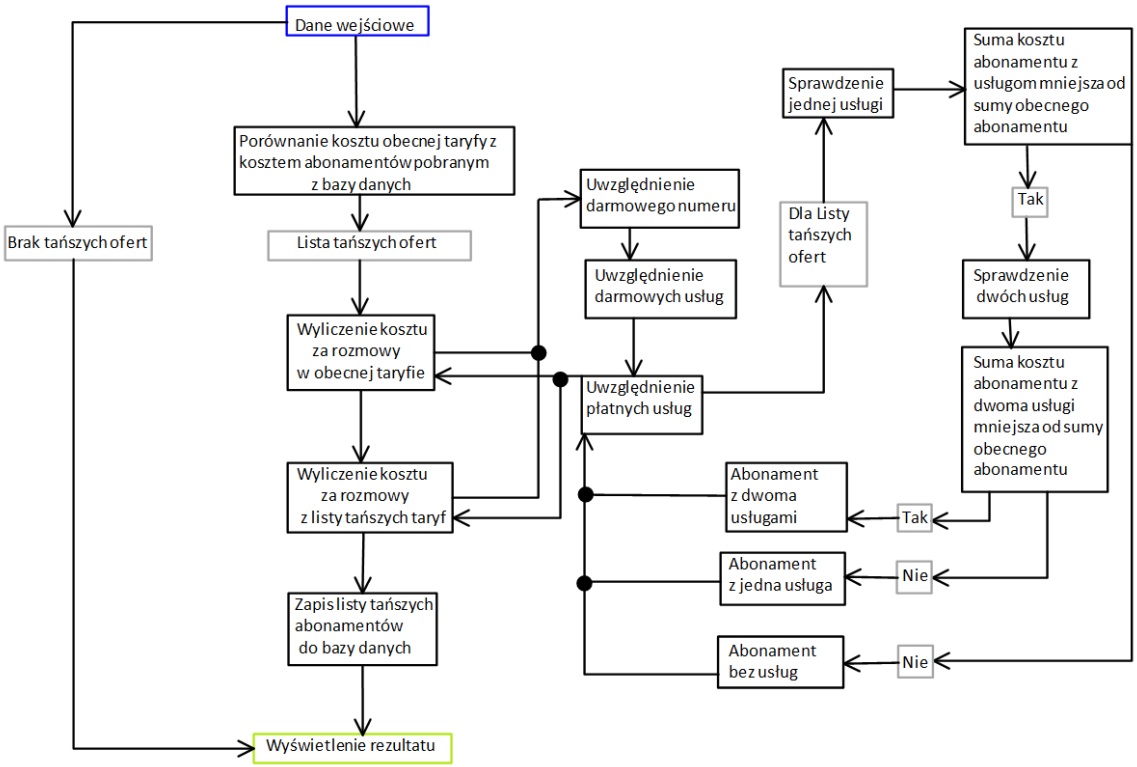
Dane wejściowe można wprowadzić na dwa sposoby opisane poniżej:

1. Użytkownik sam wprowadza informacje o czasie rozmowy z danym operatorem.
2. Informacje pobierane są z telefonu
    - sprawdzane są numery rozmów wychodzących za ostatni miesiąc i zapisywanego do odpowiedniej tabeli za pomocą metody *CountConnected*,
    - numery są przesyłane na stronę nakładki opisanej w podrozdziale 3.2.4 , przetworzony metodą *CheckProviders* rezultat w celu otrzymania nazwy operatora,

- czas trwania rozmowy dla danego operatora jest sumowany za pomocą metody *SummaryProviders*.

Istotne dane wejściowe to taryfa użytkownika oraz usługi z jakich korzysta.

