

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI

---

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

**PROJEKT INŻYNIERSKI**

System monitoringu lokalizacji przesyłek  
kurierskich

English title ŁŁAŚĆŻÓ łaścżżóę E

AUTOR:  
Monika Strachowska

PROWADZĄCY PROJEKT:  
dr hab. inz. Imie Nazwisko Prof. PWr, I-6

OCENA PROJEKTU:

# Spis treści

<b>1 Wstęp i cel pracy</b>	<b>2</b>
<b>2 Rozwiązanie - prezentacja wyników</b>	<b>3</b>
2.1 Aplikacja na system Android . . . . .	4
2.2 Serwer . . . . .	8
2.3 Baza danych . . . . .	15
<b>3 Przygotowanie i uruchomienie aplikacji</b>	<b>16</b>
<b>4 Wykorzystane technologie</b>	<b>17</b>
4.1 Java . . . . .	17
4.2 Wzorzec architektoniczny - MVC . . . . .	18
4.3 Servlet . . . . .	19
4.4 JSP - Java . . . . .	19
4.5 Technologie internetowe . . . . .	19
4.6 db - MySQL . . . . .	20
4.7 Android . . . . .	21
4.8 Google apps - mapy . . . . .	22
<b>5 Możliwość rozwinięcia w przyszłości</b>	<b>23</b>
<b>6 Wnioski i podsumowanie</b>	<b>24</b>
<b>Bibliografia</b>	<b>24</b>

# Rozdział 1

## Wstęp i cel pracy

Współcześnie coraz więcej osób korzysta z możliwości zakupów przez internet, 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. Oprócz wyżej wymienionych zakupów istotnym towarem przewożonym są dokumenty, często szybko potrzebne. Z tych powodów ludzie zamawiający usługi kurierskie chcieliby dostać jak najwyższą jakość. Zwiększenie jakości tej usługi może nastąpić poprzez skrócenie czasu dostarczenia przesyłki (co fizycznie już jest nie osiągalne), tańszy 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 taką dodatkową funkcjonalność. Z punktu widzenia klienta odczuwany jest komfort informacji, gdzie jest przesyłka, dzięki temu klient może zaplanować sobie dzień, w którym nastąpi dostarczenie.

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 odbicia przesyłki. Ponadto praca próbuje rozwiązać problem jaki istnieje w estymacji czasu dostarczaniu przesyłki do adresata, estymacja czasu dostarczenia jest bardzo niedokładna (ogólna) lub jej nie ma.

*Projekt został zrealizowany z wykorzystaniem takich technologii jak: język programowania Java, system operacyjny Android, baza danych MySQL, Google Apps. Firmy kurierskie mają najprawdopodobniej system „windows ce/mobile” na swoich urządzenia. Ja ze względu na brak takiego urządzenia (mobilnego z windowsem) zrealizuje zadanie na androidzie.*

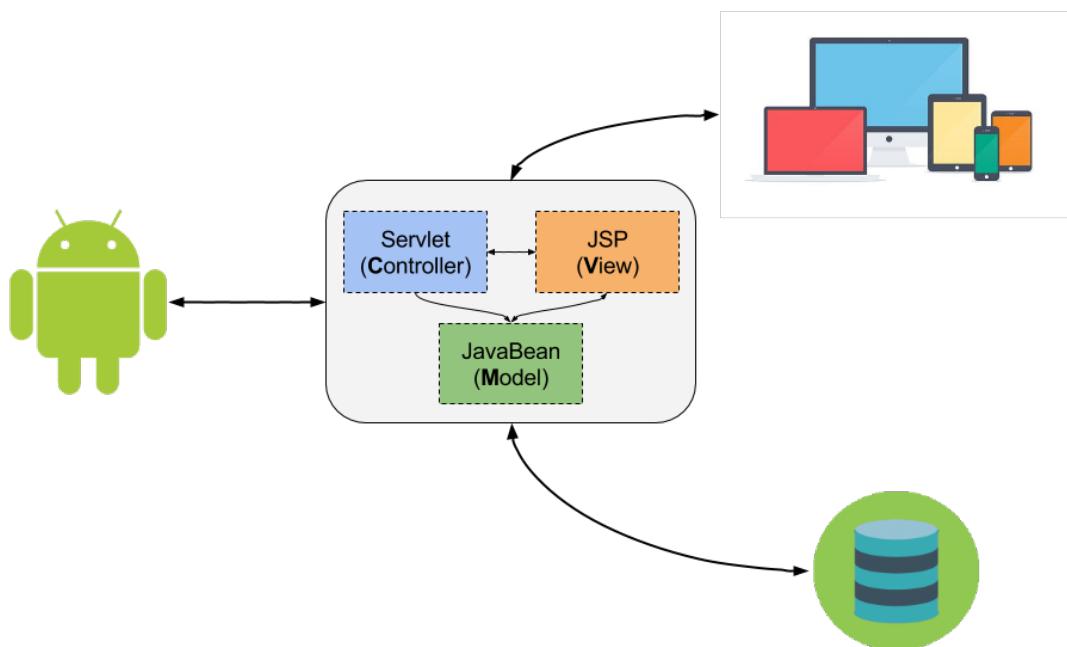


Rysunek 1.1: podpisisi

## Rozdział 2

# Rozwiązanie - prezentacja wyników

Zrealizowana aplikacja składa się z dwóch części, tj. aplikacji na androida oraz serwera www. Serwer, czyli Servlet (aplet Javy) posiada połączenie z bazą danych, w której przechowywane są informacje o przesyłce, kurierach i klientach. Aplikacja mobilna również posiada połączenie z bazą danych, jednakże połączenie to zrealizowano pośrednio. Aplikacja poprzez zapytania i odpowiedzi z serwera uzyskuje informacje o stanie bazy danych.



Rysunek 2.1: Struktura systemu

Ideę systemu zaprezentowano na rysunku 2.1. Aplikację zrealizowano w środowisku programistycznym Eclipse.

*komentarz: dopisać pierdół wstępowych*

## 2.1 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 doinstalowania pluginu ADT (Android Development Kit) do programu Eclipse. Na ADT składają się elementy Android SDK, gdzie SDK to Software Development Kit. Android SDK zawiera w sobie takie elementy jak Tools - służą 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, emulator. Natomiast Platform Tools 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. 2.2], w której jest podział na klasy zawierające logikę aplikacji (scr), pliki generowane przez kompilator (gen), folder na pliki z zasobami (assets), pliki binarne(bin), dodatkowe biblioteki (libs) oraz folder na zasoby(res), którego odróżnia od assets to, że są generowane do pliku R.java (nie trzeba podawać lokalizacji zasobów tylko jego nazwy). W katalogu res również ustalana jest konfiguracja systemu - „AndroidManifest.xml”, layout (wygląd i ustawienie elementów na ekranie systemu Android), w podfolderach Drawable pliki graficzne, w podfolderach values ciągi znaków(stringi, kolory) i ustawienia oraz w katalogu menu, w którym ustawiane są dostępne opcje menu.

Projektowanie aplikacji Android zaczyna się od ustawienia layoutu aplikacji (/res/layout/activity\_main.xml). Za projektowany przez autora layout jest prosty i przejrzysty, ponieważ ma wykonywać bardzo podstawowe funkcje [rys. 2.3]. 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 hardware'owe przyciski nie wyłączają aplikacji, jedynie ją minimalizują. Takie właściwości zostały stworzone z myślą o tym, aby kurier podczas używania aplikacji tylko w świadomym sposobie 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. 2.4(a)]. Oprócz wyżej wymienionych zachowań aplikacji, w pasku statusu na telefonie wyświetlana jest ikona podczas włączonej aplikacji a także w powiadomieniach [rys. 2.4(b)].

