POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Systemy informatyczne w automatyce (ASI)

PROJEKT INŻYNIERSKI

System monitoringu lokalizacji przesyłek kurierskich

English title

AUTOR:
Monika Strachowska

PROWADZĄCY PROJEKT:
Prof dr hab. inz. Czesław Smutnicki,
Katedra ...

OCENA PROJEKTU:

WROCŁAW 2014

Spis treści

Rozdział 1

Wstęp i cel pracy

Współcześnie coraz więcej osób korzysta z możliwości zakupów przez internet, co niesie to za sobą wiele korzyści. Często zakupiony towar jest tańszy, unikalny, bądź niedostępny w stacjonarnym sklepie czy po prostu jest to wygodniejsza forma zakupów. Konieczność dostarczenia przedmiotów nabytych w ten sposób skutkuje szybszym rozwojem usług w zakresie transportu. Oprócz wyżej wymienionych zakupów istotną rolę w branży transportowej pełnią dokumenty, które nierzadko muszą być dostarczone jak najszybciej. Z tych powodów ludzie zamawiający usługi kurierskie chcieliby otrzymać usługę o jak najwyższej jakości. Zwiększenie jakości tej usługi może nastąpić poprzez skrócenie czasu dostarczenia przesyłki (co fizycznie już jest bardzo trudne), niższy jej koszt, czy na przykład możliwość sprawdzenia, w jakim dokładnie miejscu ona się znajduje. Dokładna lokalizacja przesyłki – taka funkcjonalność usługi kurierskiej nie należy do jakości wymaganej i koniecznej, ale znacznie podniesie prestiż firmy kurierskiej, która zdecyduje się na takie dodatkowe udogodnienie. Klient może czuć się bardziej komfortowo wiedząc gdzie jest przesyłka, dzięki czemu może zaplanować sobie dzień w jakim nastąpi dostawa.

Przedstawiany tu projekt rozwija aktualną funkcjonalność firm kurierskich o graficzne przedstawienie (w formie mapy) aktualnej lokalizacji przesyłki poprzez lokalizowanie kuriera, który aktualnie w swoim samochodzie ją posiada. Taka forma prezentacji jest prosta w odbiorze i bardziej czytelna niż wyniki jakie prezentowane są aktualnie w formie tabel, w których zawarte są miejsca zeskanowania przesyłki. Ponadto praca próbuje rozwiązać problem jaki istnieje w estymacji czasu dostarczaniu przesyłki do adresata; obecnie estymacja czasu dostarczenia jest bardzo niedokładna (ogólna) lub w ogóle jej nie ma.

??

W została zrealizowana aplikacja w systemie Android, która wysyła na serwer jego lokalizację, odczytaną z czujników GPS. Odczyt ten jest przetwarzany przez serwer i zapisywany do bazy danych. Zaprojektowany serwer obsługuje klientów, którzy mają możliwość sprawdzenia aktualnej pozycji przesyłki a także dodanie nowego zlecenia przesyłki.

Komentarz: Projekt został zrealizowany z wykorzystaniem takich technologi jak: język programowania Java, system operacyjny Android, baza danych MySQL, Google Apis. Firmy kurierskie maja najprawdopodobniej system „Windows ce/mobile" na swoich urządzenia. Ja ze względu na brak takiego urządzenia (mobilnego z Windowsem) zrealizuje zadanie na Androidzie.

Rozdział 2

Wykorzystane technologie

Zrealizowany projekt bazuje na nowoczesnych technologiach. Wykorzystano urządzenie mobilne – telefon komórkowy z systemem Android, bazę danych do przechowywania informacji, a także serwer, który to łączy wszystkie elementy w spójną całość. Głównym językiem programowania użytym w projekcie jest język Java, dzięki któremu zrealizowano aplikację mobilną, obsługę Servletu, bazy danych, odpytywania i parsowania odpowiedzi serwera Google o odległości pomiędzy dwoma wskazanymi punktami, a także obsługa witryny zapytań z aplikacji webowej. Kompletny system powstał przy użyciu IDE Eclipse z odpowiednimi dodatkami.

2.1 Java

Java jest obiektowym językiem programowania ogólnego przeznaczenia. Charakteryzuje się silnym ukierunkowaniem na obiektowość oraz niezależnością od architektury sprzętowej i przenośnością aplikacji pomiędzy platformami. Oprócz wyżej wymieniowych założeniami języka Java jest prostota, sieciowość, niezawodność, bezpieczeństwo, interpretowalność, wysoka wydajność, możliwość programowania współbieżnego.

Java to język:

- Prosty – założeniami autorów języka Java było, aby programista bez specjalnych szkoleń mógł od razu zacząć pisać proste aplikacje. Składnia została uproszczona (w stosunku do C++) o arytmetykę wskaźnikową, struktury, unie, przeciążanie operatorów itd.

Rysunek 2.1: Logo Java EE.

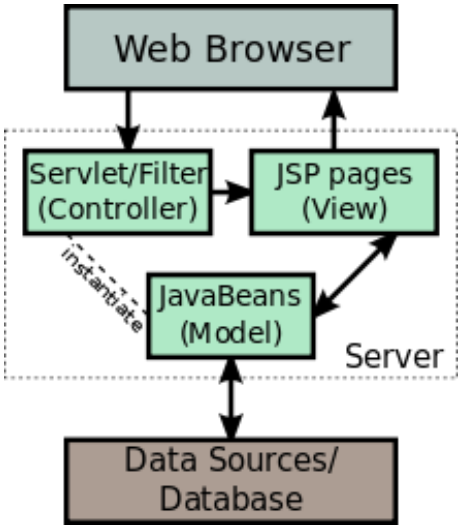
- Zorientowany obiektowo;
- Sieciowy — posiada wbudowaną bibliotekę, która w przystępny sposób umożliwia pracę z protokołami http, TCP/IP, FTP;
- Niezawodny — szczególnie skupiono się na wykrywaniu ewentualnych błędów w czasie kompilacji, zapobieganiu sytuacjom, w których błąd może wystąpić oraz obsłudze błędów w czasie działania programu;
- Bezpieczny — w związku z naciskiem położonym na zastosowania sieciowe zadbano o możliwie najlepsze zabezpieczenie przed wirusami i ingerencją osób trzecich;
- Niezależny od architektury — kod źródłowy kompilowany jest do kodu pośredniego (bajtowego), który następnie jest interpretowany na maszynie wirtualnej Javy(JVM), dostosowanej do odpowiedniego systemu i platformy sprzętowej. Maszyna wirtualna Javy jest zdolna wykonywać program z kodu pośredniego. Z tego powodu język Java stosowany jest na wielu urządzeniach oraz różnych systemach operacyjnych. Niestety konsekwencją przenośności kodu jest może być jego suboptymalna wydajność, a w konsekwencji wolniejsze działanie;
- Przenośny — posiada ściśle określone rozmiary typów danych i nie ma możliwości zmiany ich rozmiaru przez programistę czy np. zmiana kolejności bitów;
- Interpretowany — program nie jest kompilowany tylko przechowywany w postaci kodu bajtowego, podczas uruchomienia zostaje dopiero interpretowany i uruchamiany przez interpreter języka;

- Wydajny — istnieje możliwość tłumaczenia kodu bajtowego w locie, co zwiększa szybkość ładowania się programu;
- Wielowątkowy — umożliwia przetwarzanie współbieżne z wykorzystaniem wątków, a także pracę w czasie rzeczywistym;
- Dynamiczny — obiekty w Javie można zmieniać w zależności od zmieniającego się środowiska, a także możliwy jest wgląd we wszystkie obiekty, a nawet dodawać nowe metody;

Język Java wywodzi się z języków C++ i C. Wykorzystuje wiele potrzebnych i użytecznych funkcji tych języków, a z mniej użytecznych, trudnych w implementacji lub często powodujących błędy – zrezygnowano. Język Java umożliwia dziedziczenie, a ponadto wszystkie obiekty Javy są podklasą obiektu bazowego. Jednakże Java nie umożliwia dziedziczenia wielobazowego, dlatego wprowadzono interfejsy - abstrakcyjny typ, który definiuje jedynie metody bez implementacji i nie posiada stanu.Z tego powodu można jedynie implementować interfejs lub wykorzystać go do utworzenia klas anonimowych. Język Java umożliwia pisanie aplikacji stacjonarnych, webowych czy mobilnych. Duży nacisk położono na zaprojektowanie systemu wyjątków. Zbieraniem i ponownym wykorzystaniem obiektów bez referencji zajmuje się wbudowany `GarbageCollector` (kolektor nieużytków) [3].

2.2 Wzorzec architektoniczny – MVC

W projekcie do zorganizowania struktury aplikacji serwerowej został zastosowany wzorzec architektoniczny MVC (Model-View-Controller). W modelu tym Model jest odpowiedzialny za przechowywanie logiki, z której korzystają inne składowe systemu. Kolejną częścią składową tej struktury jest Widok, który to jest odpowiedzialny za funkcje prezentacji w ramach interfejsu użytkownika, ale także może posiadać swoją logikę. Ostatnim elementem systemu jest Kontroler, który spaja dwie wcześniejsze części. Kontroler odpowiada za przepływ danych od/do użytkownika, reakcję systemu w zależności od zachowania użytkownika, kontroler także zarządza Modelem i Widokiem. Takiej strukturze jest jasno zdefiniowane, która część systemu pełni jakie funkcje. Taka struktura architektoniczna jest najczęściej stosowana w aplikacjach WWW, gdzie widać wyraźną granicę pomiędzy widokiem i modelem, a kontrolerem jest serwer, który obsługuje informacje płynące z widoku (z http), odpowiednio formuje model i przekazuje go do widoku. [7]



Rysunek 2.2: Schemat systemu Model-View-Controller model 2[8]

Autor w swojej pracy użył modelu MVC2 [rys. 2.2] Uzasadnieniem użycia tego wzorca jest ułatwiona organizacja aplikacji, w której istnieje interfejs graficzny użytkownika. Dzięki niemu w prosty logiczny sposób można było rozdzielić logikę, kontrolę i widok. Kontrolerem z [rys. 2.2] jest opisany w kolejnym podrozdziale Servlet.

2.3 Servlet

Serwlety to aplikacje działające na serwerze WWW napisane w języku Java, mające zapewnić działanie aplikacji internetowych niezależnie od platformy. Umożliwia korzystanie z baz danych i obsługę żądań i odpowiedzi HTTP. Z tego powodu wykorzystywane są do budowania interaktywnych aplikacji internetowych.

Serwer Apache obsługuje komunikację z klientem za pomocą protokołu HTTP. Wybrana w projekcie dystrybucja Tomcat Apache [rys. 2.3] jest projektem o otwartym kodzie źródłowym. Wykorzystuje wielowątkowość, jest skalowalny i bezpieczny oraz zapewnia kontrolę dostępu [2].

Rysunek 2.3: Logo Apache i Apache Tomcat.

Wybór Tomcat Apache na serwer podyktowany był przez wybór Javy jako głównego języka projektu. Serwlet jest dość popularnym narzędziem, co pomogło także w uruchomieniu i skonfigurowaniu go.

2.4 JavaServer Pages

JSP (ang. JavaServer Pages) jest to technologia, dzięki której możliwe jest dynamiczne tworzenie stron internetowych. JSP bazuje na językach znaczników, np. HTML, XML oraz innych. Technologia ta jest kompatybilna z Serwletami (Apache Tomcat i innymi).