Schemat działania algorytmu



Rys. 3.14. Schemat działania algorytmu optymalizacyjnego

Opis słowny rysunku 3.14:

1. Po otrzymaniu danych wejściowych obliczona zostaje suma kosztu za obecną taryfę oraz za usługi dodatkowe.
2. Kolejno eliminowane są oferty o abonamencie droższym od wyliczonego kosztu w punkcie 1.
3. Wyliczony zostaje koszt uwzględniający informacje o czasie rozmowy wychodzącej do danego operatorem, korzystając z cenników za pomocą metody *wyliczeniekosztu* oraz *wyliczenierozmowyzagraniczne*.
    a. uwzględnione zostają usługi darmowe, gdzie za pomocą metody *odejmowanieczasdarmowe* wydzwonione minuty odejmowane są od darmowych. Dla wartości obecnej taryfy:

- jeżeli istnieją minuty darmowe dla obecnej taryfy to odjęte zostają one tylko od czasu przypisanego dla naszego operatora,
- jeżeli są minuty darmowe dla całej sieci to najpierw zostaje sprawdzone do jakiego operatora wydzwoniono największą liczbę minut, a następnie od niego zostają odjęte te minuty. Jeśli okaże się, że jeszcze zostało trochę darmowych minut to zabierane zostają kolejnemu operatorowi.

b. Wyliczana jest również różnica dla usług płatnych, ponieważ jest w obecnym abonamencie.
c. Uwzględniane są minuty dla usługi *darmowy numer*, który został wcześniej wybrany w ofercie.

4. Ponownie wykonują się te same metody, tym razem dla abonamentów, które nie zostały odrzucone w drugim punkcie algorytmu.
   a. Dodatkowo przy usługach płatnych brane są pod uwagę maksymalnie dwie dodatkowe usługi i później razem są uwzględniane.
      - Na początku zostaje wzięta pod uwagę jedna usługa i sprawdzana czy jej koszt wraz z kolejną usługą jest większy od obecnego abonamentu. Jeżeli jest to usługi są odrzucane, a jeśli nie to są brane pod uwagę.
      - Kolejnym etapem jest uwzględnienie drugiej usługi wraz z pierwszą. Tutaj znowu sprawdzane jest czy suma kosztu wraz z usługami jest większa od naszego abonamentu z usługą. Następnie postępowanie, jak w poprzednim podpunkcie.
      - Jeżeli okaże się, że nie zostały znalezione dwie usługi to brana pod uwagę jest tylko jedna usługa a w momencie gdy nawet ona nie zostanie znaleziona, to brana jest pod uwagę jedynie sama taryfa.
5. Zapisywane są pozostałe abonamenty, układane malejąco oraz wyświetlane ogólne informacje na ekranie interfejsu. Można mieć dostęp do większej szczegółowości wyników naciskając odpowiedni przycisk.

Rezultatem powyższego algorytmu powinna być lista ofert abonamentowych. Uwzględniająca taryfy i odpowiednie darmowe oraz płatne usługi, które łącznie minimalizują koszt utrzymania telefonu.

3.3.4. Interfejs użytkownika

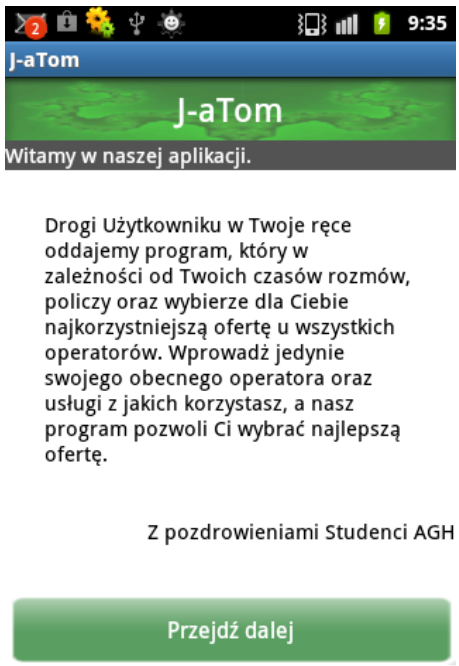
W tym podrozdziale zostanie opisany interfejs użytkownika, jego działanie oraz wygląd.