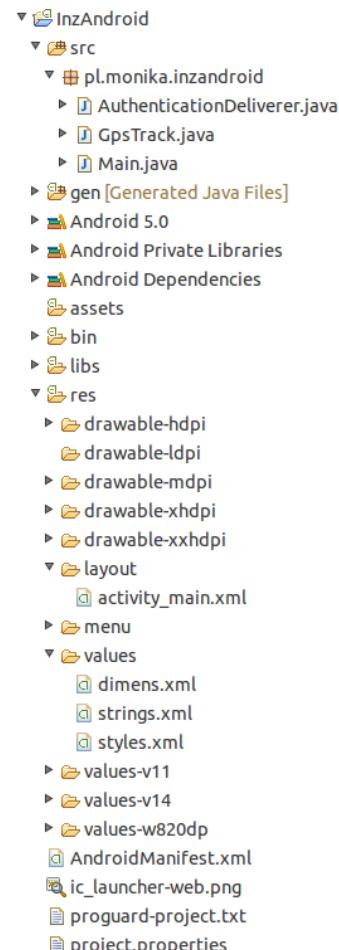
Kolejnym krokiem było nadanie odpowiednich uprawnień aplikacji, do tego służy plik „AndroidManifest.xml”. Aplikacja zostały przyznane uprawnienia do sprawdzania statusu sieci komórkowej oraz WiFi, sprawdzania statusu sygnału GPS, a także do korzystania z wyżej wymienionych [listing 2.1].

```

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

Listing 2.1: 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



Rysunek 2.2: Struktura programu aplikacji Android



Rysunek 2.3: Ekrany Androida

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 2.2].

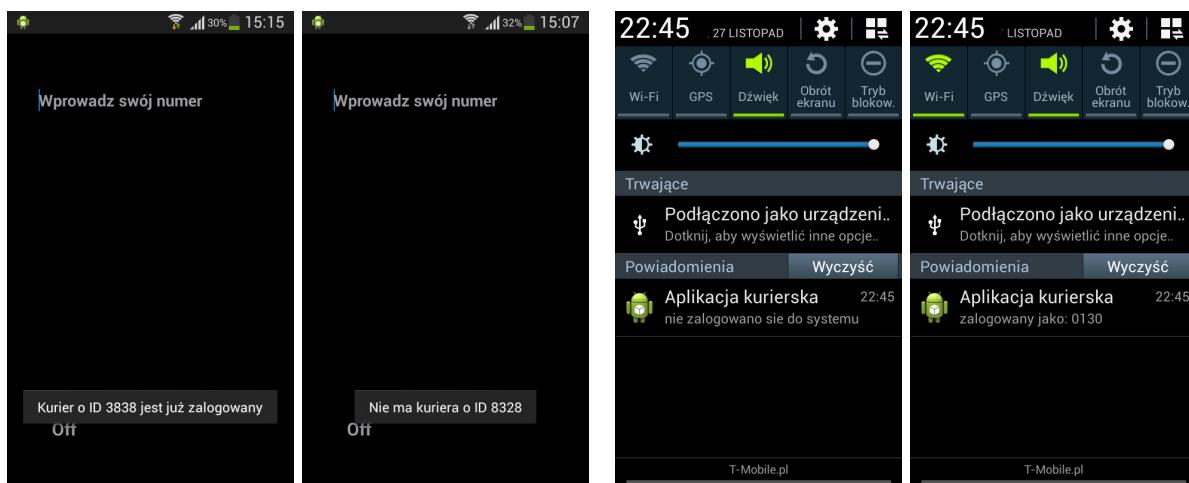
```

1 button.setOnClickListener(new View.OnClickListener() {
2     public void onClick(View v) {
3         if (savedId != "") {
4             new AuthenticationDeliverer().execute(savedId, "0", "0").get();
5             savedId = "";
6             onDestroy();
7         }
8     }
9 });

```

Listing 2.2: Ustawienie właściwości przycisku “Off” w głównej klasie aplikacji mobilnej w metodzie `onCreate()`

Na listingu 2.2 widać zmienną `savedId`, jest pole klasy, które zapisywane jest wpisany 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).



(a) sytuacja gdy jest już zajęte id lub nie istnieje

(b) status telefonu

Rysunek 2.4: Informacje o zalogowanym/nieistniejącym id i statusbar

Parametry z jakimi tworzony jest nowy obiekt to id kuriera, status aktywności oraz jego logowanie lub wylogowywanie [listing. 2.3]. Status aktywności to informacja wysyłana do serwera (zapisywana w bazie danych) mówiąca o tym, czy kurier będzie ustawał swój status na aktywny czy nieaktywny - czy rozpoczęyna 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. 2.4(a)]. Natomiast trzeci parametr mówi o tym, czy kurier się zalogowuje „1” czy wylogowuje „0” z aplikacji. Na rzecz klasy `AuthenticationDeliverer()` wywołane zostają metody: `execute()` - nakazuje wykonanie zadania, `get()` - oczekuje, aż zadanie zostanie wykonane.

```

1 protected String doInBackground(String... params) {
2     ArrayList<NameValuePair> pairs = new ArrayList<NameValuePair>();
3     pairs.add(new BasicNameValuePair("ID", params[0]));
4     pairs.add(new BasicNameValuePair("activ", params[1]));
5     pairs.add(new BasicNameValuePair("logout", params[2]));
6     httpPostAuthentication.setEntity(new UrlEncodedFormEntity(pairs));
7     new Thread(new Runnable() {
8         @Override
9         public void run() {
10            httpClient.execute(httpPostAuthentication);
11        }
12    }).start();
13 }

```

```

11     }
12 }).start();
13 httpResponse = (new DefaultHttpClient()).execute(httpGetAuthentication);
14 String entityStr = EntityUtils.toString(httpResponse.getEntity());
15 if (entityStr.contains("logIn"))
16     return "logIn";
17 else if (entityStr.contains("busy"))
18     return "busy";
19 else if (entityStr.contains("false"))
20     return "false";
21 else if (entityStr.contains("logOut"))
22     return "logOut";
23 return "";
24 }
```

Listing 2.3: klasa AuthenticationDeliverer metoda doInBackground

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, systemowo dozwolone są tylko cyfry. 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, w której nastąpił odczyt.

Odczyt pozycji GPS dostępny jest dzięki użyciu klasy LocationManager, która zapewnia dostęp do lokalizacji systemu. LocationManager należy zainicjalizować klasą Context (umożliwia dostęp do zasobów systemu i informację o nim), natomiast getSystemService jest do kontroli pobierania lokalizacji. Następnie ustalono z jaką częstotliwością ma być odczytywana zmiana pozycji. Po wykonaniu tych czynności następuje zapytanie o ostatnią znaną pozycję. Całą procedurę odczytu pozycji GPS pokazano na listingu 2.4.