To właśnie wprowadzenie plików JSP wymaga korzystania z wcześniej opisanego modelu MVC [rys. 2.2]. W dodatku JSP oprócz użycia języków skryptowych umożliwia przeplatanie ich z językiem Java. Wtedy kod, który będzie napisany w Javie musi być ujęty w znaki <% ... %>, np. fragment z listingu 3.8:

```
<% out.print("Nadawca"); %> ${regUserName} ${regUserAddr}
```

Natomiast frazy ujęte w znaki ${...} służą do dostępu (pobrania i/lub wysyłania) do argumentów i funkcji, które powstały w obiektach Javy. Żeby przekazać taką wartość z klasy Javy, klasa ta musi rozszerzać klasę Servlet (`extends HttpServlet`) oraz ustawiać parametr o nazwie jak w pliku *.jsp {} dodając go do kontekstu `HttpServletRequest`; jako przykład podano fragment z listingu 3.9:

```
req.setAttribute("regUserName", searchParcel.getRegisteredUserNameUser());
```

2.5 Technologie internetowe

W przedstawionym w tej pracy projekcie korzystano z technologi internetowych, które zapewniały interakcję z użytkownikiem oraz obsługę stron WWW. Skorzystano z takich technologii jak:

- JavaScript — jest to skryptowy język programowania stosowany głównie do tworzenia stron internetowych, zapewnia interakcję z użytkownikiem[9], służy do kontroli przeglądarki, zmiany treści strony. Składnia języka JavaScript jest zbudowana na podstawie języka C. Język umożliwia dynamiczne typowanie oraz jest obiektowy;

- HTML — jest to język znaczników służący do tworzenia stron internetowych. Język ten składa się z tagów umieszczonych w nawiasach trójkątnych (`<...>`), które mogą zawierać inne elementy HTML lub, np. napisy umieszczane są pomiędzy tagami `<tag> napis </tag>`, gdzie pierwszy znacznik jest tagiem oznaczającym początek, a drugi – zamykającym. HTML jest budulcem strony internetowej opisującym jej strukturę. Można go osadzać w takich językach jak JavaScript, a do definiowania wyglądu i układu tekstu i innych elementów na stronie można użyć stylów CSS;

- XML — jest to język znaczników przeznaczony do reprezentowania danych w strukturyzowany sposób [12]. Tak jak HTML składa się ze znaczników `<...>` i `</...>` lub samozamykających `<.../>`.

- Protokół HTTP — jest podstawy protokół komunikacji w internecie służący do wymiany informacji. Funkcjonuje w trybie żądanie-odpowiedź w modelu klient-serwer, gdzie przykładowo przeglądarka jest klientem, a aplikacja uruchomiona na komputerze serwerem. HTTP jest protokołem warstwy aplikacji, która zapewnia komunikację.

2.6 Baza Danych – MySQL

W projekcie do przechowywania danych skorzystano z baz danych. Baza danych pozwala w ustrukturyzowany sposób kolekcjonować dane niezbędne do działania programów. Przechowywane dane mogą być w dowolnym formacie i strukturze.

Systemem bazodanowym wykorzystanym w projekcie był MySQL rozwijanym przez firmę Oracle. Jest rozwiązanie o otwartym kodzie źródłowym do zarządzania relacyjnymi bazami danych. Charakteryzuje się takimi cechami jak szybki, wielowątkowy dostęp z możliwością obsłużenia dużej ilości użytkowników. Serwer MySQL może być stosowany do systemów, w których znajdują się dane o znaczeniu krytycznym lub te systemy są mocno obciążane [10].

Rysunek 2.4: Logo MySQL [10]

Język MySQL posiada takie typy danych jak signed/unsigned int, long, float, double, char, varchar, date, time, datetime, enum i inne. Język MySQL składa się z takich typów komend jak: DML (ang. Data Manipulation Language – dotyczące manipulowania), DLL (ang. Data Defitnition Language) oraz DCL (ang. Data Control Language). Komendy DML:

- `select` — służy do otrzymywania wierszy z wybranych tabeli, najpopularniejsza forma użycia komendy `select`:

  ```
  SELECT select_expr [, select_expr] [FROM table_name] [WHERE where_condition];
  ```

- `insert` — komenda ta wstawia nowe wiersze do istniejącej tabeli, istnieją trzy przypadki użycia tej komendy:

  ```
  INSERT [INTO] table_name [(col_name,...)] VALUES | VALUE (expr | DEFAULT,...),
  (...), ... [ ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ... ];

  INSERT [INTO] table_name SET col_name=expr | DEFAULT, ... [ ON DUPLICATE KEY
  UPDATE col_name=expr [, col_name=expr] ... ];

  INSERT [INTO] table_name [(col_name,...)] SELECT ...
  [ ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ... ];
  ```

- `update` — aktualizuje kolumny istniejących wierszy, struktura użycia komendy:

  ```
  UPDATE table_name SET col_name1=expr1|DEFAULT [, col_name2=expr2|DEFAULT] ...
  [WHERE where_condition];
  ```

- `delete` — usuwa pojedyncze wiersze, użycie komendy:

  ```
  DELETE FROM table_name [WHERE where_condition];
  ```

- i inne

Komendy DLL:

- `create` — komenda stosowana w połączeniu z database, function, index, procedure, table, trigger, view. Komenda używana jest do dodawania nowej bazy danych czy tabeli;
- `alter` — komenda używana w połączeniu z database, table, function, procedure oraz view. Służy do zmiany struktury bazy danych, tabeli, itp., np. można dodać lub usunąć kolumnę istniejącej tabeli;
- `drop` — umożliwia usunięcie bazy danych i tabeli.

Komendy DML:

- `commit` — zatwierdza wpisaną transakcję, wprowadzona zmiana jest permanentna;
- `rollback` — wycofuje ostatnią wpisaną transakcję.

Oprócz wyżej wymienionych MySQL posiada również inne komendy, jednak są one znacznie rzadziej wykorzystywane, dlatego nie zostały tutaj przytoczone.

2.7 Android

Android jest systemem operacyjnym wykorzystywanym na platformach mobilnych, zbudowanym w oparciu o jądro Linux. Umożliwia tworzenie aplikacji na wiele urządzeń – dokonywane jest przez optymalizację pliku XML, w którym dostosowuje się aplikację do konkretnego urządzenia. System Android ma taką właściwość, że każdy włączona aplikacja jest uruchamiana na maszynie wirtualnej i jest niezależna od pozostałych. Najnowszą wersją systemu jest Lollipop 5.0.

Rozpoczęcie pracy deweloperskiej z Androidem zaczyna się od instalacji środowiska — Eclipse z dodatkiem SDK Android lub Android Studio. Po zainstalowaniu IDE projektowanie aplikacji zaczyna się od opracowania layout'u interfejsu użytkownika, następnie przechodzi się do oprogramowania funkcjonalności oraz logiki aplikacji. Po zakończeniu etapu budowania aplikacji następuje testowanie jej [1].

System Android składa się z czterech podstawowych elementów:

- Activites — reprezentuje ekran użytkownika, przykładem jest aplikacja do obsługi emaila, która posiada jedną aktywność do listowania nowych wiadomości i inną do pisania ich. Aktywności implementuje się używając podklasę `Activity`;
- Services — jest to składnik aplikacji, który działa w tle. Przykładem `Services` może być odtwarzacz muzyki, który jest uruchomiony w tle i w tym samym czasie użytkownik może korzystać z innych aplikacji w urządzeniu. Takie własność aplikacji implementuje się za pomocą podklasy `Service`;

Rysunek 2.5: Logo systemu Android [1]

- Content providers — służy do zarządzania wspólnymi danymi, a także danymi, które są prywatne dla danej aplikacji. Zarządzanie danymi implementuje się stosując podklasę `ContentProvider`;
- Broadcast receivers — komponent systemu, który jest odpowiedzialny za rozgłaszanie informacji po systemie, np. informacja o niskim stanie baterii jest wyświetlana niezależnie od aktywności w urządzeniu. Taką cechę aplikacji uzyskuje się implementując podklasę `BroadcastReceiver`.

Aplikacja w systemie Android składa się z wielu części, które są ze sobą luźno powiązane. Zazwyczaj jedna część aplikacji jest główna – najczęściej ta, która jest pokazywana użytkownikowi po uruchomieniu aplikacji. Następnie każde działanie wykonane na tym ekranie może uruchamiać inny ekran, proces czy działanie. Wtedy przy każdym uruchomieniu nowej działalności wcześniejsza jest zatrzymywana i przechowywana na stosie – LIFO (ang. last in first out). Dzięki temu użytkownik ma możliwość wrócić do poprzedniej działalności (ekranu) cofając się, wtedy zdejmowana jest ona ze stosu w takim stanie jakim była pozostawiona.

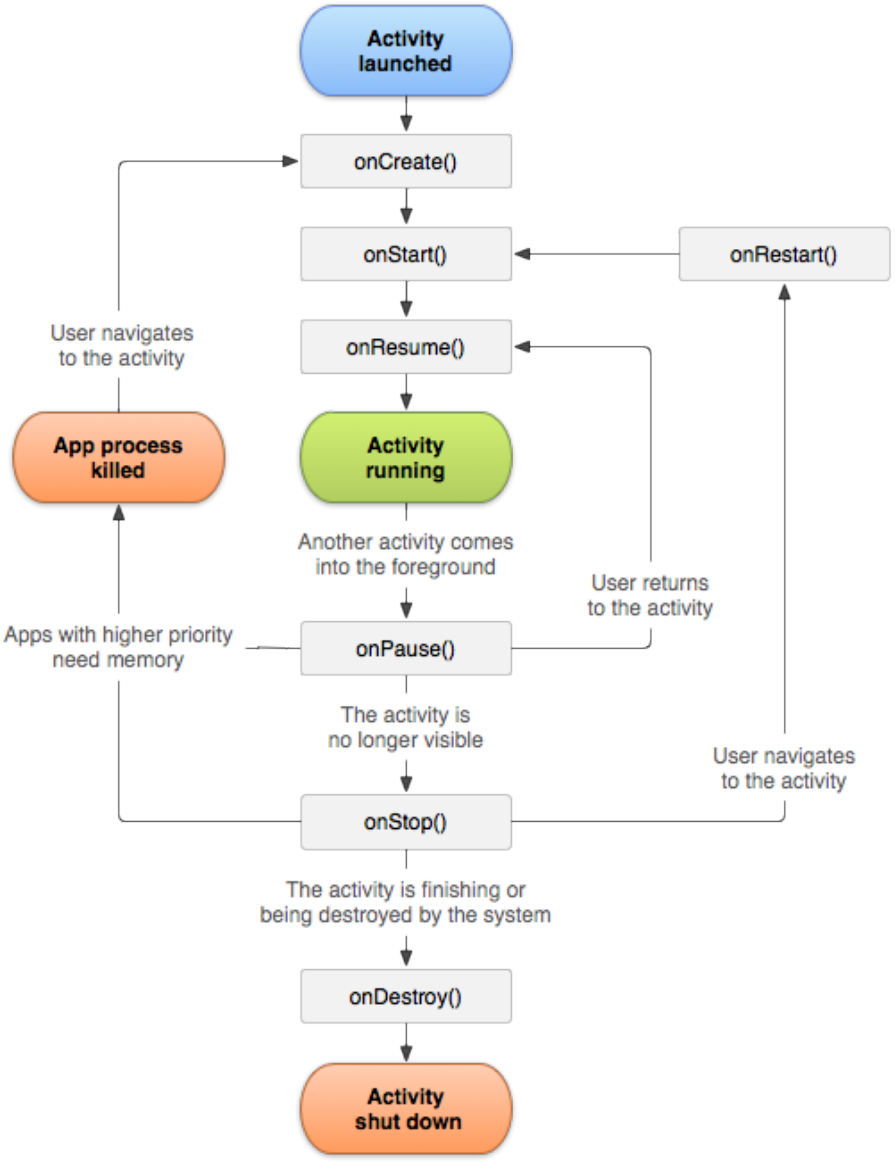
Warty uwagi jest element `Activity`, reprezentujący ekran użytkownika, oferujący współdziałanie z użytkownikiem. Najczęściej okno aplikacji wypełnia cały wyświetlacz, ale może być mniejsze lub pływające na wierzchu innych okien.

Utworzenie aplikacji typu `Activity` wymaga zaimplementowania podklasy `Activity`, w tej podklasie należy zdefiniować metody, które będą określały aplikację pomiędzy różnymi stanami cyklu jej życia. Przez cykl życia aplikacji rozumie się jej tworzenie, zatrzymywanie, wznawianie, niszczenie. Metoda `onCreate()` określa stan aplikacji podczas jej tworzenia. Metoda `onPause()` definiuje stan aplikacji w momencie opuszczenia jej przez użytkownika. Opuszczenie aplikacji nie musi oznaczać jej zakończenia, typowo zachowywany jest aktualny stan.