Najistotniejsze cechy zaprojektowanego interfejsu graficznego:

- skorzystanie z mechanizmu aktywności, w celu przechodzenia pomiędzy panelami interfejsu oraz zwiększenia czytelności dla użytkownika,
- użycie *XML-a* do opisu interfejsu użytkownika oraz elementów, takich jak lista taryf, podtaryf czy usług,
- wykonanie przejrzystej grafiki do przycisków oraz tła nagłówka.

Poniżej przedstawiono przykładowe ekrany aplikacji wraz z opisem funkcjonalności.

Ekran początkowy (rysunek 3.15) informuje użytkownika o podstawowej funkcjonalności aplikacji. Po naciśnięciu przycisku *Przejdź dalej* istnieją możliwości przejścia do wyboru ofert lub ekranu głównego.



Rys. 3.15. Ekran początkowy

Ekran wybór taryfy służy do specyfikacji posiadanej taryfy. Przykładową opcję zaprezentowano na rysunkach 3.16 a) – f).


15:48
J-aTom
Jak długo w sieci:
Zaznacz jeśli uzyskasz telefon:
Telefon
Nowy
Przedluzenie
Przejscie

(a)


15:49
J-aTom
Jak długo w sieci:
Tu pojawią się twoje preferencje
Podaj jak długo jesteś w sieci w miesiacach
Wybierz wartosc
Dalej

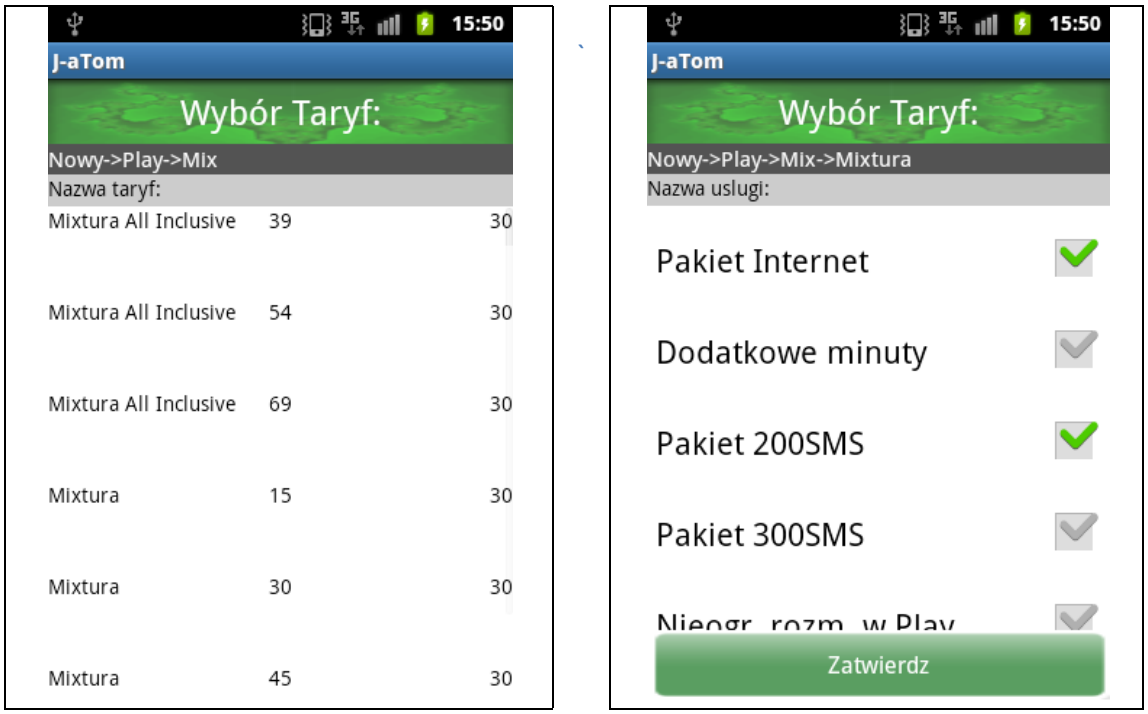
(b)