```

31
32 public Location getLocation() {
33     locationManager = (LocationManager) context
34         .getSystemService(Context.LOCATION_SERVICE);
35     locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
36         minTime, minDistance, (android.location.LocationListener) this);
37     if (locationManager != null) {
38         location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
39         if (location != null) {
40             longitude = location.getLongitude();
41             latitude = location.getLatitude();
42         }
43     }
44     return location;
45 }
```

Listing 2.4: Pobieranie lokalizacji kuriera metoda getLocation() z klasy GpsTrack

Ostatnią istotną częścią jaka realizowana jest na systemie Android to połączenie z serwerem. Niemniej zostanie nawiązane połączenie konieczne jest przygotowanie treści wiadomości jaka ma zostać wysłana na adres serwer. Następuje to poprzez wpisanie par ciągów znaków, w tym jedna część to identyfikator, a druga to jego wartość, do tablicy. Na jej podstawie wystosowany jest URL httpPost i wykonywany na kliencie http (tutaj na Servlecie). 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("aktiv", 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 2.5: 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 na systemie Android. W opisie i w listingach ominięto obsługę wyjątków.

## 2.2 Serwer

W niniejszej pracy skorzystano możliwości rozszerzenie języka Java o funkcje serwera WWW. Przez takie wykorzystanie języka Java takie serwer nazywany jest wtedy Serwletem - aplet Javy. Zadaniem serwera było obsłuzenie klientów firmy kurierskiej, którzy wpisując numer swojej przesyłki mogli sprawdzić gdzie ona się aktualnie znajduje. Oprócz tej funkcji Servlet też jest pośrednikiem między aplikacją mobilną a bazą danych.

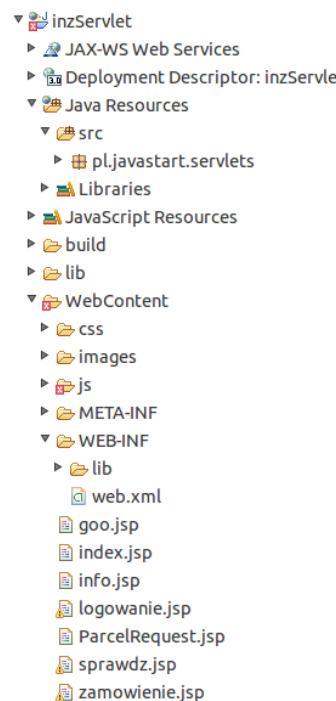
Rozpoczęcie pracy z servletem wymagało zainstalowanie dodatków do środowiska Eclipse:

- 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 wersji 7.0. Serwer jest dostępny pod adresem localhost:8080.

Po skonfigurowaniu środowiska można było przystąpić do utworzenia nowego dynamicznego projektu sieciowego (*Dynamic Web Project*). W strukturze projektu najważniejsze są dwa foldery [rys.2.5], jeden zawierający w sobie klasy Javy, drugi zawierający obsługę, wygląd stronServletu. W tymże folderze znajduje się plik „web.xml”, w którym definiuje zachowanie serwera w zależności od tego jaki adres został wpisany w przeglądarce.

Na poniższym listingu widać ustawienie strony startowej (<welcome-file-list>), tzn. gdy w polu adresy wpisany jest <http://localhost:8080/inzServlet> uruchamia się strona główna aplikacji webowej - Index.jsp. W następnej liniach jest



Rysunek 2.5: Struktura plików Servletu

załadowanie klasy Javy(`pl.javastart.servlets.Sprawdz`) do Servletu pod nazwą `Sprawdz` i zmapowanie tej klasy pod adres `http://localhost:8080/inzServlet/sprawdz`. Pokazany tu sposób dotyczy wszystkich używanych przez Servlet klas Javy, 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>
7 </servlet>
8 <servlet-mapping>
9   <servlet-name>Sprawdz</servlet-name>
10  <url-pattern>/sprawdz</url-pattern>
11 </servlet-mapping>
12 <servlet>
13  <servlet-name>Index</servlet-name>
14  <jsp-file>/index.jsp</jsp-file>
15 </servlet>
16 <servlet-mapping>
17  <servlet-name>Index</servlet-name>
18  <url-pattern>/index</url-pattern>
19 </servlet-mapping>

```

Listing 2.6: Fragment pliku web.xml

Każda klasa Javy, która rozszerza `Servlet` posiada dwie istotne metody do obsługi stron serwera - `doPost` i `doGet`, obydwie przyjmują argument typu `HttpServletRequest` i `HttpServletResponse`. Metody mają zapewniać komunikację pomiędzy serwerem a klientem. Argument `HttpServletRequest` zawiera żądanie klienta do wykonania na serwecie, natomiast `HttpServletResponse` jest odpowiedziom serwera na żądanie klienta. Wspomniana metoda `doGet` obsługuje żądania jakie powstały w nagłówku http - pobiera parametry i przetwarza je - obsługuje zadania typu request. Metoda `doPost` również obsługuje żądanie powstałe z uzupełnienia formularzy.

Ważną klasą w projekcie jest klasa obsługująca bazę danych. Klasa do obsługi bazy danych musi składać się z następujących polecień: załadowanie sterownika bazy danych, nawiązanie połączenia z bazą, pobranie/zapisanie danych, zamknięcie połączenia. Przykładem obsługi bazy danych jest poniższy listing [listing 2.7]. Przedstawiona metoda sprawdza czy istnieje w bazie dany kurier. 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 sprawdziłaby czy wynik zapytania jest pusty. Po uzyskaniu wyniku, że kurier istnieje sprawdzana jest jego aktywność, gdy kurier jest nie aktywny a funkcja była wywoływana w celu zalogowania systemu przygotowywane jest zapytanie o zaktualizowanie danych dla tego kuriera i ustawnieniu jego aktywności na stan aktywny (`true`). Natomiast gdy aktywność `id` kuriera jest `true`, a kurier pragnie się zalogować (metoda została wykonana dla zalogowania do systemu) metoda zwraca ciąg „busy” - informuje o zajętości `id`. W momencie wylogowywania kuriera (kiedy aktywność == `true`) a `logout` wynosi 0 to przygotowywana jest komenda aktualizująca bazę danych i pod wskazaną krotką o numerze `id` zapisywana wartość 0 dla `activ`.

```

1 Class.forName("com.mysql.jdbc.Driver");
2 Connection connection = DriverManager.getConnection(
3   "jdbc:mysql://localhost:3306/deli", "root", "sun5flower");
4 Statement statement = connection.createStatement();
5 StringSelect = "select * from deli.deliverer where id=" + id;
6 ResultSet resultSet = statement.executeQuery(stringSelect);
7
8 while (resultSet.next()) {
9   delivererId = resultSet.getInt("id");
10  delivererActiv = resultSet.getBoolean("activ");
11 }
12 if (delivererId > 0) {
13   if (delivererActiv == false) {
14     sqlInsert = " update deli.deliverer set activ=1 where id=" + id;
15     statement.executeUpdate(sqlInsert);
16     connection.close();
17     statement.close();
18   return "logIn";
}