Z powyższego da się zauważyć, że istotną kwestią jest zarządzanie życiem aplikacji. Aktywność może być zatem w jednym z trzech stanów: stan wznawiania, wstrzymanie oraz zatrzymania. Stan wznawiania określa stan, w którym aplikacja jest włączona i znajduje się na głównym planie ekranu użytkownika. Stan wstrzymania dotyczy stanu, w którym aplikacja jest widoczna (włączona), jednak nie znajduje się na głównym ekranie. Stan zatrzymania oznacza, że aplikacja jest całkowicie przysłonięta przez inne i pracuje w tle. W stanie wstrzymania i zatrzymania system, w przypadku braku pamięci, może zatrzymać aplikację.

Na rysunku 2.6 przedstawiony został cykl życia aktywności. Wspomniana metoda `onCreate()` tworzona jest zawsze wtedy, gdy aplikacja jest uruchamiana po raz pierwszy, w tym miejscu tworzone są widoki. Następną wywoływaną metodą jest `onStart()`, która wywołuje pokazanie się aplikacji pod restarcie lub zatrzymaniu aplikacji. Metoda `onResume()` wywoływana jest zawsze po pauzie i rozpoczyna interakcję z użytkownikiem, odpowiada za przywołanie aplikacji na szczyt stosu. `onPause()` wywoływana jest w momencie rozpoczęcia innej czynności. Metoda `onStop()` zostaje wykonana, gdy aplikacja ma przejść w stan niewidoczny dla użytkownika – po wywołaniu tej metody aplikacja może zostać zniszczona, zamknięta

lub zrestartowana. Ostatnią metodą przed zniszczeniem aplikacji jest `onDestroy()` – zamyka wszystkie otwarte procesy.



Rysunek 2.6: Schemat cyklu życia aktywności[1]

2.8 Google Maps API

Firma Google udostępnia korzystanie deweloperom ze swoich produktów [5]. W projekcie autor korzysta z Google Maps API. Google Maps jest to zbiór interfejsów http usług Google do stosowania w połączeniu z mapą. Wysłanie zapytania http o określonym formacie URL o położenie, trasę czy odległość pomiędzy punktami generuje odpowiedź po stronie serwera. Po zapytaniu odpowiedz dostarczana jest w formacie JSON lub XML. W projekcie autor korzystał z dwóch usług Google Maps API – jedną była `Google Maps JavaScript API v3 Directions Service`, drugą `Distance Matrix API`.

`Google Maps JavaScript API v3` jest usługą Google, która oblicza trasę pomiędzy dwoma punktami za pomocą obiektu `DirectionsService`. Wspomniany obiekt łączy się z serwisem `Google Maps API`, uzyskuje wskazówki co do żądania obliczenia trasy i zwraca wyniki. Do wyrysowania mapy i trasy z uzyskanej odpowiedzi `DirectionsService` korzysta się z `DirectionsRenderer`. Zapytanie `JavaScript API V3 Directions Service` jest wykonane w języku JavaScript.

Rysunek 2.7: Logo Google Developers [5]

Na listingu 2.1 przedstawione zostały wszystkie możliwe parametry jakie może przyjmować obiekt `DirectionsRequest`. Oczywiście większość tych parametrów jest opcjonalna, wymagane są jedynie `origin`, `destination` i `travelMode`.

```
{
  origin: LatLng | String,
  destination: LatLng | String,
  travelMode: TravelMode,
  transitOptions: TransitOptions,
  unitSystem: UnitSystem,
  durationInTraffic: Boolean,
  waypoints[]: DirectionsWaypoint,
  optimizeWaypoints: Boolean,
  provideRouteAlternatives: Boolean,
  avoidHighways: Boolean,
  avoidTolls: Boolean
  region: String
}
```

Listing 2.1: Parametry jakie może przyjmować obiekt typu `DirectionsRequest`

Na powyższym listingu przedstawione parametry oznaczają:

- `origin` — punkt startowy;
- `destination` — punkt końcowy;
- `travelMode` — rodzaj transportu: pojazd, rower, tranzyt, na pieszo;
- `transitOptions` — określa, czy wynik zapytania powinien zawierać informację o natężeniu ruchu;
- `unitSystem` — określa w jakich jednostkach ma być wyświetlany dla użytkownika wynik: jednostki metryczne lub jednostki imperialne;
- `durationInTraffic` — wynik zawiera czas podróży przy aktualnym natężeniu ruchu;
- `waypoints[ ]` — punkty pośrednie trasy
- `optimizeWaypoints` — optymalizuje trasę na podstawie dostarczonych punktów pośrednich trasy pod kątem najkrótszej drogi;
- `provideRouteAlternatives` — gdy wartość ustawiona jest na `true` może dostarczać więcej niż jedną trasę alternatywną;
- `avoidHighways` — wyznaczanie trasy stara się omijać autostrady, gdy parametr ustawiony jest na `true`;
- `avoidTolls` — gdy parametr ustawiony jest na `true` obliczona trasa powinna unikać dróg płatnych;
- `region` — kod regionu (kraju), np. „pl”.

Zapytanie o trasę wykonuje się poprzez inicjalizację `DirectionsService` metodą `route()`, przyjmującą parametry `request` i `response`. Do parametru `request` podaje się wcześniej utworzoną strukturę (listing 2.1). W odpowiedzi zwracana jest trasa (`DirectionsResult`) i status odpowiedzi (`DirectionsStatus`). `DirectionsStatus` może przyjmować między innymi takie wartości jak: `OK`, `NOT_FOUND`, `UNKNOWN_ERROR`. Nim przystąpi się przed wyświetlaniem trasy należy sprawdzić status odpowiedzi. `DirectionsResult` zawiera wynik zapytania, który zostaje przekazany do obiektu `DirectionsRenderer` i dopiero w nim tworzony jest obraz mapy.

```
directionsService.route(directionsRequest, function(directionsResult, directionsStatus) {
  if (directionsStatus == google.maps.DirectionsStatus.OK) {
    directionsRenderer.setDirections(directionsResult);
  }
});
```

Listing 2.2: Przykład wywołania funkcji `route()`

`Distance Matrix API` jest podstawową usługą Google Maps API. Wynikiem zapytania jest wyznaczenie odległości pomiędzy puntem początkowym i docelowym, trasa wyznaczana jest na podstawie zalecanej trasy. W skład odpowiedzi oprócz odległości podawany jest czas. Ta usługa nie zawiera szczegółowych informacji. Zapytanie tworzone jest na podstawie zapytania http:
`https://maps.googleapis.com/maps/api/distancematrix/output?parameters`
`Output` jest formatem w jakim podany jest wynik – JSON lub XML. `Parameters`

- `origins` — punkt początkowy;
- `destinations` — punkt końcowy;
- `key` — indywidualny klucz dewelopera, określa ilość zapytań;
- `mode` — definiuje środek transportu (pojazd, na pieszo, rower);
- `language` — określa język w jakim zwracany jest wynik;
- `avoid` — informuje serwis, co ma omijać trasa (autostradę, drogi płatne, prom);
- `units` — określa w jakich jednostkach ma być wyświetlany wynik: jednostki metryczne lub jednostki imperialne;
- `departuretime` — określa czas wyjazdu, dla którego ma zostać wyliczona trasa

Przykładowe zapytanie:

```
https://maps.googleapis.com/maps/api/distancematrix/xml?origins=Wroclaw
    ↪&destinations=Warszawa&mode=bicycling&language=pl-PL&key=API_KEY
```

Wynik tego zapytania jest przedstawiony w XML i jego wynik zaprezentowano na listingu 2.3.

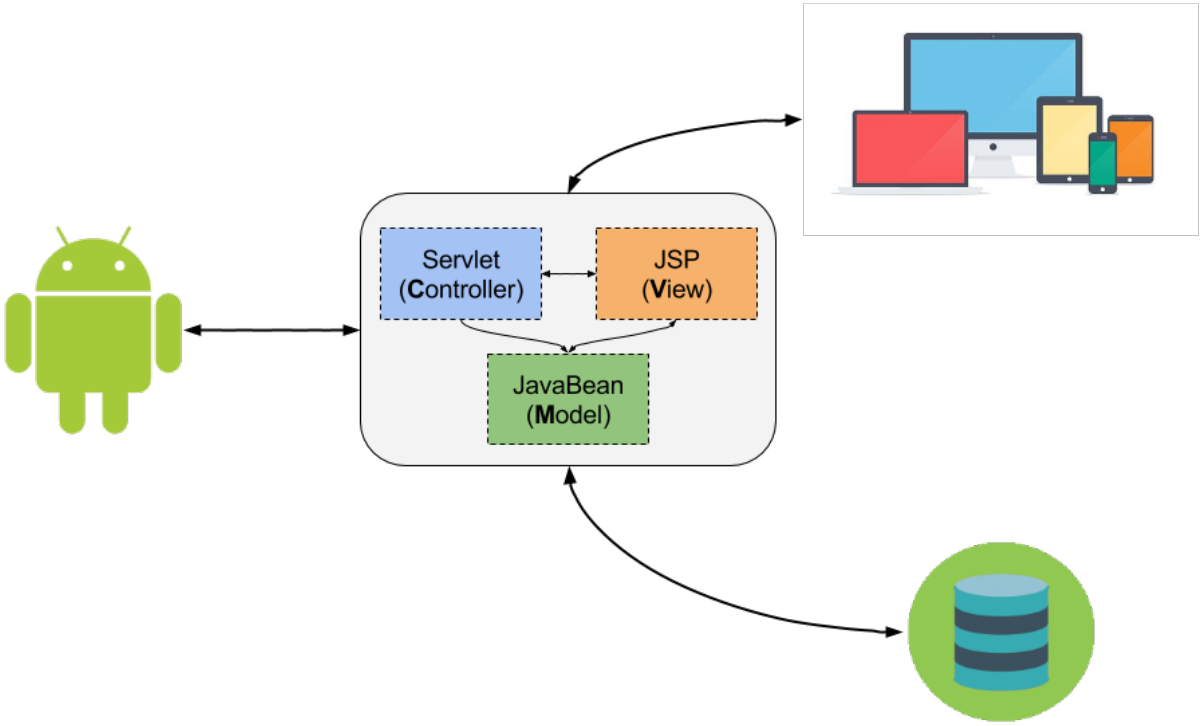
```
<DistanceMatrixResponse>
  <status>OK</status>
  <origin_address>Wroclaw, Polska</origin_address>
  <destination_address>Warszawa, Polska</destination_address>
  <row>
  <element>
    <status>OK</status>
    <duration>
      <value>64870</value>
      <text>18 godz. 1 min</text>
    </duration>
    <distance>
      <value>357656</value>
      <text>358 km</text>
    </distance>
  </element>
  </row>
</DistanceMatrixResponse>
```

Listing 2.3: Przykład odpowiedzi na zapytanie `DistanceMatrix` trasa Wrocław – Warszawa

Rozdział 3

Rozwiązanie – prezentacja wyników

Zrealizowana aplikacja składa się z dwóch części, tj. aplikacji na Androida oraz serwera WWW. Serwer, czyli Servlet (aplet Javy) łączy się z bazą danych, w której przechowywane są informacje o przesyłce, kurierach i klientach. Aplikacja mobilna również łączy się z tą samą bazą danych, jednakże połączenie to zrealizowano pośrednio. Aplikacja poprzez zapytania i odpowiedzi z serwera uzyskuje informacje o stanie bazy danych. Ideę systemu zaprezentowano na rysunku 3.1.