15:49
J-aTom
Wybór Taryf:
Nowy
Nazwa taryf:
Orange
Play
Plus
T-Mobile
Stacjonarne
Cyfrowy Polsat

(c)


15:49
J-aTom
Wybór Taryf:
Nowy->Play
Nazwa taryf:
Abonament
Karta
Mix

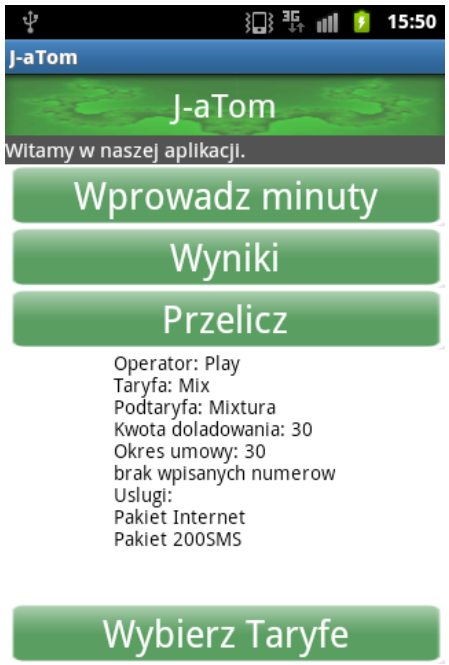
(d)



(e) (f)

Rys. 3.16. a) Wybór czasu w przebywania w sieci oraz zaznaczenie opcji z telefonem, b) Określenie czasu abonamentu, c) Wybór operatora, d) Wybór rodzaju taryfy, e) Wybór podtaryfy, f) Wybór usług dodatkowych

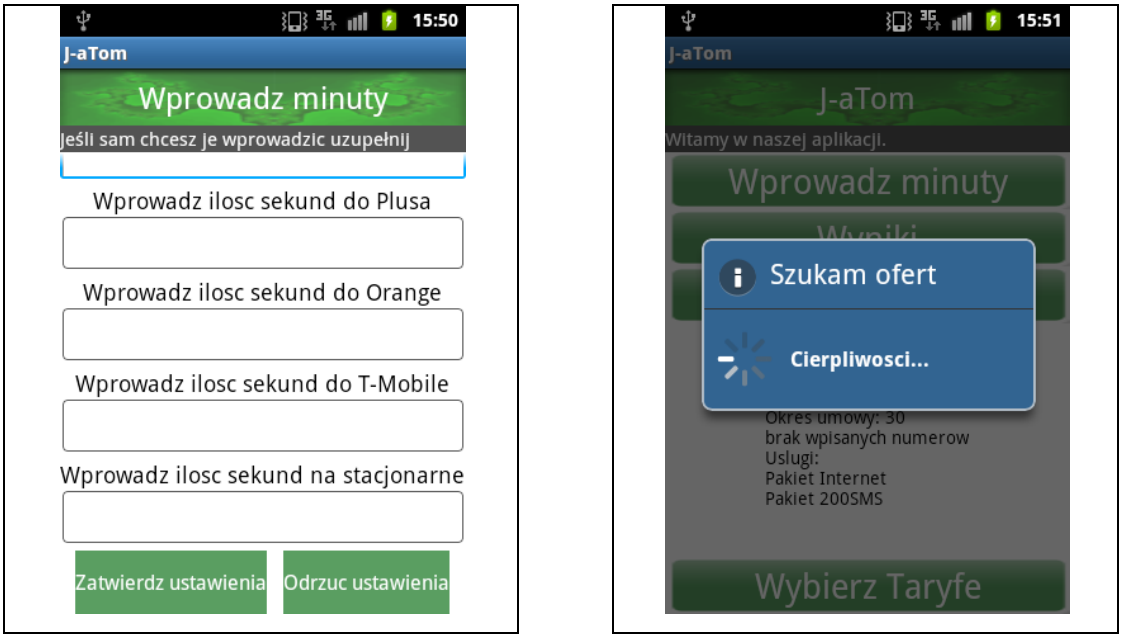
Ekran główny zaprezentowany na rysunku 3.17 to miejsce, w którym można wprowadzić czas trwania połączeń do poszczególnych operatorów, przeprowadzić obliczenia oraz sprawdzić wyniki, jeżeli obliczenia zostały wykonane.