```

```

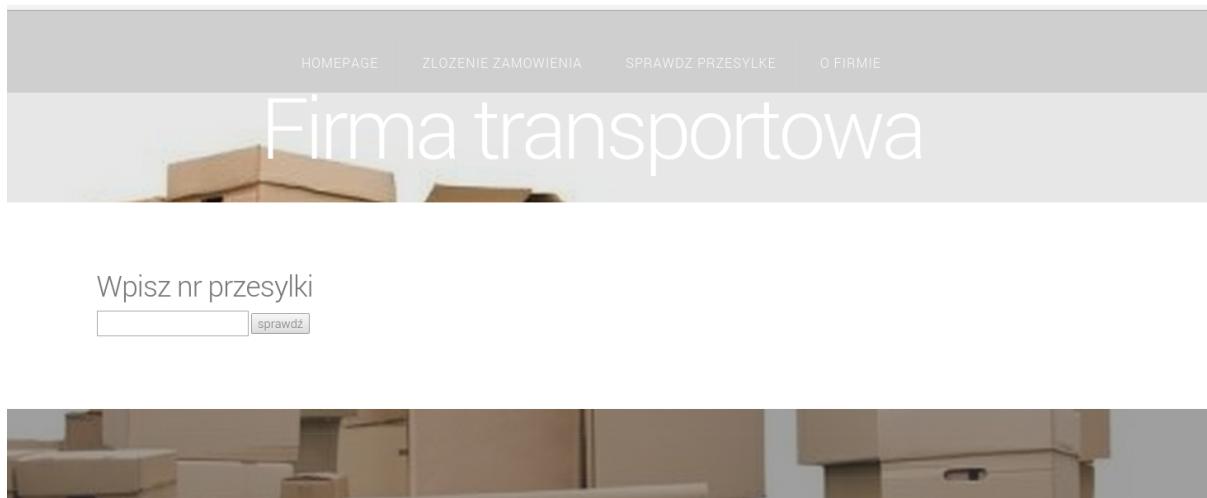
19 } else if (delivererActiv == true) {
20     if (logout.equals("0")) {
21         sqlInsert = " update deli.deliverer set activ=0 where id=" + id;
22         statement.executeUpdate(sqlInsert);
23         connection.close();
24         statement.close();
25         return "logOut";
26     } else if (logout.equals("1")) {
27         connection.close();
28         statement.close();
29         return "busy";
30     }
31 }
32 } else
33     return "false";

```

Listing 2.7: 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 PRZESYŁKE wpisywany jest numer przesyłki jaką klient chce sprawdzić [rys. 2.6]. Wpisany ciąg znaków jest sprawdzany pod kątem poprawności. Sprawdzane jest czy ciąg w formularzu jest liczbowy i czy istnieje taka przesyłka. Gdy w bazie nie istnieje przesyłka pojawia się odpowiedni komunikat informujący o nie istniejącej przesyłce [rys. 2.8]. Gdy użytkownik wprowadzi poprawny numer przesyłki zostaną wyświetlane takie informacje, jak nadawca, odbiorca, czas nadania i czas odbioru oraz ostatnie sczytane położenie przesyłki. Ponadto wyświetlana jest mapa pokazująca jak trasę pokona przesyłka - znaczniki A - B. Na mapie dodatkowo pokazany jest znaczek, który wyznacza pozycję kuriera z daną przesyłką - znaczek „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 (main) pliku.



Rysunek 2.6: Widok strony internetowej SPRAWDZ serwisu z zachętą do wprowadzenia numeru przesyłki do sprawdzeni

Z formularza [rys. 2.6] pobierany jest numer poszukiwanej przesyłki [listing ?? linie 1-4] i ustawiany jako post formularza i przesyłany do klasy Java 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 ?? linia 43] dotycząca błędного id. Natomiast gdy wpisany id jest bezbłędne w tabeli [rys. 2.7] zostają wypisane dane dotyczące przesyłki oraz mapa. Linia 39 listingu 2.8 jest miejscem użycia JavySciprt'owego kodu google.maps. 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 przebywaniu przesyłki w centrali.

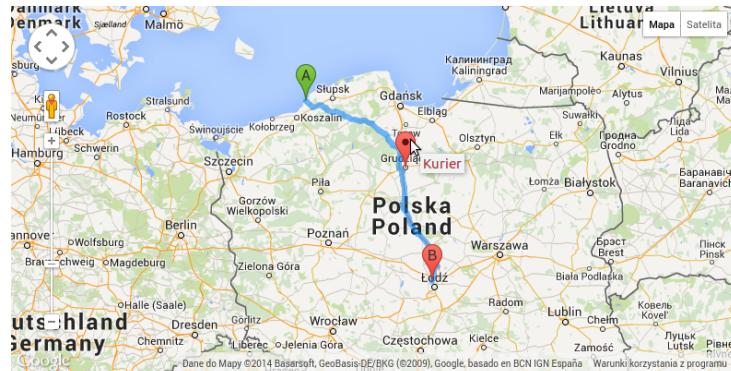
HOMEPAGE ZŁOŻENIE ZAMÓWIENIA SPRAWDZ PRZESYŁKE O FIRMIE

# Firma transportowa

Wpisz nr przesyłki

sprawdź

Numer przesyłki 200  
 Nadawca Anna Kwiatkowska  
 Dluga 2 Darlowo 76-153  
 Czas nadania 2014-11-20 19:23  
 Odbiorca Jozia Piotrkowa  
 Piaskowa 22 Zgierz 95-100  
 Planowany czas dostarczenia 2014-11-22 16:00  
 Ostatnie zarejestrowane 2014-11-21 15:55  
 położenie



Rysunek 2.7: Widok strony internetowej SPRAWDZ z wyznaczoną trasą dla przykładowej przesyłki

HOMEPAGE ZŁOŻENIE ZAMÓWIENIA SPRAWDZ PRZESYŁKE O FIRMIE

# Firma transportowa

Wpisz nr przesyłki

sprawdź

Nie mamy przesyłki w bazie o numerze: 6423



Rysunek 2.8: Przykład odpowiedzi serwisu na nieprawidłowe wpisanie numeru przesyłki

*komentarz: zrobić tu jakaś przeorganizowanie, żeby nie było tak że zdjecie obok listingu*

```

1 <h2>Wpisz nr przesyłki </h2>
2   <form id="formularz" method="post" action="">
3     <input type="text" name="nr" /> <input type="submit" value="sprawdź" />
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 przesyłki"); %</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>${userUserName}<br> ${userUserAddr} </td>
26 </tr>
27 <tr>
28   <td><% out.print("Planowany czas"); %><br> <% out.print("dostarczenia"); %>
29   </td>
30   <td>${timeDelivery}</td>
31 </tr>
32 <tr>
33   <td><% out.print("Ostatnie zarejestrowane"); %><br>
34   <% out.print("położenie"); %></td>
35   <td>${lastTime}<br>${msg}</td>
36 </tr>
37 </table>
38 <div id="mapka"> </div>
39 <%
40 <%
41 <%
42 <%
43 else {
44 <%
45 <br> ${msg} ${id}
46 <%
47 <%
48 <%