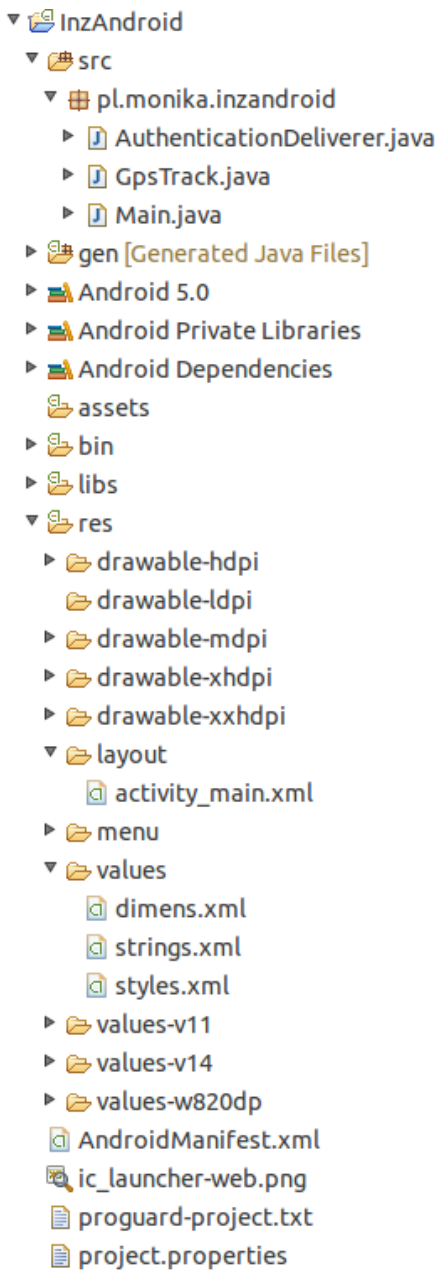
Rysunek 3.1: Struktura systemu

Zaprojektowaną aplikację zrealizowano w środowisku programistycznym Eclipse. Wybór takie środowiska programistecznego uwarunkowany był dużym wyborem dodatków, rozszerzeń i możliwości jaką on oferuje.

3.1 Aplikacja na system Android

Założeniem aplikacji mobilnej było lokalizowanie kuriera i wysyłanie jego pozycji na serwer, który zapisuje ją w bazie danych. Zasada działania aplikacji polega na odpytaniu kuriera o jego numer id, a następnie sprawdzeniu czy jest włączony GPS. Ponadto aplikacja wykrywa czy wprowadzono poprawne id oraz zapobiega zalogowaniu się dwóch kurierów na tym samym id. Po wpisaniu poprawnego id kuriera aplikacja zapamiętuje go i do automatycznie wysyła swoją pozycję.

Stworzenie aplikacji na system Android zostało rozpoczęte od instalacji pluginu ADT (Android Development Tools) dla programu Eclipse. W skład ADT wchodzi między innymi Android SDK (Software Development Kit). Android SDK zawiera w sobie takie elementy jak Tools – służy do tworzenia aplikacji niezależnie od wersji systemu Android oraz Platform Tools – narzędzia stworzone pod kątem wersji systemu Android. W skład SDK Tools wchodzą takie funkcje jak: zarządzanie projektami, modułami czy maszynami wirtualnymi, debugger czy emulator, Platform Tools natomiast zawiera biblioteki do systemu Android. Plugin ADT korzysta z funkcji Android SDK i pomaga tworzyć, budować, instalować oraz debugować aplikacje na system Android w środowisku Eclipse. W programie Eclipse tworzona jest z aplikacją odpowiednia struktura projektu [rys. 3.2], w której jest podział na klasy zawierające logikę aplikacji (src), pliki generowane przez kompilator (gen), folder na pliki z zasobami (assets), pliki binarne (bin), dodatkowe biblioteki (libs) oraz folder na zasoby (res), który odróżnia od assets to, że znajdują się w nim zasoby generowane do pliku R.java (nie trzeba podawać lokalizacji zasobu tylko jego nazwę). W katalogu projektu ustalana również jest konfiguracja systemu – „AndroidManifest.xml", layout (wygląd i ustawienie elementów na ekranie systemu Android), w podfolderach drawable umieszcza się pliki graficzne, w podfolderach values ciągi znaków (stringi, kolory i in.) i ustawienia, a w katalogu menu – dostępne opcje menu.



Rysunek 3.2: Struktura programu aplikacji Android

Projektowanie aplikacji Android zaczyna się od ustawienia layoutu aplikacji (/res/layout/activity_main.xml). Zaprojektowany przez autora layout jest prosty i przejrzysty, ponieważ ma wykonywać bardzo podstawowe funkcje [rys. 3.3]. I tak, zawiera w sobie pole do wpisywania id kuriera, pole wyświetlające komunikaty oraz przycisk wyłączający aplikację. Aplikacja została tak przemyślana, że aby ją wyłączyć trzeba użyć przycisku „Off", pozostałe sprzętowe przyciski nie wyłączają aplikacji, jedynie ją minimalizują. Takie właściwości zostały stworzone z myślą o tym, aby kurier podczas używania aplikacji tylko w świadomy sposób mógł ją zamknąć.

Autor przewidział również odpowiednie zachowanie aplikacji, gdy kurier próbuje zalogować się na nie istniejące w bazie id lub na id, które jest już zajęte - tzn. na które już zalogował się inny kurier [rys. 3.4(a)]. Oprócz wyżej wymienionych zachowań aplikacji, w pasku statusu na telefonie wyświetlana jest ikona podczas włączonej aplikacji a także w powiadomieniach [rys. 3.4(b)].

Kolejnym krokiem było nadanie odpowiednich uprawnień aplikacji, do czego służy plik „AndroidManifest.xml". Aplikacji zostały przyznane uprawnienia do sprawdzania statusu sieci komórkowej oraz WiFi, sprawdzania statusu sygnału GPS, a także do korzystania z wyżej wymienionych [listing 3.1].

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

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

Listing 3.1: Nadanie uprawnień aplikacji Android w pliku AndroidManifest.xml

Po przygotowaniu wyglądu aplikacji oraz nadaniu jej uprawnień można przejść do oprogramowania logiki aplikacji. Klasa główna, która zajmuje się obsługą aplikacji dziedziczy po klasie `Activity`. Klasa `Activity` zajmuje się obsługą interakcji pomiędzy użytkownikiem a urządzeniem. Klasa ta tworzy okno aplikacji oraz na przykład umieszcza ustawione wcześniej przyciski interfejsu użytkownika. Znajdują się w niej takie metody jak `onCreate()`, `onDestroy()`, `onStart()`, `onStop()` i inne. Metodę `onCreate()` należy przesłonić, jeśli mają być zainicjowane wcześniej wspomniane przyciski i pola z layoutu. Autor w swojej aplikacji nadaje właściwość dla przycisku, która ma po kliknięciu go w dowolnym momencie działania aplikacji wywołać zamknięcie aplikacji. Takie zachowanie osiąga się poprzez wywołanie na rzecz niego metody `public void setOnClickListener(View.OnClickListener l)`[listing 3.2].

```

button.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    if (savedId != "")
      new AuthenticationDeliverer().execute(savedId, "0", "0").get();
    savedId = "";
    onDestroy();
  }
});
```

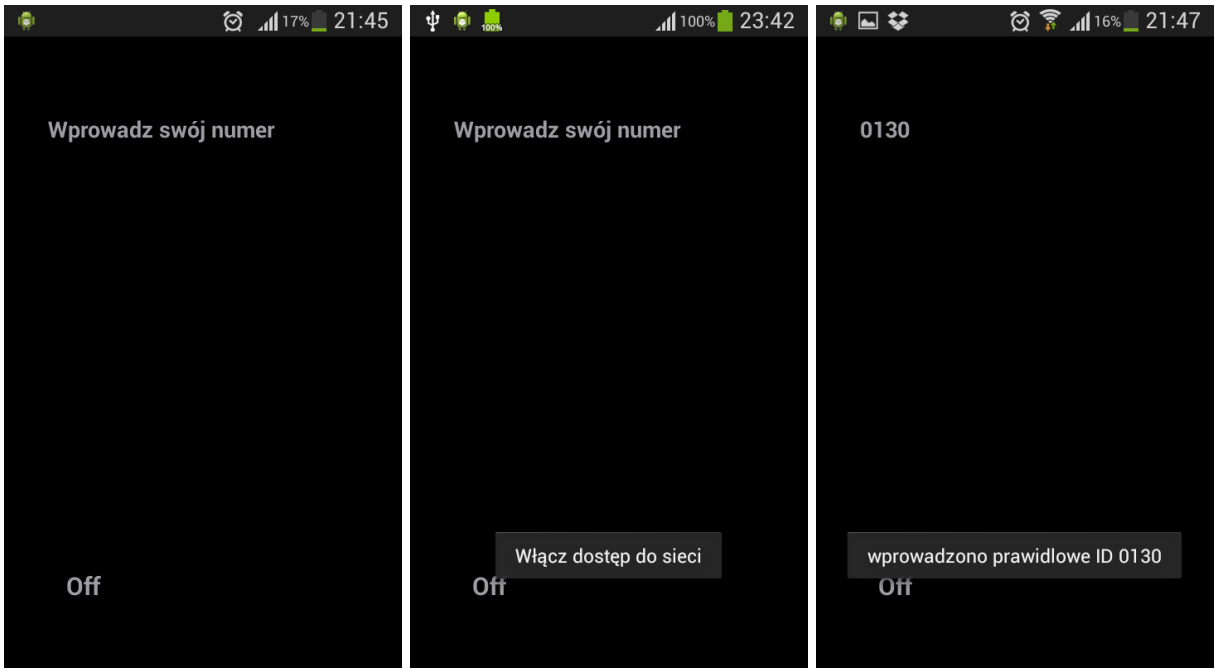
Listing 3.2: Ustawienie właściwości przycisku “Off” w głównej klasie aplikacji mobilnej w metodzie onCreate()

Na listingu 3.2 widać zmienną `savedId`, jest pole klasy, które zapisywane jest wpisanym przez kuriera jego `id`. Gdy użytkownik wyłącza aplikację system sprawdza czy `savedId` nie jest puste i w przypadku gdy `savedId` posiada jakąś wartość tworzony jest nowy obiekt `AuthenticationDeliverer()` z odpowiednimi parametrami – nazwa zmiennej, status aktywności oraz status logowania. Klasa `AuthenticationDeliverer()` dziedziczy po `AsyncTask` (umożliwia tworzenie nowego wątku i pozwala na wykonywanie operacji w tle).

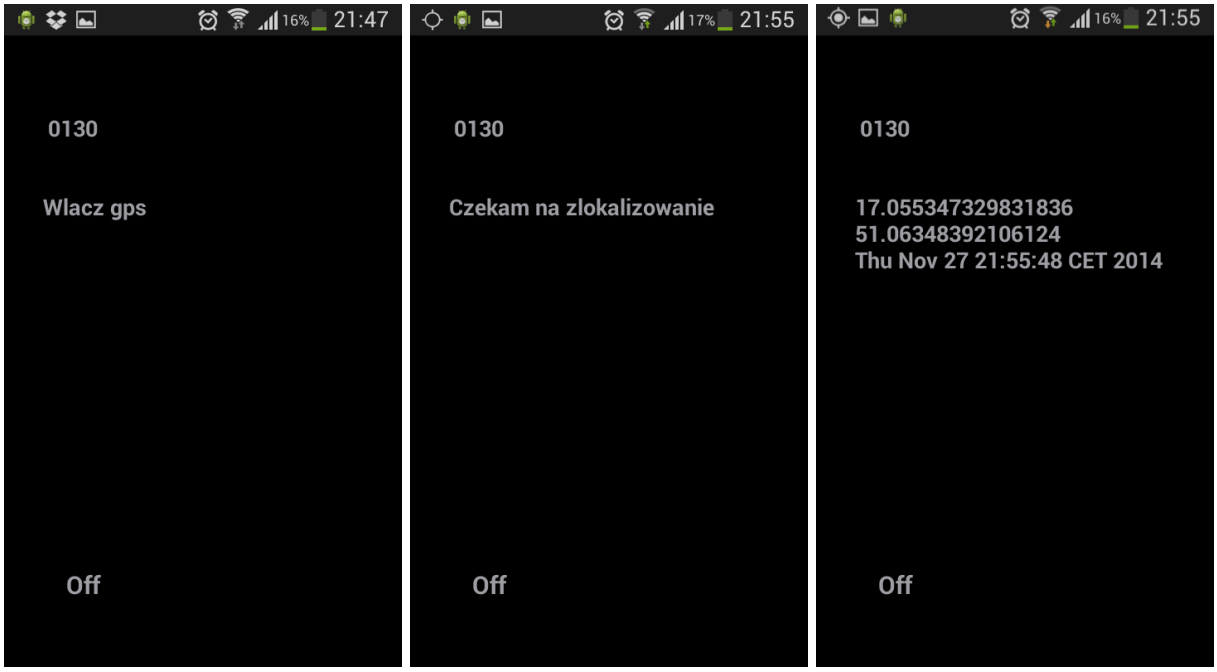
Parametry z jakimi tworzony jest nowy obiekt to id kuriera, status aktywności oraz jego logowanie lub wylogowywanie[listing. 3.3]. Status aktywności to informacja wysyłana do serwera (zapisywana w bazie danych) mówiąca o tym, czy kurier będzie ustawiał swój status na aktywny czy nieaktywny – czy rozpoczyna pracę czy ją kończy. Status aktywności sprawdzany jest ze statusem w bazie danych, jeśli kurier próbuje się zalogować, ale w bazie jest już ktoś zalogowany na ten id pojawi się komunikat o zalogowanym już kurierze o tym numerze [rys. 3.4(a)]. Natomiast trzeci parametr mówi o tym, czy kurier się zalogowywuje („1”) czy wylogowuje („0”) z aplikacji. Na rzecz klasy AuthenticationDeliverer() wywołane zostają metody: `execute()` – nakazuje wykonanie zadania, `get()` – oczekuje, aż zadanie zostanie wykonane.

```
protected String doInBackground(String... params) {
  ArrayList<NameValuePair> pairs = new ArrayList<NameValuePair>();
  pairs.add(new BasicNameValuePair("ID", params[0]));
  pairs.add(new BasicNameValuePair("activ", params[1]));
  pairs.add(new BasicNameValuePair("logout", params[2]));
  httpPostAuthentication.setEntity(new UrlEncodedFormEntity(pairs));
  new Thread(new Runnable() {
    @Override
    public void run() {
        httpClient.execute(httpPostAuthentication);
    }
  }).start();
  httpResponse = (new DefaultHttpClient()).execute(httpGetAuthentication);
  String entityStr = EntityUtils.toString(httpResponse.getEntity());
  if (entityStr.contains("logIn"))
    return "logIn";
  else if (entityStr.contains("busy"))
    return "busy";
  else if (entityStr.contains("false"))
    return "false";
  else if (entityStr.contains("logOut"))
    return "logOut";
  return "";
}
```