Rys. 3.17. Ekran główny

Na rysunku 3.18 a) zaprezentowane jest okno aplikacji, w którym można samemu wpisać odpowiednie liczby sekund wydzwonione do różnych operatorów. Ta opcja jest przydatna, ponieważ pozwala na skorzystanie z aplikacji osobom, które nie korzystają na co dzień z telefonu z platformą Android, a mają możliwość skorzystania z aplikacji na telefonie innej osoby.

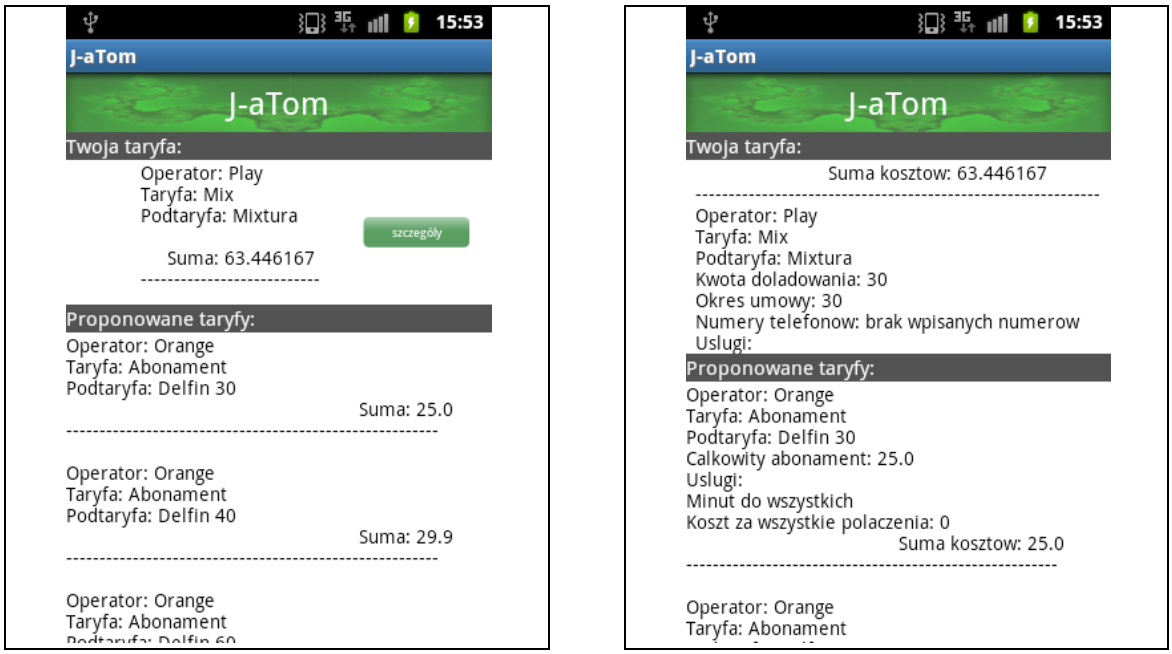
Po wybraniu opcji przeliczenia pojawia się okno informujące o trwających obliczeniach (rys. 3.18. b)).