```

Listing 2.8: Ciało pliku JavaServlet Pages - sprawdz.jsp

Natomiast ustawianie parametrów umieszczonych w \${...} zrealizowano poprzez klasę rozszerzającą Servlet. Na listingu 2.9 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 2.11). Następnie sprawdzane jest w instrukcji warunkowej if czy taka przesyłka istnieje i wprowadzone znaki są typu integer, jeśli nie istnieje to w zależności od tego jaki błąd wystąpił przygotowywana jest wiadomość zwrotna dotycząca błędu. Natomiast jeśli przesyłka istnieje ustawiane są parametry zwrotne req.setAttribute("...", ...). Warto zwrócić uwagę na to, że pierwszy argument setAttribute musi być zgodny z oczekiwana nazwą zmiennej w pliku JSP, w inny wypadku nie zostanie przypisana wartość z klasy Javy do JSP. Na przykład argumenty z listingu ?? z linii 45 muszą się zgadzać argument z listingu 2.9 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.getRegisteredUserCodeUser());
16            req.setAttribute("timeSend", searchParcel.getParcelSendTime().substring(0, 16));
17            req.setAttribute("userUserName", 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", "Przesyłka w bazie: "
27             + searchParcel.getCentreNameCentre());
28     } else {
29         req.setAttribute("msg", "Nie mamy
30         przesyłki w bazie o numerze: ");
31         req.setAttribute("id", id);
32     }
33     if (id == null || (Sth.isInteger(id)) == false)
34         req.setAttribute("msg", "Prosze podac poprawny numer przesyłki");
35     req.setAttribute("isParcelExists", isParcelExists);
36     req.removeAttribute("nr");
37     view.forward(req, resp);
38 }
39

```

Listing 2.9: Fragment klasy Sprawdz.java metoda doPost()

Na podstawie źródła [5] 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 wypełnienie(renderowanie) wyświetlanej mapy oraz tworzony jest nowy obiekt DirectionsService, który jest wylicza trasę pomiędzy punktami. Do obiektu DirectionsRenderer przypisywane jest umiejscowienie elementu układzie strony przez pobranie jego id oraz możliwe jest ustawnienie parametrów mapy. Kolejnym krokiem jest utworzenie znacznika na pozycji kuriera i dodanie jej do mapy. Ostatnim etapem jest wyliczenie trasy pomiędzy dwoma punktami, adresem nadawcy a adresem odbiorcy.

```

1 <script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
2 <script>
3     var lat = "${lat}";
4     var lon = "${lon}";
5     var start = "${regUserAddr}";
6     var end = "${userAddr}";
7     var directionsRenderer = new google.maps.DirectionsRenderer();
8     var directionsService = new google.maps.DirectionsService();
9     var map;
10
11     function initialize() {
12         map = new google.maps.Map(document.getElementById('mapka'));
13         directionsRenderer.setMap(map);
14         new google.maps.Marker({
15             position : new google.maps.LatLng(lat, lon),
16             map : map,
17             title : "Kurier"
18         });
19         var request = {
20             origin : start,
21             destination : end,
22             travelMode : google.maps.TravelMode.DRIVING
23         };
24         directionsService.route(request, function(response, status) {
25             if (status == google.maps.DirectionsStatus.OK) {
26                 directionsRenderer.setDirections(response);
27             }
28         });
29     }
30     google.maps.event.addDomListener(window, 'load', initialize);
31 </script>

```

Listing 2.10: Kody JavaScript'owy pobierający mapę z serwera Google

Klasa Javy 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 2.11. Klasa otwiera połaczenie z bazą danych, następnie wykonując komendy MySQL pobiera dane dotyczące przesyłki, następnie posiadając już id kuriera, który przewozi tą przesyłkę, pobiera jego położenie, oraz pobiera z bazy informacje o nadawcy (zarejestrowanym użytkowniku). Ostatnim krokiem jest zamknięcie połączenia z bazą danych.