Listing 3.3: klasa AuthenticationDeliverer metoda doInBackground

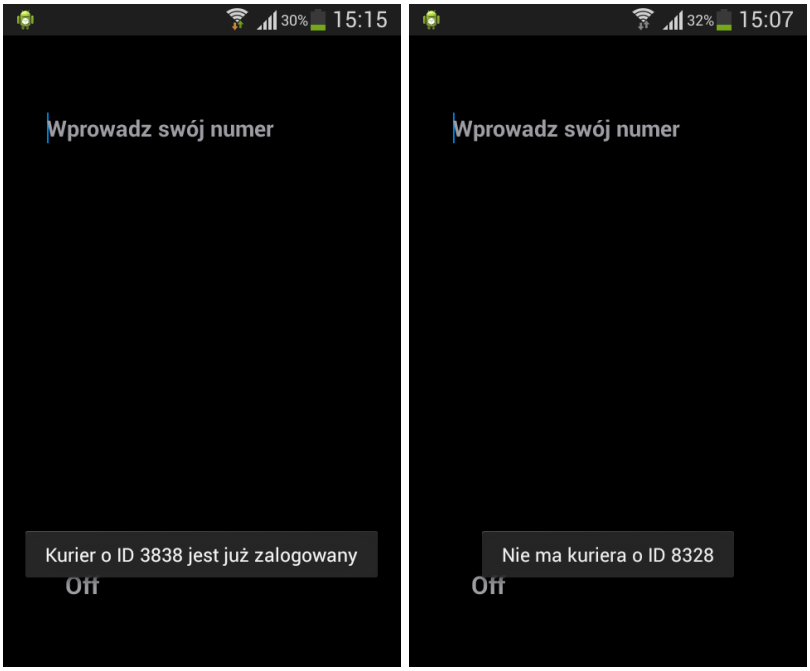


(a) (b) (c)

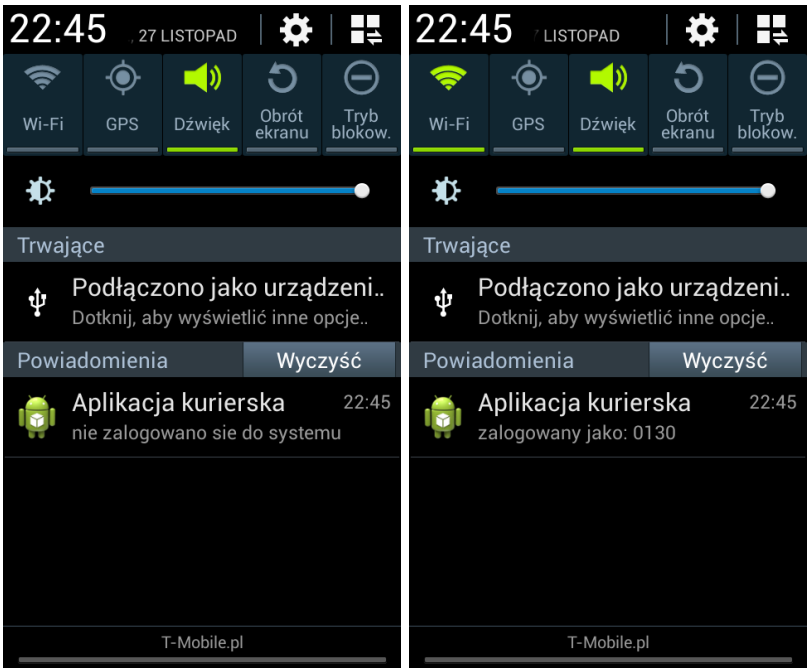


(d) (e) (f)

Rysunek 3.3: Ekrany Androida: (a) Główny ekran aplikacji, oczekuje na wprowadzenie id kuriera; (b) Gdy nie jest włączona sieć komórkowa lub WiFi na ekranie pokazuje się komunikat i zostaje zablokowane pole do wprowadzania id; (c) Komunikat o poprawnie wpisanym id; (d) Informacje o niewłączonym GPS; (e) System czeka na ustalenie lokalizacji; (f) System lokalizuje i wyświetla informacje o lokalizacji i czasie odczytu



(a) Sytuacja, w której id jest już zajęte lub nie istnieje



(b) Pasek status telefonu

Rysunek 3.4: Informacje o zalogowanym lub nieistniejącym Id oraz prezentacja uruchomionej aplikacji w pasku Statusbar

Po zainicjalizowaniu pola edycji zostaje wywołana na rzecz niego metoda `public void addTextChangedListener(TextWatcher watcher)`, która oczekuje, aż w polu tekstowym użytkownik wpisze odpowiedni ciąg znaków - dozwolone są tylko cyfry od 0 do 9. W tym przypadku wykorzystano metodę `abstract void afterTextChanged(Editable s)` z interfejsu `TextWatcher`. Po wpisaniu przez kuriera czterech cyfr następuje weryfikacja wpisanego id. Po pozytywnym przejściu weryfikacji id kuriera aplikacja sprawdza czy w urządzeniu jest włączony moduł GPS. Gdy spełnione zostaną wszystkie warunki uruchamiany jest handler, który sczytuje pozycję kuriera i wysyła ją na serwer wraz z datą odczyt.

Odczyt pozycji GPS dostępny jest dzięki użyciu klasy `LocationManager`, która zapewnia dostęp do systemowej usługi lokalizacji. `LocationManager` należy zainicjalizować instancją klasy `Context` (umożliwia dostęp do zasobów systemu i informacji o nim), natomiast `getSystemService` służy do kontroli pobierania lokalizacji. Następnie ustawiono z jaką częstotliwością ma być odczytywana zmiana pozycji. Po wykonaniu tych czynność następuje zapytanie o ostatnią znaną pozycję. Całą procedurę odczytu pozycji GPS pokazano na listingu 3.4.

```
31 public Location getLocation() {
32   locationManager = (LocationManager) context
33       .getSystemService(Context.LOCATION_SERVICE);
34   locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
35       minTime, minDistance, (android.location.LocationListener) this);
36   if (locationManager != null) {
37     location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
38     if (location != null) {
39       longitude = location.getLongitude();
40       latitude = location.getLatitude();
41     }
42   }
43   return location;
44 }
```

Listing 3.4: Pobieranie lokalizacji kuriera metoda getLocation() z klasy GpsTrack

Ostatnią istotną częścią jaka realizowana jest na systemie Android to połączenie z serwerem. Nim zostanie nawiązane połączenie konieczne jest przygotowanie treści wiadomości jaka ma zostać wysłana na adres serwer. Następuje to poprzez wpisanie par ciągów znaków do tablicy, z czego jedna jest kluczem identyfikator, a druga to jego wartość. Na jej podstawie budowany jest URL w stylu `httpPost` i wykonywany do zapytania http (tutaj na Servlecie) typu `Get`. Taka wiadomość może wyglądać w tym przypadku tak:

```
http://192.168.1.2:8080/inzServlet/insert?ID=0130&longitude=50.3545&latitude=11.3483
                                          ↪&timestamp=2014-11-25+15:14:55&activ=1
```

```
1  private String localhost = "192.168.1.2:8080";
2  private HttpClient httpClient = new DefaultHttpClient();
3  private HttpPost httpPost = new HttpPost("http://" + localhost
4      + "/inzServlet/insert");
5                                                    ⋮
6  private void startSendGpsDate() {
7    if (gpsTrack.isGpsEnable()) {
8      gpsTrack.getLocation();
9      double lat = gpsTrack.getLatitude();
10     double lon = gpsTrack.getLongitude();
11     if (lat != 0.0 && lon != 0.0) {
12       Date date = new Date(System.currentTimeMillis());
13       textView.setText(lon + "\n" + lat + "\n" + date);
14       ArrayList<NameValuePair> pairs = new ArrayList<NameValuePair>();
15       pairs.add(new BasicNameValuePair("ID", savedId));
16       pairs.add(new BasicNameValuePair("longitude", lon + ""));
17       pairs.add(new BasicNameValuePair("latitude", lat + ""));
18       pairs.add(new BasicNameValuePair("timestamp",
19           new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(date)));
20       pairs.add(new BasicNameValuePair("activ", 1 + ""));
21       httpPost.setEntity(new UrlEncodedFormEntity(pairs));
22       new Thread(new Runnable() {
23         @Override
24         public void run() {
25           httpClient.execute(httpPost);
26         }
27       }).start();
```

```
28      } else
29        textView.setText("Czekam na zlokalizowanie");
30    } else {
31      textView.setText("Wlacz gps");
32    }
33  }
```

Listing 3.5: Metoda startSendGpsDate() klasy Main.java. Metoda sprawdza stan sygnału GPS i przygotowuje wiadomość do wysłania na serwer oraz wysyła ją

W powyższym paragrafie zostały opisane kluczowe funkcje klas zaimplementowanych w systemie Android. W opisie i w listingach pominięto obsługę wyjątków.

3.2 Serwer

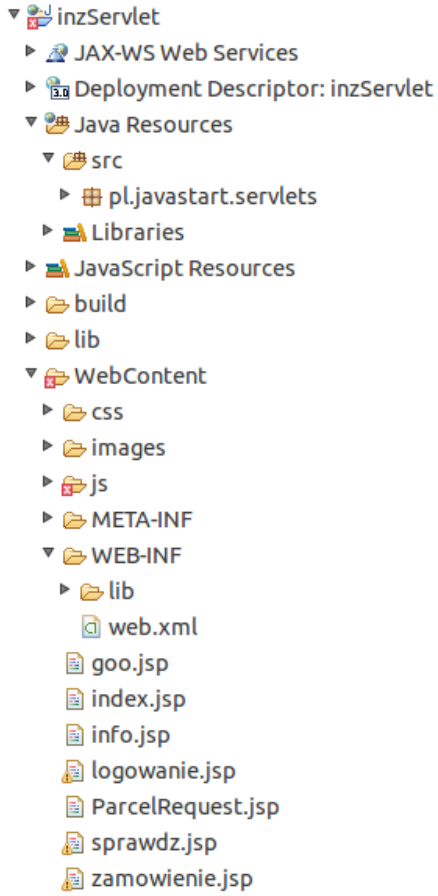
W niniejszej pracy skorzystano z możliwości rozszerzenie języka Java o funkcje serwera WWW. Zadaniem Servletu (apletu Javy) było obsłużenie klientów firmy kurierskiej, którzy wpisując numer swojej przesyłki mogli sprawdzić gdzie się ona aktualnie znajduje. Oprócz tej funkcji Servlet też jest pośrednikiem między aplikacją mobilną, a bazą danych.