(a) (b)

Rys. 3.18. a) Ekran wprowadzenia minut, b) Przeliczanie ofert

- Na rysunku 3.19. a) przedstawiono listę wszystkich ofert, jakie zostały zaproponowane przez algorytm obliczający.
- Na rysunku 3.19. b) przedstawiono szczegółowy opis ofert wraz z dodatkowymi usługami, które są używane w danej podtaryfie.



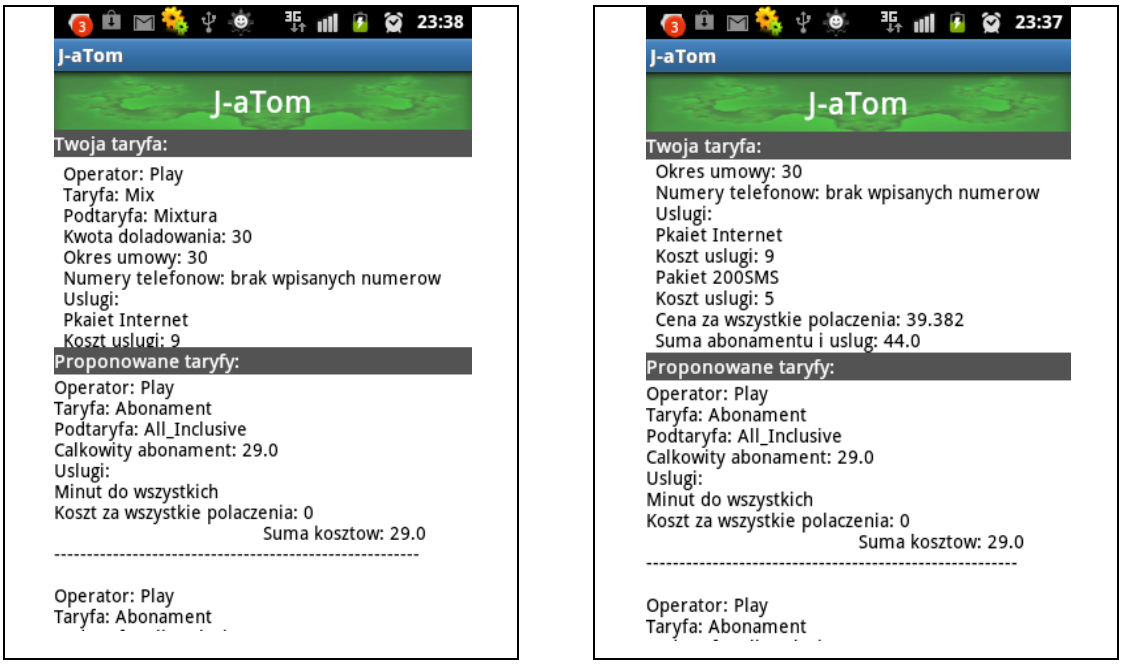
Rys. 3.19. a) Wynik ogólny, b) Wynik szczegółowy

Podsumowując, interfejs użytkownika został zaprojektowany tak, by używanie aplikacji nie sprawiało użytkownikom problemów.

3.3.5. Analiza wyników

Przeprowadzone zostało badanie mające na celu sprawdzenie, czy napisany program poprawnie pobiera dane z telefonu oraz prawidłowo optymalizuje taryfy. Wzięty został pod uwagę biling w formie elektronicznej, z którego za pomocą skryptów *dodawanie_minut_cz1.php*, *dodawanie_minut_cz2.php* oraz *gNN.php,* wyciągnięte zostały potrzebne informacje przedstawiające zsumowane długości połączeń w sekundach wydzwonione do poszczególnych operatorów. Następnie wartości te zostały przemnożone przez obecne stawki w taryfie. Biling jak i wyniki obliczeń znajdują się w załączniku H. Następnym krokiem było porównanie wyników z obliczeniami aplikacji mobilnej, która zsumowała długości połączeń za pomocą metody *CountConnected().* W dalszej kolejności w algorytmie optymalizacyjnym odpowiednie wartości czasów do operatorów zostały przemnożone przez ceny dla obecnej taryfy. Wyniki tych obliczeń zostały zaprezentowane w aplikacji. Rysunek 3.20. a) przedstawia wybraną taryfę, natomiast na rysunku 3.20. b) można zobaczyć, iż obliczona cena za wszystkie połączenia do wszystkich operatorów wynosi 39.38 zł. Widoczna jest również proponowana taryfa, która okazała się zdecydowanie