```

1 public SearchParcel(int _nrParcel) {
2     selectFromDB(_nrParcel);
3 }
4 void selectFromDB(int nrParcel) {
5     String strSelect = "";
6     Class.forName("com.mysql.jdbc.Driver");
7     Connection connection = DriverManager.getConnection(
8         "jdbc:mysql://localhost:3306/deli", "root", "sun5flower");
9     Statement statement = connection.createStatement();
10    strSelect = "select * from deli.parcel where id=" + nrParcel;
11    ResultSet resultSet = statement.executeQuery(strSelect);
12    while (resultSet.next()) {
13        isParcelExists = true;
14        parcelId = resultSet.getInt("id");
15        parcelFromUserId = resultSet.getInt("fromUserId");
16        parcelAddresseeName = resultSet.getString("addresseeName"); req.setAttribute("regUserName",
17            searchParcel.getRegisteredUserNameUser());
18        parcelAddresseeStreet = resultSet.getString("addresseeStreet");
19        parcelAddresseeCity = resultSet.getString("addresseeCity");
20        parcelAddresseeCityCode = resultSet
21            .getString("addresseeCityCode");
22        parcelSendTime = resultSet.getString("sendTime");
23        parcelTimePos = resultSet.getString("timePos");
24        parcelDeliveryTime = resultSet.getString("deliveryTime");
25        parcelDeliverer = resultSet.getInt("deliverer");
26    }
27    if (isParcelExists == true) {
28        strSelect = "select * from deli.deliverer where id="
29            + parcelDeliverer;
30        resultSet = statement.executeQuery(strSelect);
31        while (resultSet.next()) {
32            delivererId = resultSet.getInt("id");
33            delivererLatitude = resultSet.getDouble("latitude");
34            delivererLongitude = resultSet.getDouble("longitude");
35            delivererTimePos = resultSet.getString("timePos");
36        }
37        strSelect = "select * from deli.registeredUser where id="
38            + parcelFromUserId;
39        resultSet = statement.executeQuery(strSelect);
40        while (resultSet.next()) {
41            registeredUserId = resultSet.getInt("id");
42            registeredUserNameUser = resultSet.getString("nameUser");
43            registeredUserStreetUser = resultSet
44                .getString("streetUser");
45            registeredUserCityUser = resultSet.getString("cityUser");
46            registeredUserCodeUser = resultSet
47                .getString("cityCodeUser");
48        }
49    }
50    connection.close();
51    statement.close();
52}

```

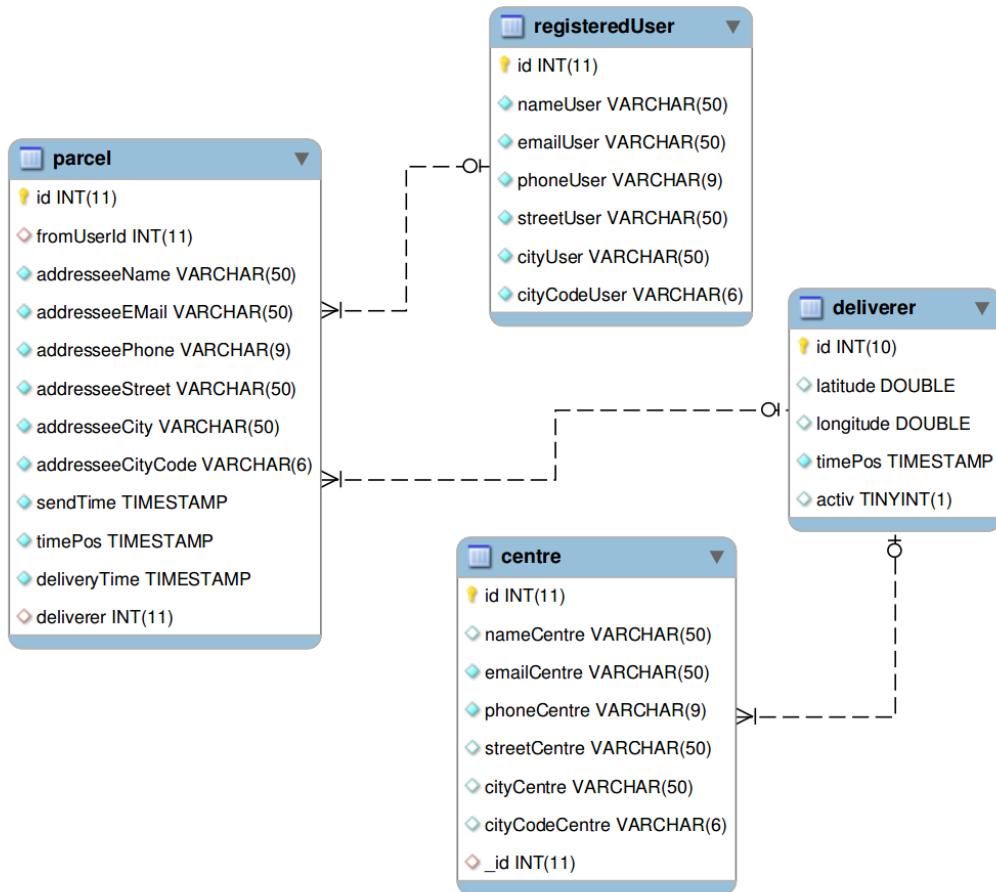
Listing 2.11: Klasa SearchParcel.java przedstawiona metoda selectFromDB() która reprezentuje połączenie z bazą danych oraz pobiera wszystkie dane o przesyłce

*komentarz: Wyznaczanie przyblzonego czasu dostawy zrealizowano korzystajac (to i wiecej juz o serwecie nie trzeba)*

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>.

## 2.3 Baza danych

W przedstawionym projekcie konieczne było utworzenie bazy danych, która przechowuje dane dotyczące kurierów, przesyłek, klientów, centrali kurierskich. W tym celu stworzono 4 tabele: `parcel`, `deliverer`, `registeredUser` i `centre`.



Rysunek 2.9: Diagram ERD zaprojektowanej bazy danych

Na rysunku 2.9 przedstawiono diagram ERD pokazujący tabele oraz połączenia pomiędzy tabelami. Każda z tabel ma swój klucz główny `id`. Tabela `parcel` (przesyłka) zawiera w sobie dwa obce klucze (`foreign key`) odwołującą się one do: `deliverer.id` - klucz główny tabeli kurier oraz `registeredUser.id` - klucz główny tabeli użytkownika zarejestrowanego, czyli tego który złożył zamówienie na przesyłkę. Tabela `centre`, która ma zawierać wpisy dotyczące nazwy, ulicy, miasta zawiera w sobie obcy klucz 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 3

# Przygotowanie i uruchomienie aplikacji

Do 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 co najmniej 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 emulator do obrazu 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 serwera realizuje się od otworzenia zakładki Window>Preferences>Server>Runtime Environment i tam należy dodać (add) Apache Tomcat również w najnowszej wersji. Serwer będzie już wtedy zainstalowany.

Autor korzysta z systemu operacyjnego Linux dystrybucja Ubuntu, więc zostanie opisana instalacja  
*...Darku opisz, ja nie wiem jak, bo Ty mi to zrobisz :\**

Jeśli w IDE Eclipse nie ma dodanych odpowiednich perspektyw trzeba je dodać (Open Perspective) będzie potrzebna perspektywa „Java” oraz „SQL Explorer”. Po spełnieniu wszystkich poprzednio wymienionych kroków można eksportować projekty. Import projektów wybiera się w zakładce File>Import>Editing Project Into Workspace.

## Rozdział 4

# Wykorzystane technologie

Zrealizowany tu projekt bazuje na nowoczesnych technologiach. Skorzystano z mobilnego urządzenia - telefonu komórkowego z systemem Android, bazy danych do przechowywania informacji, a także serweru, który łączy wszystkie elementy w jedną spójną całość. Głównym językiem programowania wykorzystanym w projekcie jest język Java, dzięki któremu zrealizowano aplikację mobilną, obsługę Servletu, bazy danych, odpytywania i parsowania odpowiedzi serwera Google o widok mapy i odległości pomiędzy dwoma punktami, a także obsługa witryny http. Całą aplikację stworzono za pomocą IDE Eclipse z odpowiednimi dodatkami.

### 4.1 Java

Java jest obiektowym językiem programowania ogólnego przeznaczenia. Charakteryzuje się silnym ukrankowaniem na obiektowość oraz niezależnością i przenosalnością kodu od architekty. Oprócz wyżej wymienionych założeniami języka Java jest prostota, sieciowość, niezawodność, bezpieczeństwo, interpretowalność, wysokowydajny, wielowątkowy, dynamiczny oraz niezależny od architektury.

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



Rysunek 4.1: Logo Java EE.

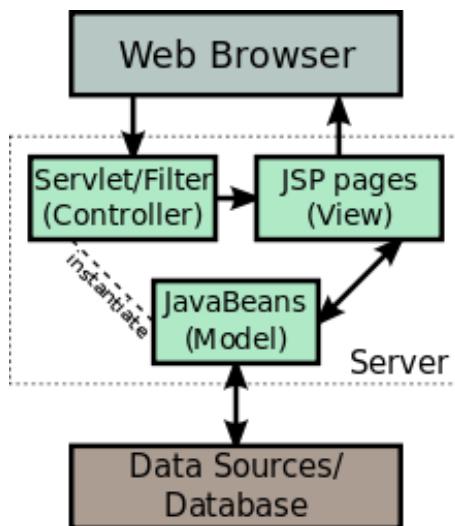
- Zorientowany obiektowo;
- Sieciowy - Java posiada bibliotekę, która w przystępny sposób umożliwia pracę z protokołami http, TCP/IP, FTP;
- Niezawodny - szczególnie skupiono się na wykrywaniu ewentualnych problemów, zapobieganiu sytujom, w których może błęd nastąpić oraz sprawdzaniu błędów podczas działania programu;
- Bezpieczny - Java może służyć do zastosowań sieciowych, z tego powodu zadbane o możliwe najlepsze zabezpieczenie przed wirusami i ingerencją osób trzecich;
- Niezależny od architektury - Java kompilowana jest do kodu pośredniego (bajtowego), który następnie jest interpretowany na maszynie wirtualnej Javy, która jest dostosowana do odpowiedniego systemu. Maszyna wirtualna Javy(JVM) jest zdolna wykonywać program z kodu pośredniego. Z tego powodu język Java stosowany jest na wielu urządzeniach oraz różnych systemach operacyjnych. Niestety konsekwencją przenosalności kodu jest jego wolniejsze wykonanie;