Rozpoczęcie pracy z Servletem wymagało zainstalowania dodatków do środowiska Eclipse:

- Eclipse Java EE Developer Tools
- Eclipse Java Web Developer Tools
- Eclipse Web Developer Tools
- JST Server Adapters
- JST Server Adapters Extensions

Po zainstalowaniu dodatków należało skonfigurować środowisko. W tym celu należało stworzyć nowy lokalny serwer. Autor skorzystał z popularnego serwera Apache Tomcat w wersji 7.0. Serwer nasłuchuje połączeń pod adresem localhost na porcie 8080.

Po skonfigurowaniu środowiska można było przystąpić do utworzenia nowego dynamicznego projektu sieciowego (`Dynamic Web Project`). W strukturze projektu najważniejsze są dwa foldery [rys. 3.5]: pierwszy zawierający w sobie klasy Javy, oraz drugi definiujący zachowanie i wygląd stron Servletu. W tymże folderze znajduje się plik „web.xml", w którym definiuje się zachowanie serwera w zależności od tego jaki adres został wpisany w przeglądarce.



Rysunek 3.5: Struktura plików Servletu

Na poniższym listingu widać ustawienie strony startowej (`<welcome-file-list>`). Gdy w polu adresy wpisany jest `http://localhost:8080/inzServlet` uruchamia się strona główna aplikacji webowej – Index.jsp. W następnych liniach ładowana jest klasa Javy (`pl.javastart.servlets.Sprawdz`) do Servletu pod nazwą `Sprawdz` i następuje zmapowanie tej klasy pod adres `http://localhost:8080/inzServlet/sprawdz`. Pokazany tu sposób dotyczy wszystkich używanych przez Servlet klas. W podobny sposób można zadeklarować użycie pliku *.jsp (różnica polega na zamianie identyfikatora `<servlet-class>` na `<jsp-file>`).

```
1 <welcome-file-list>
2   <welcome-file>index.jsp</welcome-file>
3 </welcome-file-list>
4 <servlet>
5   <servlet-name>Sprawdz</servlet-name>
6   <servlet-class>pl.javastart.servlets.Sprawdz</servlet-class>
```

```
</servlet>
<servlet-mapping>
  <servlet-name>Sprawdz</servlet-name>
  <url-pattern>/sprawdz</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>Index</servlet-name>
  <jsp-file>/index.jsp</jsp-file>
</servlet>
<servlet-mapping>
  <servlet-name>Index</servlet-name>
  <url-pattern>/index</url-pattern>
</servlet-mapping>
```

Listing 3.6: Fragment pliku web.xml

Każda klasa Javy, która rozszerza Servlet posiada dwie istotne metody do obsługi stron serwera – `doPost` i `doGet`. Obydwie przyjmują argumenty typu `HttpServletRequest` i `HttpServletResponse`. Metody mają zapewniać komunikację pomiędzy serwerem i klientem. Argument `HttpServletRequest` zawiera żądanie klienta do wykonania na Servlecie, natomiast `HttpServletResponse` jest odpowiedzią serwera na żądanie klienta. Wspomniana metoda `doGet` obsługuje żądania jakie zostały przesłane w nagłówku http – pobiera parametry i przetwarza je. Metoda `doPost` również obsługuje żądanie powstałe z uzupełnienia formularza.

Ważną klasą w projekcie jest klasa obsługująca bazę danych. Klasa do obsługi bazy danych musi składać się z następujących poleceń: załadowanie sterownika bazy danych, nawiązanie połączenia z bazą, pobranie/zapisanie danych, zamknięcie połączenia. Przykładem obsługi bazy danych jest poniższy listing [listing 3.7]. Przedstawiona metoda sprawdza czy dany kurier istnieje w bazie. Sprawdzenie istnienia kuriera realizowane jest przez zapytanie `"select * from deli.deliverer where id=" + id`, gdzie id jest identyfikatorem kuriera. Po wykonaniu zapytania pod `ResultSet` zapisywane są dane pobrane z bazy danych, z nich wyszukiwane są potrzebne informacje, czyli id kuriera oraz jego status aktywności. `Id` pobierane jest tylko w celu sprawdzenia czy w bazie istnieje kurier, niestety nie ma metody dla `ResultSet` która sprawdzałaby czy wynik zapytania jest pusty. Po uzyskaniu wyniku, że kurier istnieje sprawdzana jest jego aktywność. Gdy kurier jest nieaktywny a funkcja była wywoływana w celu zalogowania do systemu przygotowywane jest zapytanie o zaktualizowanie danych dla tego kuriera i ustawieniu jego aktywności na stan aktywny (`true`). Natomiast gdy aktywność id kuriera ma wartość logiczną `true`, a kurier chce się zalogować (metoda została wykonana dla zalogowania do systemu) metoda zwraca ciąg „busy" – informuje o zajętości id. W momencie wylogowywania kuriera (kiedy `activ == true`) a `logout` wynosi 0 – to przygotowywana jest komenda aktualizująca bazę danych i pod wskazaną krotkę o numerze id zapisywana jest wartość 0 dla parametru `activ`.

```
Class.forName("com.mysql.jdbc.Driver");
Connection connection = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/deli", "root", "sun5flower");
Statement statement = connection.createStatement();
stringSelect = "select * from deli.deliverer where id=" + id;
ResultSet resultSet = statement.executeQuery(stringSelect);

while (resultSet.next()) {
  delivererId = resultSet.getInt("id");
  delivererActiv = resultSet.getBoolean("activ");
}
if (delivererId > 0) {
  if (delivererActiv == false) {
    sqlInsert = " update deli.deliverer set activ=1 where id=" + id;
    statement.executeUpdate(sqlInsert);
    connection.close();
    statement.close();
    return "logIn";
  } else if (delivererActiv == true) {
    if (logout.equals("0")) {
      sqlInsert = " update deli.deliverer set activ=0 where id=" + id;
      statement.executeUpdate(sqlInsert);
      connection.close();
      statement.close();
      return "logOut";
    } else if (logout.equals("1")) {
      connection.close();
      statement.close();
      return "busy";
```

```
30      }
31    }
32  } else
33    return "false";
```

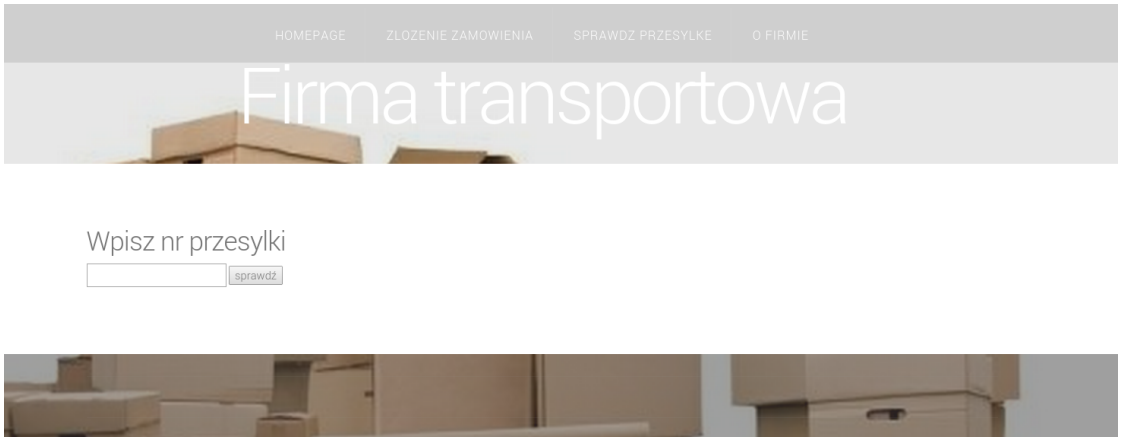
Listing 3.7: Połączenia z bazą danych na przykładzie metody sprawdzającej istnienie kuriera oraz jego stan używanej przez aplikację mobilną

Projekt dążył do stworzenia przyjaznego dla użytkownika – klienta widoku lokalizacji przesyłki. Do osiągnięcia tego celu użyto zmodyfikowanego przez autora darmowego szablonu pobranego z internetu [11]. W zakładce `SPRAWDZ PRZESYLKE` wpisywany jest numer przesyłki jaką klient chce sprawdzić [rys. 3.6(a)]. Wpisany ciąg znaków jest sprawdzany pod kątem poprawności. Sprawdzane jest czy ciąg w formularzu jest wartością liczbową i czy istnieje taka przesyłka. Gdy w bazie nie istnieje przesyłka pojawia się odpowiedni komunikat informujący o nieistniejącej przesyłce [rys. 3.6(c)]. Gdy użytkownik wprowadzi poprawny numer przesyłki zostaną wyświetlone takie informacje jak: nadawca, odbiorca, czas nadania i czas odbioru oraz ostanie sczytane położenie przesyłki. Ponadto wyświetlana jest mapa pokazująca jaką trasę pokona przesyłka - znaczniki A – B. Na mapie dodatkowo pokazany jest znacznik, który wyznacza pozycję kuriera z daną przesyłką – znacznik „kurier".

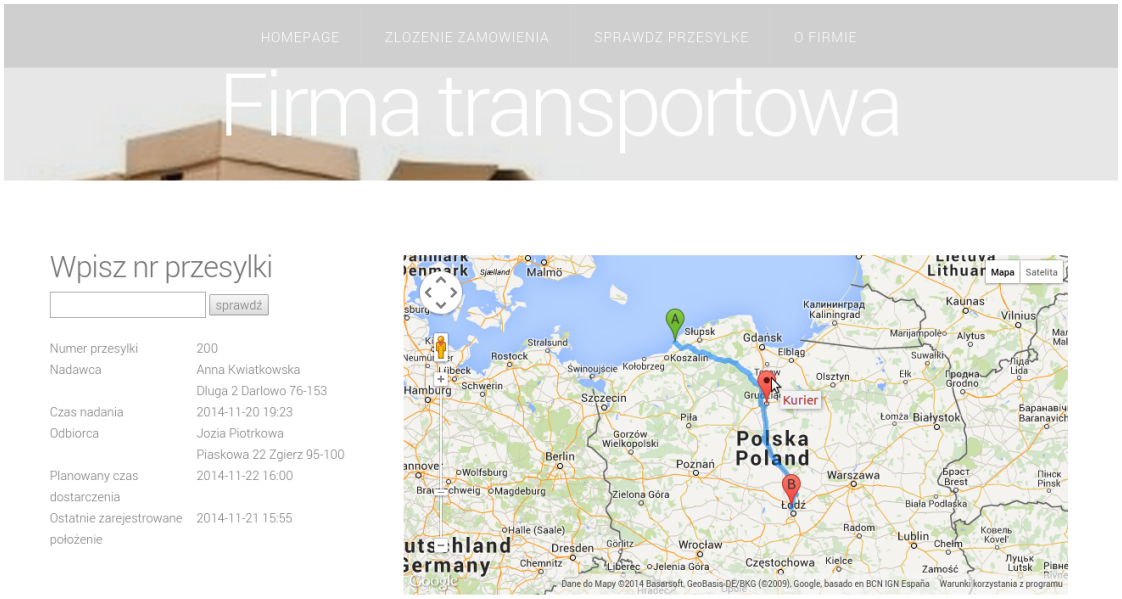
Istotną częścią projektu było zaprojektowanie strony, na której klient może sprawdzać status swojej przesyłki. Zrealizowane jest to poprzez utworzenie pliku *.jsp, i przypisanie jej adresu http. Ogólny zarys pliku sprawdz.jsp jest pobrany z szablonu [11], autor zmienił jedynie ciało pliku.