korzystniejsza od obecnej, ze względu na darmowe minuty wliczone w abonament. Koszt za wszystkie połączenia w otrzymanej taryfie wyniósł zero złotych, ponieważ korzystaliśmy z darmowych usług w abonamencie. Zatem porównując łączną kwotę za abonament jaką musielibyśmy zapłacić dla obecnej taryfy 44 zł do proponowanego kosztu 29zł, możemy wnioskować, że algorytm wybrał dla nas korzystniejszą ofertę.



(a) (b)

Rys. 3.20. a) Ekran z taryfą, b) Szczegóły wyliczeń

3.4. Perspektywy rozwoju

3.4.1. Serwis WWW

Planowane jest rozszerzenie dostępu do panelu administracyjnego również na inne państwa. W tym celu potrzebne jest przyspieszenie procesu uzupełniania bazy właściwymi ofertami sieci operatorów oraz pozwolenie na uzupełnianie baz ofert także zagranicznym użytkownikom. Wiąże się to z przetłumaczeniem strony na inne języki jak i z utworzeniem zbioru pewnych praw, definiujących poziomy dostępu do bazy danych. Drugim etapem rozwoju jest zaimplementowanie w ramach serwisu WWW algorytmu optymalizującego. Działałby na takiej samej zasadzie co algorytm

w telefonie, jednak dane użytkownika brane byłyby z wprowadzonych przez użytkownika fragmentów bilingów lub z wprowadzonych sum długości połączeń wykonanych do każdej ze wskazanych sieci operatorskich.

3.4.2. Aplikacja mobilna

Zwiększenie funkcjonalności aplikacji możliwe jest w głównej mierze przez dołożenie do obecnej wersji bloku metod pozwalających na odczyt liczby SMS-ów. Obecnie nie udało się tego dokonać z powodu braku odpowiedniego API w systemie Android. Odczytanie liczby, długości oraz odbiorcy każdej wiadomości, zwiększyłoby poprawność uzyskiwanych wyników optymalizacji. Drugim czynnikiem wpływającym na wyniki działania aplikacji jest zróżnicowana charakterystyka korzystania z określonych usług w zależności od dnia czy miesiąca. W trakcie urlopu liczba wydzwonionych minut zwykle znacznie się różni od liczby minut wydzwonionych w okresie pracy, tym samym rachunki telefoniczne za dwa różne miesiące mogą być całkiem do siebie niepodobne. Dlatego planowane jest umożliwienie operowania naszej aplikacji na bilingach połączeń z kilku lub nawet kilkunastu miesięcy wstecz, aby ocena predyspozycji telefonicznych użytkownika była bardziej wiarygodna. Kolejnym etapem rozwoju jest obsługa wszystkich błędów mogących powstać w wyniku użytkowania z aplikacji. Żeby użytkownik wiedział, jak długo musi czekać na zakończenie procesu optymalizacji, zostanie wprowadzone poprawione okienko dialogowe. Będzie ono informować na bieżąco o postępie analizowania bazy danych, która już została przebadana w celu znalezienia najkorzystniejszej ofert dla użytkownika. Ostatnim z elementów potencjalnego rozwoju aplikacji jest wprowadzenie szyfrowanej wymiany informacji pomiędzy aplikacją a nakładką.