- Przenośny - Java posiada ściśle określone rozmiary typów danych i nie ma możliwości zmiany rozmiaru przez programistę przez co nie następuje np. zmiana kolejności bajtów;
- Interpretowany - program nie jest kompilowany tylko przechowywany w postaci kodu źródłowego, podczas uruchomienia zostaje dopiero interpretowany i uruchamiany przez interpreter języka;
- Wysokowydajny - istnieje możliwość tłumaczenia kodu bajtowego w locie, co zwiększa szybkość ładowania się programu;

- Wielowątkowy - pozwala na interaktywność między procesami, a także pracę w czasie rzeczywistym;
- Dynamiczny - obiekty w Javie można zmieniać w zależności od zmieniającego się środowiska oraz 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 funkcjonalności tych języków, z nieużytecznych, trudnych lub powodujących często błędy zrezygnowano. Język Java umożliwia dziedziczenie, a ponadto wszystkie obiekty Javy są pochodną obiektu bazowego. Jednakże Java nie umożliwia dziedziczenia wielobazowego, dlatego do Javy wprowadzono interfejsy - abstrakcyjny typ, który posiada jedynie operacje, ale nie posiada danych, z tego powodu można tylko implementować interfejs i nie można utworzyć obiektów tego typu. Język Java umożliwia pisanie aplikacji stacjonarnych, webowych czy mobilnych. Język Java ma rozbudowaną obsługę wyjątków. Posiada dobrze rozbudowanego *GarbageCollector* (odśmieciciela) [3].

## 4.2 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 4.2: Schemat systemu Model-View-Controller model 2[8]

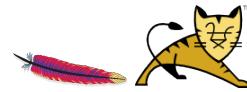
Autor w swojej pracy użył modelu MVC2 [rys. 4.2] Uzasadnieniem użycia tego wzorca jest ułatwiona organizacja aplikacji, w której istnieje interfejs graficzny użytkownika. Dzięki niemu w prosty logiczny sposób można było rozdzielić logikę, kontrolę i widok. Kontrolerem z [rys. 4.2] jest opisany w kolejnym podrozdziale Servlet.

### 4.3 Servlet

Servlet są to aplikacje działające na serwerze WWW korzystające z języka Java. Serwlety mają zapewniać budowanie aplikacji internetowych niezależnych od platformy. Servlet umożliwia korzystanie z baz danych i http. Z tego powodu wykorzystywane są do budowania interaktywnych aplikacji internetowych.

Serwer Apache obsługuje www za pomocą protokołu. Wybrana w projekcie dystrybucja to Servlet Tomcat Apache [rys. 4.3], który jest http, jest otwarty, zapewnia wielowątkowość, skalowalność, bezpieczeństwo oraz kontrolę dostępu [2].

Wybór Tomcat Apache na serwer podkutowany był przez wybór jako głównego języka aplikacji - Javy. Servlet jest dość popularnym narzędziem, co pomogło także w uruchomieniu i skonfigurowaniu go.



Rysunek 4.3: Logo Apache i Apache Tomcat.

### 4.4 JSP - Java

JSP (ang. JAvAServer Pages) jest to technologia, dzięki której możliwe jest dynamiczne tworzenie stron webowych. JSP bazuje na językach znacznikowych, np. HTML, xml oraz innych. Technologia ta jest kompatybilna z servletami (Apache Tomcat, i innymi).

To właśnie wprowadzenie plików JSP wymaga korzystanie z wcześniej opisanego modelu MVC [rys. 4.2]. W dodatku JSP oprócz użycia języków skryptowych umożliwia przeplatanie ich z językiem Java. Wtedy kod, który będzie napisany w Javie musi być wzięty w znaki <% ... %>, np. fragment z listingu 2.8:

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

Natomiast frazy ujęte w znaki \${... } służą do dostępu (pobrania i/lub wysyłania) do danych i funkcji, które powstały w obiektach Javy. Żeby przekazać taką wartość z klasy Javy, klasa ta musi rozszerzać Servlet (`extends HttpServlet`) ustawać taki parametr jaki jest pomiędzy nawiasami {} korzystając z `HttpServletRequest`, jako przykład podano fragment z listingu 2.9:

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

### 4.5 Technologie internetowe

W przedstawionym w tej pracy projekcie korzystano z technologii internetowych, które obsługiwały interakcję z użytkownikiem oraz ‘strony 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 składa się z tagów umieszczonych w <tag>, elementy, np. napisy, umieszczone są pomiędzy tagami <tag> napis </tag>, gdzie pierwszy znacznik jest tagiem oznaczającym początek, a drugi zamkającym. HTML jest budulcem strony internetowej, opisuje jej strukturę. HTML można osadzać w takich językach jak JavaScript, może odnosić się do stylów CSS, który służy do definiowania wyglądu i układu tekstu i innych na stronie;
- XML - jest to język znaczników przeznaczony do reprezentowania danych w strukturyzowany sposób [12]. Tak jak HTML składa się ze znaczników <...> i </...> i tagów.
- Protokół http - jest podstawą do komunikacji danych w WWW, służy do wymiany i przesyłania. 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.
- ...

## 4.6 db - MySQL

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

Systemem, do zarządzania bazą danych w projekcie był MySQL. MySQL jest rozwijany przez firmę ORACLE. Jest opensource'owe rozwiązanie 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].

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 Definition 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 wtedy gdy chce się dodać nową bazę danych czy tabelę.
- **alter** - komenda używana w połączeniu z database, table, function, procedure oraz view. Służy do zmiany struktury bazy danych, tabeli, ..., np można dodać lub usunąć kolumnę istniejącej tabeli.
- **drop** - umożliwia usunięcie bazy danych i tabeli

Komendy DML:

- **commit** - komituje 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.

## 4.7 Android

Android jest systemem wykorzystywanym na platformach mobilnych. Android jest systemem operacyjnym z rodziną Linux, oparty na jądrze Linux. Android umożliwia tworzenie aplikacji na wiele urządzeń, optymalizacji podlega plik xml, gdzie można dostosować aplikację do konkretnych urządzeń. Każdy proces uruchomiony na Androidzie jest uruchamiany na maszynie wirtualnej i jest niezależny od pozostałych aplikacji. Najnowszą wersją systemu jest Android Lollipop 5.0.

Rozpoczęcie pracy z Androidelem zaczyna się od instalacji środowiska, może to być Eclipse z dodatkiem SDK Android lub Android Studio. A samo tworzenie aplikacji od projektu interfejsu użytkownika, następnie dopiero oprogramowuje się obsługę oraz logikę aplikacji, ostatnim etapem jest testowanie aplikacji [1].

System Android składa się z czterech podstawowych elementów.