Z formularza [rys. 3.6(a)] pobierany jest numer poszukiwanej przesyłki [listing 3.8 linie 1-4] i ustawiany jako post formularza i przesyłany do klasy Servletu. W tej klasie po pobraniu danych z bazy danych zwracane są wartości dotyczące istnienia przesyłki o wskazanym id. Najpierw sprawdzane jest, we fragmencie kodu Javy w instrukcji warunkowej, czy przesyłka istnieje (`isParcelExists`). Gdy przesyłka nie zostanie odnaleziona w bazie lub wpisany ciąg w formularzu jest niepoprawny wyświetlana jest wiadomość [listing 3.8 linia 43] dotycząca błędnego id. Natomiast gdy wpisany id jest bezbłędny w tabeli [rys. 3.6(b)] zostają wypisane dane dotyczące przesyłki oraz mapa. Linia 39 listingu 3.8 jest miejscem użycia JavaScript'owego kodu map Google. W danych wejściowych zmienna `msg` oznacza wiadomość dotyczącą niepoprawnie wpisanego numeru przesyłki, lub jeśli przesyłka została poprawnie wpisana, ale jej ostatnie położenie jest zarejestrowane w centrali to w tabeli w wierszu „Ostatnie zarejestrowane położenie" pojawia się dodatkowa informacje o pobycie przesyłki w centrali.

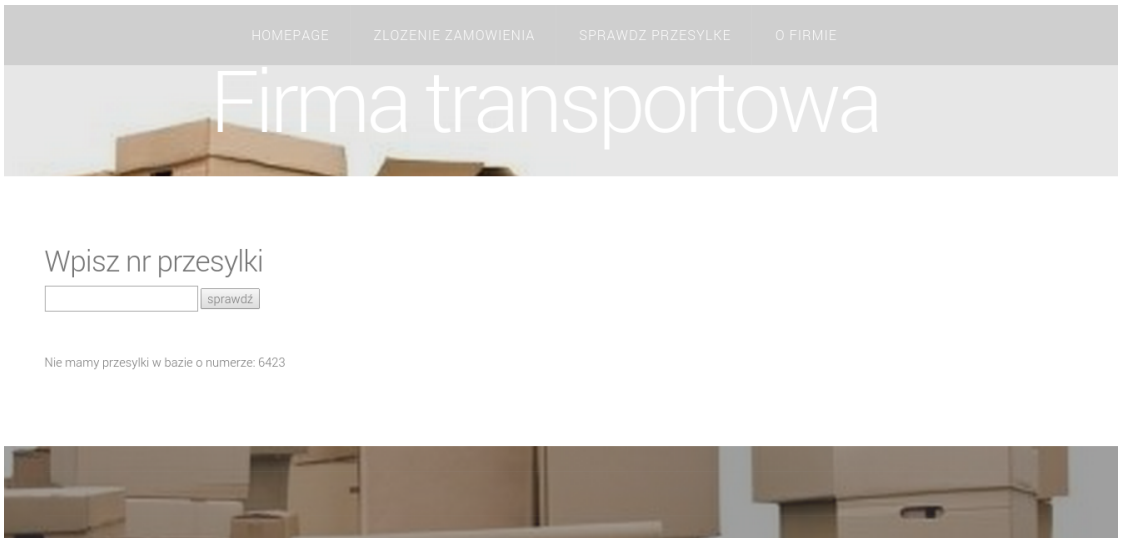
```
1  <h2>Wpisz nr przesylki</h2>
2    <form id="formularz" method="post" action="">
3    <input type="text" name="nr" /> <input type="submit" value="sprawdz" />
4  </form>
5     :
6  <%
7  if (request.getAttribute("isParcelExists") != null &&
8  request.getAttribute("isParcelExists").equals(true)) {
9  %>
10 <table style="width:100%">
11   <tr>
12     <td><% out.print("Numer przesylki"); %></td>
13     <td>${id}</td>
14   </tr>
15   <tr>
16     <td><% out.print("Nadawca"); %></td>
17     <td>${regUserName}<br> ${regUserAddr} </td>
18   </tr>
19   <tr>
20     <td><% out.print("Czas nadania"); %></td>
21     <td>${timeSend}</td>
22   </tr>
23   <tr>
24     <td><% out.print("Odbiorca"); %></td>
25     <td>${userName}<br> ${userAddr} </td>
26   </tr>
27   <tr>
28     <td><% out.print("Planowany czas"); %><br> <% out.print("dostarczenia"); %>
29     </td>
30     <td>${timeDelivery}</td>
31   </tr>
32   <tr>
33     <td><% out.print("Ostatnie zarejestrowane"); %><br>
34       <% out.print("polozenie"); %></td>
35     <td>${lastTime}<br>${msg}</td>
```



(a)



(b)



(c)

Rysunek 3.6: Okno aplikacji webowej i jego widoki: (a) Widok strony internetowej `SPRAWDZ` serwisu z zachętą do wprowadzenia numeru przesyłki do sprawdzenia; (b) Widok strony internetowej `SPRAWDZ` z wyznaczoną trasą dla przykładowej przesyłki; (c) Przykład odpowiedzi serwisu na nieprawidłowe wpisanie numeru przesyłki

```
36    </tr>
37  </table>
38     :
39  <div id="mapka"> </div>
40     :
41  <%
42  }
43  else {
44  %>
45  <br> ${msg} ${id}
46  <%
47  }
48  %>
```

Listing 3.8: Ciało pliku JavaServlet Pages - sprawdz.jsp

Ustawianie parametrów umieszczonych w ${...} zrealizowano poprzez klasę rozszerzającą Servlet. Na listingu 3.9 przedstawiono metodę `doPost()` klasy `Sprawdz.java`, która odpowiada za uzupełnianie pliku JSP `sprawdz.jsp` danymi dotyczącymi przesyłki, które są wyświetlane w widoku strony dla klienta. Klasa `Sprawdz` tworzy nowy obiekt klasy `SearchParcel` wywołując ją z argumentem id przesyłki. Obiekt klasy `SearchParcel` odczytuje bazę danych i przypisuje wartości zmiennym (listing 3.11). Następnie sprawdzane jest w instrukcji warunkowej if czy taka przesyłka istnieje i wprowadzone znaki są liczbą całkowitą. Jeśli nie istnieje to w zależności od tego jaki błąd wystąpił przygotowywana jest wiadomość zwrotna dotycząca tego błędu. Jeżeli jednak przesyłka istnieje ustawiane są parametry zwrotne `req.setAttribute("...",...)`. Warto zwrócić uwagę na to, że pierwszy argument `setAttribute` musi być zgodny z oczekiwaną nazwą zmiennej w pliku JSP, w inny wypadku nie zostanie przypisana wartość z klasy Javy do JSP. Na przykład argumenty z listingu 3.8 z linii 45 muszą się zgadzać argument z listingu 3.9 z linii 30-32.

```
1  protected void doPost(HttpServletRequest req, HttpServletResponse resp) {
2    String id = req.getParameter("nr");
3    boolean isParcelExists = false;
4    RequestDispatcher view = req.getRequestDispatcher("/sprawdz.jsp");
5    if (id != null && (Sth.isInteger(id)) == true) {
6      SearchParcel searchParcel = new SearchParcel(Integer.parseInt(id));
7      if (searchParcel.getIsParcelExists() == true) {
8        isParcelExists = true;
9        req.setAttribute("id", id);
10       req.setAttribute("lat", searchParcel.getDelivererLatitude());
11       req.setAttribute("lon", searchParcel.getDelivererLongitude());
12       req.setAttribute("regUserName", searchParcel.getRegisteredUserNameUser());
13       req.setAttribute("regUserAddr", searchParcel.getRegisteredUserStreetUser() +
14           " " + searchParcel.getRegisteredUserCityUser() + " " +
15       searchParcel.getRegisteredUserCityCodeUser());
16       req.setAttribute("timeSend", searchParcel.getParcelSendTime().substring(0, 16));
17       req.setAttribute("userName", searchParcel.getParcelAddresseeName());
18       req.setAttribute("userAddr", searchParcel.getParcelAddresseeStreet() + " " +
19           searchParcel.getParcelAddresseeCity() + " " +
20           searchParcel.getParcelAddresseeCityCode());
21       req.setAttribute("timeDelivery", searchParcel .getParcelDeliveryTime()
22           .substring(0, 16));
23       req.setAttribute("lastTime", searchParcel.getParcelTimePos().substring(0, 16));
24       if (searchParcel.getDelivererId() > 999000) {
25         searchParcel.selectBase(searchParcel.getDelivererId());
26         req.setAttribute("msg", "Przesylka w bazie: "
27             + searchParcel.getCentreNameCentre());
28       }
29     } else {
30       req.setAttribute("msg", "Nie mamy
31       przesylki w bazie o numerze: ");
32       req.setAttribute("id", id);
33   }
34   if (id == null || (Sth.isInteger(id)) == false)
35     req.setAttribute("msg", "Prosze podac poprawny numer przesylki");
36   req.setAttribute("isParcelExists", isParcelExists);
37   req.removeAttribute("nr");
38   view.forward(req, resp);
39 };
```

Listing 3.9: Fragment klasy Sprawdz.Java metoda doPost()

Na podstawie źródła [6] autor utworzył kod, który wyświetla mapę ze znacznikami – markerami. Z danych metody post pobierane są wartości lat i lon (szerokość i długość geograficzna) dla kuriera oraz wartości `regUserAddr` i `userAddr` oznaczające adres nadawcy i odbiorcy. W skrypcie tworzony jest nowy obiekt klasy `DirectionsRenderer`, który jest odpowiedzialny za wypełnienie (renderowanie) wyświetlanej mapy oraz tworzony jest nowy obiekt `DirectionsService`, który wylicza trasę pomiędzy punktami. Do obiektu `DirectionsRenderer` przypisywane jest umiejscowienie elementu w układzie strony przez pobranie jego id oraz możliwe jest ustawienie parametrów mapy. Kolejnym krokiem jest utworzenie znacznika na pozycji kuriera i dodanie go do mapy. Ostatnim etapem jest wyliczenie trasy pomiędzy dwoma punktami: adresem nadawcy i adresem odbiorcy.

```
<script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
<script>
  var lat = "${lat}";
  var lon = "${lon}";
  var start = "${regUserAddr}";
  var end = "${userAddr}";
  var directionsRenderer = new google.maps.DirectionsRenderer();
  var directionsService = new google.maps.DirectionsService();
  var map;

  function initialize() {
    map = new google.maps.Map(document.getElementById('mapka'));
    directionsRenderer.setMap(map);
    new google.maps.Marker({
      position : new google.maps.LatLng(lat, lon),
      map : map,
      title : "Kurier"
    });
    var directionsRequest = {
      origin : start,
      destination : end,
      travelMode : google.maps.TravelMode.DRIVING
    };
    directionsService.route(directionsRequest, function(directionsResult,
    directionsStatus) {
        if (directionsStatus == google.maps.DirectionsStatus.OK) {
          directionsRenderer.setDirections(directionsResult);
      }
    });
  }
  google.maps.event.addDomListener(window, 'load', initialize);
</script>
```