Podsumowanie

Rynek rozwiązań technicznych wymaga dużej funkcjonalności od nowych programów. Jeżeli konkretny produkt ma przynieść korzyści, musi działać sprawnie, być innowacyjny i zarazem prosty w obsłudze. Takie właśnie cele postawione zostały przy projektowaniu aplikacji.

Projektowana aplikacja miała na celu wyliczać najtańszą taryfę wraz z usługami dopasowanymi do potrzeb użytkownika telefonii komórkowej. Dodatkowo wymagane było utworzenie bazy danych zawierającej oferty operatorów oraz strony internetowej służącej do zarządzania zbiorem tych informacji.

W wyniku prac postawione cele zostały spełnione. Zaprojektowana aplikacja w przedstawionym przykładzie wyliczyła nowy dopasowany abonament o 2/3 obecnych kosztów utrzymania telefonu. Tak więc została wskazana korzystniejsza taryfa dla użytkownika. Dodatkowo powstała strona internetowa, na której użytkownik może przeglądać taryfy i cenniki operatorów komórkowych.

Zaprojektowana aplikacja na platformę Android oraz serwis WWW są unikatowymi rozwiązaniami. Wszystkie operacje związane z przetwarzaniem informacji na temat wykonanych połączeń oraz wyborem najkorzystniejszej taryfy, wykonywane są lokalnie na platformie Android znajdującej się w telefonie danego użytkownika. W ten sposób dane wrażliwe nie są wysyłane poza telefon użytkownika.

Rozwój aplikacji związany jest z wyliczaniem bardziej optymalnego abonamentu, jak i ze zwiększaniem funkcjonalności. Na pierwszy czynnik składa się dopisanie kodu w aplikacji mobilnej odpowiedzialnego za odczyt liczby SMS. Funkcjonalność zaś może zostać zwiększona poprzez rozszerzenie produktu na inne kraje oraz dodanie do strony WWW interfejsu do przeliczania bilingów użytkowników. Dzięki takiemu zabiegowi, nie będzie wymagane posiadanie przez użytkownika telefonu z systemem operacyjnym Android w chęci skorzystania z algorytmu optymalizacji.

Bibliografia

[1] Główny Urząd Statystyczny, Społeczeństwo informacyjne w Polsce 2006-2010, 2010, http://www.stat.gov.pl/cps/rde/xbcr/gus/PUBL_nts_spolecz_inform_w_polsce_2006-2010.pdf.

[2] Oracle Corporation, MySQL 5.0 Reference Manual, 2012, http://dev.mysql.com/doc/refman/5.0/en/index.html.

[3] The PostgreSQL Global Development Group, PostgreSQL 8.1.11 Documentation, 2011, http://www.postgresql.org/files/documentation/pdf/8.1/postgresql-8.1-A4.pdf.

[4] The PostgreSQL Global Development Group, PostgreSQL 7.3.2 Tutorial, 1996–2002, http://www.postgresql.org/files/documentation/pdf/7.3/tutorial-7.3.2-US.pdf.

[5] SDK – Android Documentation, http://developer.android.com/sdk/index.html.

[6] Gartner, Sales of Mobile Devices in Third Quarter of 2011, 2011, http://www.gartner.com/it/page.jsp?id=1848514.

[7] NDK – Native Development Kit, http://developer.android.com/sdk/ndk/index.html.

[8] S. Conder, L. Darcey, Android. Programowanie aplikacji na urządzenia przenośne. Wydanie II, 2011.

[9] A. Hoog, Android Forencis Investigation, Analysis and Mobile Security for Google Android, 2011.

[10] S. Hashimi, S. Komatineni, D. MacLean, Android 2: Tworzenie aplikacji, 2010.

[11] PHP manual, http://php.net/manual.

[12] S. Friedl, SQL Injection Attacks by Example, 2007, http://www.unixwiz.net/techtips/sql-injection.html.

[13] P. Moulding, PHP Czarna księga, 2002.