- Activites - reprezentuje ekran użytkownika, np. aplikacja do obsługi emaila posiada jedną aktywność do listowania nowych emali, a inną do pisania email. Aktywności implementuje się używając podklasę **Activity**;
- Services - jest to składnik aplikacji, który działa w tle, np. w urządzeniu odtwarzana jest muzyka w tle, a użytkownik korzysta z innej aplikacji w tym czasie. Takie zachowanie aplikacji implementuje się za pomocą podklasy **Service**;
- Content providers - służy do zarządzania wspólnymi danymi, a także danymi, które są prywatne dla danej aplikacji. Zarządzanie danymi implementuje się używając podklasy **ContentProvider**;
- Broadcast receivers - jest komponentem systemu, który jest odpowiedzialny za rozgłaszenie informacji po systemie, np. informacja o niskim stanie baterii. Taką właściwość implementuje się korzystając z podklasy **BroadcastReceiver**.

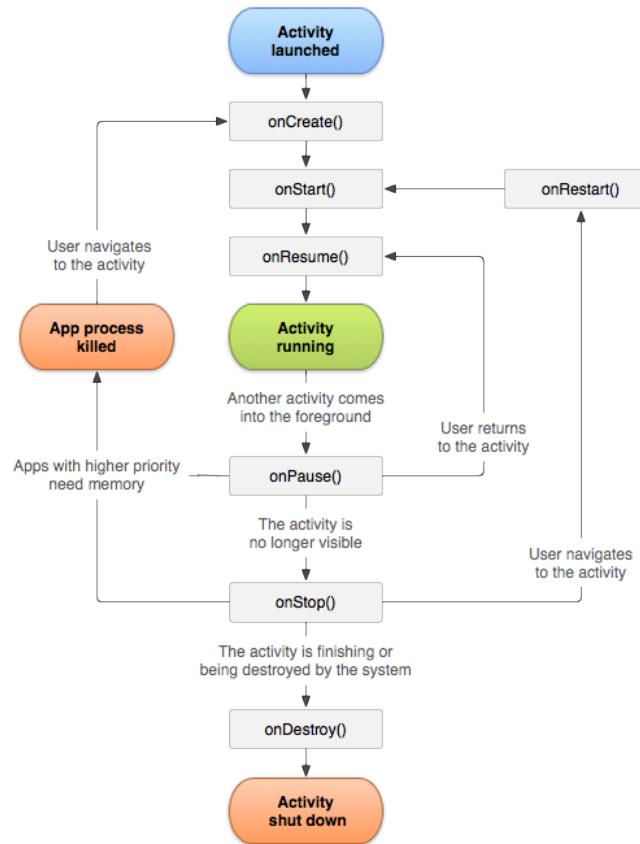
Wartą częścią do roziwnięcia jest element **Activity**. **Activity** jak już wcześniej wspomniano jest reprezentuje ekran użytkownika. Ekran oferuje współdziałanie z użytkownikiem, zazwyczaj okno aplikacji wypełnia cały ekran, ale może być tak naprawdę każdego rozmiaru, także może być pływające na wierzchu innych okien.

Taka aplikacja składa się z wielu części, które są ze sobą powiązane w luźny sposób. Zazwyczaj jedna część aplikacji jest „główna”, najczęściej ta, która jest pokazywana użytkownikowi przy uruchomieniu aplikacji. Następnie każde działanie wykonane na tym ekranie może uruchamiać inny ekran, proces czy działanie. Wtedy przy każdym uruchomieniu nowej działalności „stara” jest zatrzymywana i przechowywana na stosie - LIFO (ang. last in first out). Dzięki temu gdy użytkownik zapragnie wrócić do poprzedniej działalności (ekranu) przyciska przycisk wstecz i poprzednia działalność jest zdejmowana ze stosu.

Utworzenie aplikacji typu **Activity** wymaga utworzenie podklasy **Activity**, w tej podklasie należy zaimplementować metody, które będą określały aplikację pomiędzy różnymi stanami cyklu jej życia. Mówiąc cykl życia aplikacji ma się na myśli tworzenie, zatrzymywanie, wznawianie, niszczenie. Najważniejszymi metodami do zaimplementowania są zatem metody **onCreate()** oraz **onPause()**. Metoda **onCreate()** jak sama nazwa wskazuje, określa stan aplikacji podczas jej tworzenia, natomiast metoda **onPause()** określa co się stanie z aplikacją jeśli opuści ją użytkownik, co nie musi oznaczać, że aplikacja zostanie zakończona, zazwyczaj zachowywany jest wtedy aktualny stan.

Oczywiście istotną kwestią jest zarządzanie życiem aplikacji. Aktywność może być zatem w jednym z trzech stanów: stan wznawiania, wstrzymanie oraz zatrzymania. Stan wznawiania określa stan aplikacji kiedy jest ona włączona i jest na głównym planie ekranu użytkownika - „działa”. Stan wstrzymania dotyczy stanu, w którym aplikacja jest widoczna, ale nie jest na górze, może być częściowo widoczna na ekranie. Stan zatrzymania oznacza, że aplikacja jest całkowicie przysłonięta przez inne i działa w tle. W stanie wstrzymania i zatrzymania system w przypadku braku pamięci może zatrzymać aplikację.

Na rysunku 4.4 przedstawiony został cykl życia aplikacji. Wspomniana metoda **onCreate()** tworzona jest zawsze wtedy, gdy aplikacja jest tworzona, w tym miejscu tworzy się widoki. Następną wywoływaną metodą jest **onStart()** powoduje pokazanie się aplikacji pod restarcie lub zatrzymaniu aplikacji. Metoda **onResume()** wywoływana jest zawsze po pauzie aplikacji i rozpoczyna interakcję z użytkownikiem, przywołuje aplikację na szczyt stosu. **onPause()** wywoływana w momencie, gdy aplikacja ma rozpoczęć inną czynność. Metoda **onStop()** zostaje użyta, gdy aplikacja ma się stać niewidoczna dla użytkownika, aplikacja po tym może być zniszczona, może zostać zamknięta jej aktywność lub może być zrestartowana. Ostatnią metodą przez zniszczeniem aplikacji jest **onDestroy()**, która jest metodą ostatnią jaka może zostać wykonana, wszystkie procesy są zamkane.



Rysunek 4.4: Schemat cyklu życia aplikacji[1]

Przedstawiony na rysunku 4.4 schemat cyklu życia aplikacji w systemie Android jest oczwywiście podstawowy i może posiadać więcej funkcji, jednakże jest to ogólny schemat budowania aplikacji typu **Activity**.

## 4.8 Google apps - mapy

Firma Google udostępnia korzystanie deweloperom ze swoich produktów [6]. W swoim projekcie korzystałam z Google Maps API.

## 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](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. <https://developers.google.com/maps/documentation/webservices/>. [Online; dostęp 16.11.2014].
- [6] Google. <http://developer.google.com>. [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

1.1	podpisisi . . . . .	2
2.1	Struktura systemu . . . . .	3
2.2	Struktura programu aplikacji Android . . . . .	4
2.3	Ekrany Androida . . . . .	5
2.4	Informacje o zalogowanym/nieistniejącym id i statusbar . . . . .	6
2.5	Struktura plików Servletu . . . . .	8
2.6	Widok strony internetowej SPRAWDZ serwisu z zachętą do wprowadzenia numeru przesyłki do sprawdzeni . . . . .	10
2.7	Widok strony internetowej SPRAWDZ z wyznaczoną trasą dla przykładowej przesyłki . . . . .	11
2.8	Przykład odpowiedzi serwisu na nieprawidłowe wpisanie numeru przesyłki . . . . .	11
2.9	Diagram ERD zaprojektowanej bazy danych . . . . .	15
4.1	Logo Java EE. . . . .	17
4.2	Schemat systemu Model-View-Controller model 2[8] . . . . .	18
4.3	Logo Apache i Apache Tomcat. . . . .	19
4.4	Schemat cyklu życia aplikacji[1] . . . . .	22

# Listings

2.1	Nadanie uprawnień aplikacji Android w pliku AndroidManifest.xml . . . . .	4
2.2	Ustawienie właściwości przycisku “Off” w głównej klasie aplikacji mobilnej w metodzie onCreate() . . . . .	6
2.3	klasa AuthenticationDeliverer metoda doInBackground . . . . .	6
2.4	Pobieranie lokalizacji kuriera metoda getLocation() z klasy GpsTrack . . . . .	7
2.5	Metoda startSendGpsDate() klasy main.java. Metoda sprawdza stan sygnału GPS i przy- gotowuje wiadomość do wysłania na serwer oraz wysyła ją . . . . .	7
2.6	Fragment pliku web.xml . . . . .	9
2.7	Połączenia z bazą danych na przykładzie metody sprawdzającej istnienie kuriera oraz jego stan używanej przez aplikację mobilną . . . . .	9
2.8	Ciało pliku JavaServlet Pages - sprawdz.jsp . . . . .	11
2.9	Fragment klasy Sprawdz.java metoda doPost() . . . . .	12
2.10	Kody JavaScript’owy pobierający mapę z serwera Google . . . . .	13
2.11	Klasa SearchParcel.java przedstawiona metoda selectFromDB() która reprezentuje połą- czenie z bazą danych oraz pobiera wszystkie dane o przesyłce . . . . .	14