Listing 3.10: Kod JavaScript'owy pobierający mapę z serwera Google

Klasa `Sprawdz.java` obsługująca `sprawdz.jsp` tworzy nowy obiekt klasy `SearchParcel` i podbiera od niego potrzebne dane do wyświetlenia na stronie i przesyła je metodą `doPost`. Klasa `SearchParcel.java` przedstawiona jest na listingu 3.11. Klasa otwiera połączenie z bazą danych, następnie wykonując zapytania SQL pobiera dane dotyczące przesyłki, a posiadając już id kuriera, który przewozi tę przesyłkę, pobiera jego położenie oraz informacje o nadawcy (zarejestrowanym użytkowniku). Ostatnim krokiem jest zamknięcie połączenia z bazą danych.

```
public SearchParcel(int _nrParcel) {
  selectFromDB(_nrParcel);
}
void selectFromDB(int nrParcel) {
  String strSelect = "";
  Class.forName("com.mysql.jdbc.Driver");
  Connection connection = DriverManager.getConnection(
      "jdbc:mysql://localhost:3306/deli", "root", "sun5flower");
  Statement statement = connection.createStatement();
  strSelect = "select * from deli.parcel where id=" + nrParcel;
  ResultSet resultSet = statement.executeQuery(strSelect);
  while (resultSet.next()) {
    isParcelExists = true;
    parcelId = resultSet.getInt("id");
    parcelFromUserId = resultSet.getInt("fromUserId");
    parcelAddresseeName = resultSet.getString("addresseeName");req.setAttribute("
    regUserName",
        searchParcel.getRegisteredUserNameUser());
    parcelAddresseeStreet = resultSet.getString("addresseeStreet");
    parcelAddresseeCity = resultSet.getString("addresseeCity");
```

```
      parcelAddresseeCityCode = resultSet
          .getString("addresseeCityCode");
      parcelSendTime = resultSet.getString("sendTime");
      parcelTimePos = resultSet.getString("timePos");
      parcelDeliveryTime = resultSet.getString("deliveryTime");
      parcelDeliverer = resultSet.getInt("deliverer");
    }
    if (isParcelExists == true) {
      strSelect = "select * from deli.deliverer where id="
          + parcelDeliverer;
      resultSet = statement.executeQuery(strSelect);
      while (resultSet.next()) {
        delivererId = resultSet.getInt("id");
        delivererLatitude = resultSet.getDouble("latitude");
        delivererLongitude = resultSet.getDouble("longitude");
        delivererTimePos = resultSet.getString("timePos");
      }
      strSelect = "select * from deli.registeredUser where id="
          + parcelFromUserId;
      resultSet = statement.executeQuery(strSelect);
      while (resultSet.next()) {
        registeredUserId = resultSet.getInt("id");
        registeredUserNameUser = resultSet.getString("nameUser");
        registeredUserStreetUser = resultSet
            .getString("streetUser");
        registeredUserCityUser = resultSet.getString("cityUser");
        registeredUserCityCodeUser = resultSet
            .getString("cityCodeUser");
      }
    }
    connection.close();
    statement.close();
}
```

Listing 3.11: Klasa SearchParcel.java przedstawiona metoda selectFromDB() która reprezentuje połączenie z bazą danych oraz pobiera wszystkie dane o przesyłce

komentarz:Wyznaczanie przybliżonego czasu dostawy zrealizowano korzystając (to i więcej już o Serwlecie nie trzeba) **??**

Estymację czasu dostarczenia przesyłki zrealizowano korzystając z usługi Google Distance Matrix API, z której dostarczana jest odległość pomiędzy punktem nadania i odbioru przesyłki. Wyznaczanie czasu dostarczenia wprowadzono równolegle z zapisem przesyłki do bazy danych. Autor nie mógł w inny sposób niż przedstawiony tutaj [listning 3.12], tego wykonać z braku rzeczywistych próbek w postaci prawdziwy danych.

```
public String wyliczCzasDostarczenia(Date sendTime) {
  int days = 1;
  // zamiana inta na date
  /* String datee = */new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
      .format(new Date(czasPomiedzyDwomaPunktami * 1000L));
  //
  // pobranie godziny z daty
  Calendar calendar = GregorianCalendar.getInstance();
  calendar.setTime(sendTime);
  //
  // ustawienie dni dostarczania
  // godzina nadania po 18
  if (calendar.get(Calendar.HOUR_OF_DAY) > 18) {
    calendar.set(Calendar.HOUR_OF_DAY, 11);
    days++;
  }
  // godzina nadania jest pomiedzy 0 a 12
  if (calendar.get(Calendar.HOUR_OF_DAY) > 0
      && calendar.get(Calendar.HOUR_OF_DAY) < 12) {
    calendar.set(Calendar.HOUR_OF_DAY, 15);
  }
  // czas pomiedzy dwoma puntami mniejszy niz 1.5h
  if (czasPomiedzyDwomaPunktami < 1000) {
    days--;
  }
  // czas pomiedzy punktami wiecej niz 6,5h
  if (czasPomiedzyDwomaPunktami > 20000) {
    days++;
```

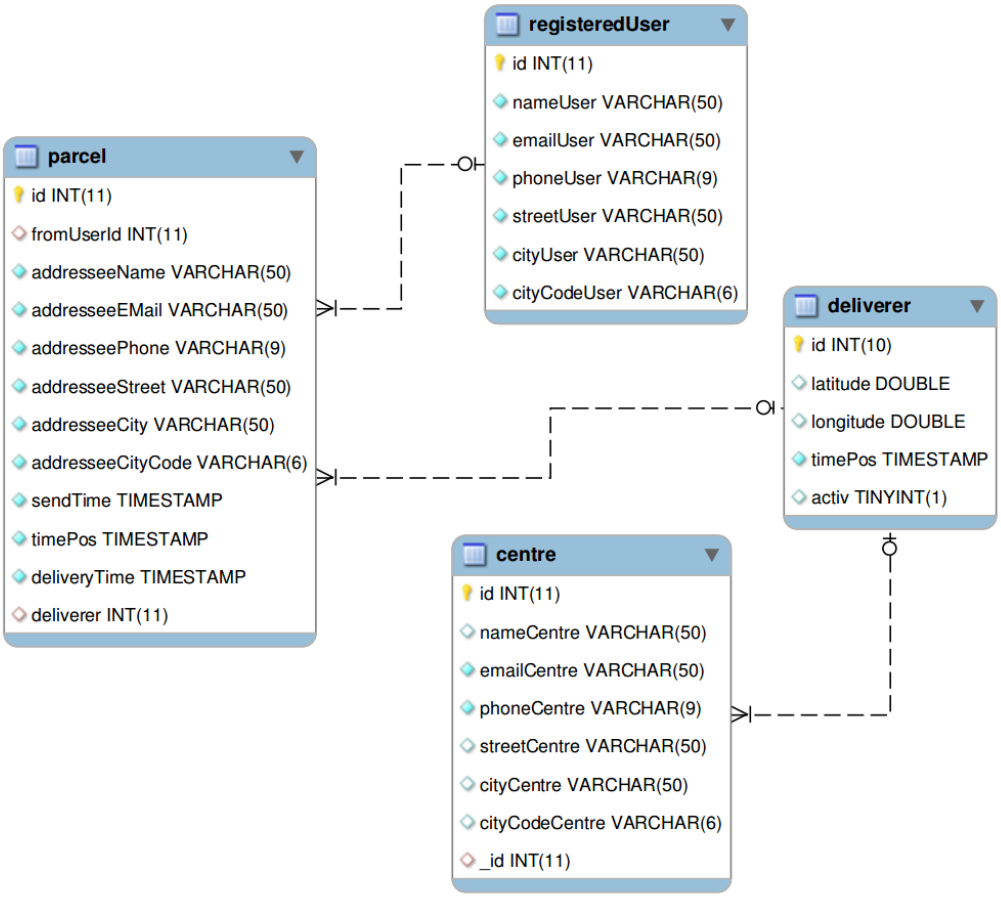
```
29      }
30      calendar.add(Calendar.DAY_OF_MONTH, days);
31      return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(calendar
32              .getTime());
33      }
```

Listing 3.12: Metoda wyliczCzasDostarczenia z klasy DistanceMatrix

W podrozdziale Servlet nie pokazano obsługi wyjątków, a przedstawione listingi są kluczowe do zrozumienia działania Servletu. Układ graficzny aplikacji webowej został pobrany ze strony z darmowymi szablonami *http://templated.co/linear*.

3.3 Baza danych

W przedstawionym projekcie konieczne było utworzenie bazy danych, która przechowuje dane dotyczące kurierów, przesyłek, klientów, centrali kurierskich. W tym celu stworzono cztery tabele: `parcel`, `deliverer`, `registeredUser` i `centre`.



Rysunek 3.7: Diagram ERD zaprojektowanej bazy danych

Na rysunku 3.7 przedstawiono diagram ERD pokazujący tabele oraz połączenia pomiędzy nimi. Każda z tabel ma swój klucz główny (`id`). Tabela `parcel` (przesyłka) zawiera w sobie dwa klucze obce(`foreign key`) odwołujące się do `deliverer.id` – klucza głównego tabeli zawierającej informacje o kurierach oraz `registeredUser.id` - klucz główny do tabeli użytkownika zarejestrowanego, czyli tego który złożył zamówienie na dostarczenie przesyłki. Tabela `centre`, która ma zawierać wpisy dotyczące nazwy, ulicy oraz miasta zawiera w sobie klucz obcy do tabeli `deliverer`. Wybrano takie rozwiązanie ponieważ w tabeli `deliverer` są podawane długość i szerokość geograficzna, które w łatwy sposób powiązano z tabelą `centre`.

Rozdział 4

Przygotowanie i uruchomienie aplikacji

Aby utworzyć opisywany w tej pracy projekt należy zacząć od instalacji IDE Eclipse. Najprościej jest zacząć od instalacji Eclipse ADT z dodatkiem Android SDK [4], po jego instalacji w widoku Eclipse pojawi się ikona „Android SDK Manager". Po załadowaniu się menedżera należy wybrać przynajmniej SDK Tools, SDK Platform-tools, SDK Build-tools (najwyższą dostępną wersję), oraz w folderze z ostatnią wersją systemu Android X.X zaznaczyć SDK Platform i obraz emulatora systemu Android (ARM EABI v7a System Image). Ponadto na potrzeby tego projektu konieczne jest zainstalowanie Google Repository i Google Play Services z folderu Extras.

Następnym krokiem jest zainstalowanie serwera Apache Tomcat w IDE Eclipse. Dodanie nowe serwera rozpoczyna się od otworzenia zakładki `Window>Preferences>Server>Runtine Enviroment`, gdzie należy dodać (`add`) Apache Tomcat również w najnowszej wersji. Spowoduje to dodanie serwera do bieżącej konfiguracji IDE.

Autor korzysta z systemu operacyjnego Linux dystrybucja Ubuntu, więc zostanie opisana instalacja silnika bazy danych *. . . Opisać* **??**

Jeśli w środowisku deweloperskim nie ma dodanych odpowiednich perspektyw, należy je dodać (Open Perspective). Przydatne będą perspektywy „Java" oraz „SQL Explorer". Po spełnieniu wszystkich wymienionych kroków można zaimportować projekty. Import projektów odbywa się poprzez wybranie `Import>Existing Project Into Workspace` z menu `File`.

Rozdział 5

Możliwość rozwinięcia w przyszłości

Aplikację można „podpiąć” pod prawdziwe urządzenia jakie posiadają kurierzy – te na których się człowiek podpisuje – ale konieczne będzie zrefakturyzowanie(?)/zmianie kodu pod system, który mają tam zainstalowany. Fajnie by było to wrzucić na prawdziwe tablety, można by sprzedawać/zarobić. Ogólnie koszt takiego urządzenia to byłoby tablet/telefon + wycena za program. Normalnie kurierzy używają kolektorów danych.

Nie tylko GPS do lokalizacji, bo także odbicia na czytnikach u kurierów Projekt opiera się

Rozdział 6

Wnioski i podsumowanie

??

Bibliografia

[1] Android. http://developer.android.com. [Online; dostęp 16.11.2014].

[2] Apache HTTP serwer. http://en.wikipedia.org/wiki/Apache_HTTP_Server. [Online; dostęp 16.11.2014].

[3] dokumentacj Javy. https://docs.oracle.com/javase/specs/jls/se8/html/index.html. [Online; dostęp 16.11.2014].

[4] eclipse with adnroid sdk. https://developer.android.com/sdk/index.html. [Online; dostęp 11.11.2014].

[5] Google. http://developer.google.com. [Online; dostęp 16.11.2014].

[6] Google. https://developers.google.com/maps/documentation/webservices/. [Online; dostęp 16.11.2014].

[7] Java MVC. http://www.oracle.com/technetwork/articles/javase/index-142890.html. [Online; dostęp 26.11.2014].

[8] Java MVC. http://www.javatpoint.com/model-1-and-model-2-mvc-architecture. [Online; dostęp 26.11.2014].

[9] JavaScript wiki. http://pl.wikipedia.org/wiki/JavaScript. [Online; dostęp 16.11.2014].

[10] MySQL. http://dev.mysql.com/doc/refman/5.6/en/introduction.html. [Online; dostęp 26.11.2014].

[11] szablon. http://templated.co/linear. [Online; dostęp 16.11.2014].

[12] xml wiki. http://pl.wikipedia.org/wiki/XML. [Online; dostęp 16.11.2014].

Spis rysunków

Listings