

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

1 Wstęp i cel pracy	2
2 Rozwiązanie - prezentacja wyników	3
2.1 Aplikacja na system Android	3
2.2 Serwer	7
2.3 Baza danych	9
3 Przygotowanie i uruchomienie aplikacji	11
4 Wykorzystane technologie	12
4.1 Java	12
4.1.1 podpodrozdział	13
4.2 Wzorzec architektoniczny - MVC	13
4.3 Servlet	13
4.4 JSP - Java	14
4.5 Technologie internetowe	14
4.6 db - MySQL	14
4.7 Android	14
4.8 Google apps - mapy	14
5 Możliwość rozwinięcia w przyszłość	15
6 Wnioski i podsumowanie	16
Bibliografia	16

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 poprostu 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. 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 dostraczania przesyłki do adresata, esytmacja 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. <http://www.lokalizacja.info/pl/testy/monitoring/gdzie-jest-moja-paczka-test-firm-kurierskich.html#VEzCPVS988o> 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 adroida oraz serwera www. Serwer, czyli Servlet(plet Javy) posiada połaczenie z bazą danych, w której przechowywane są informacje o przesyłce, kurierach i klientach.

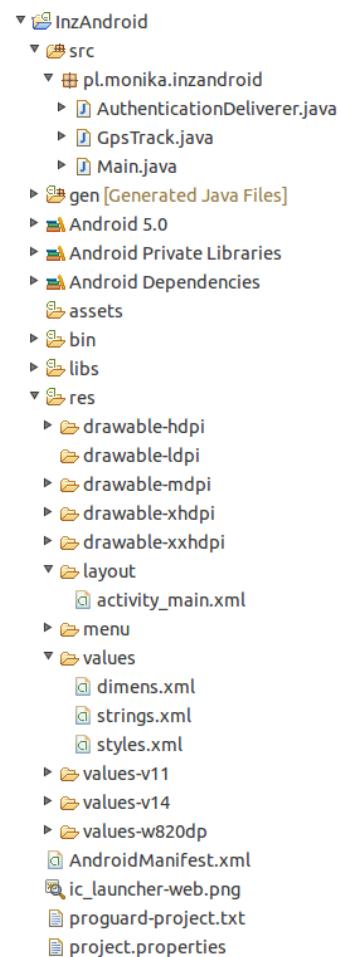
Aplikację zrealizowano w środowisku programistycznym Eclipse.
schemat - wejście (adnroid) - środek system - wyjście www z mapką
dopisać pierdół wstępowych
oprócz tego dopisać jak testowałam to
dorzucić schemat systemu

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 taki funkcje jak zarządzanie projekta, 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. 2.1], 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 nazwę). 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,

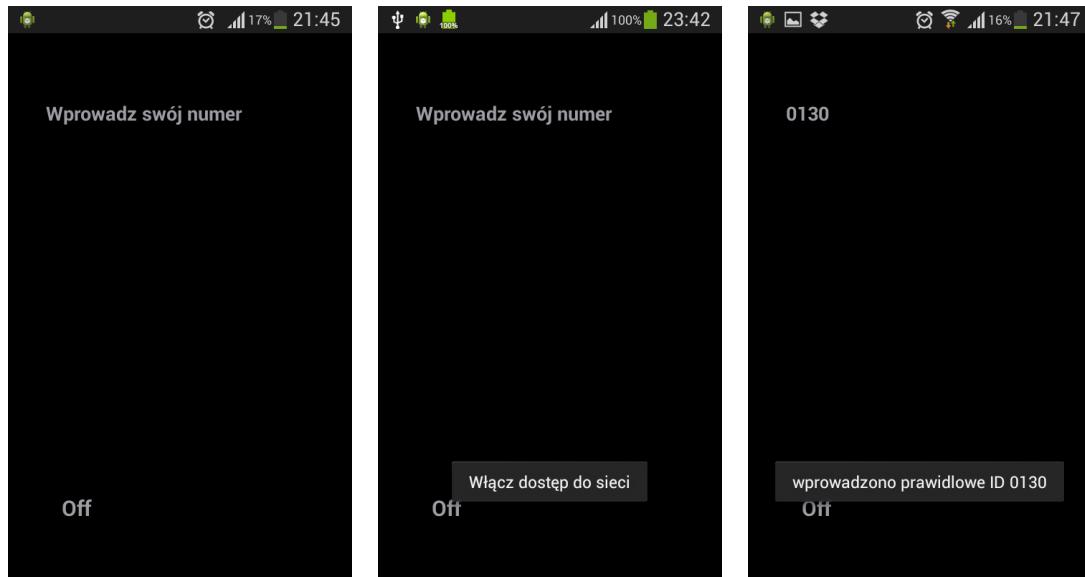


Rysunek 2.1: Struktura programu aplikacji Android

{android}

w którym ustawiane są dostępne opcje menu.

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. 2.2]. 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 hardwarowe przyciski nie wyłączają aplikacji, jedynie ją mimalizują. Takie właściwości zostały stworzone z myślą o tym, aby kurier podczas używania aplikacji tylko w świadomym sposób mógł ją zamknąć.

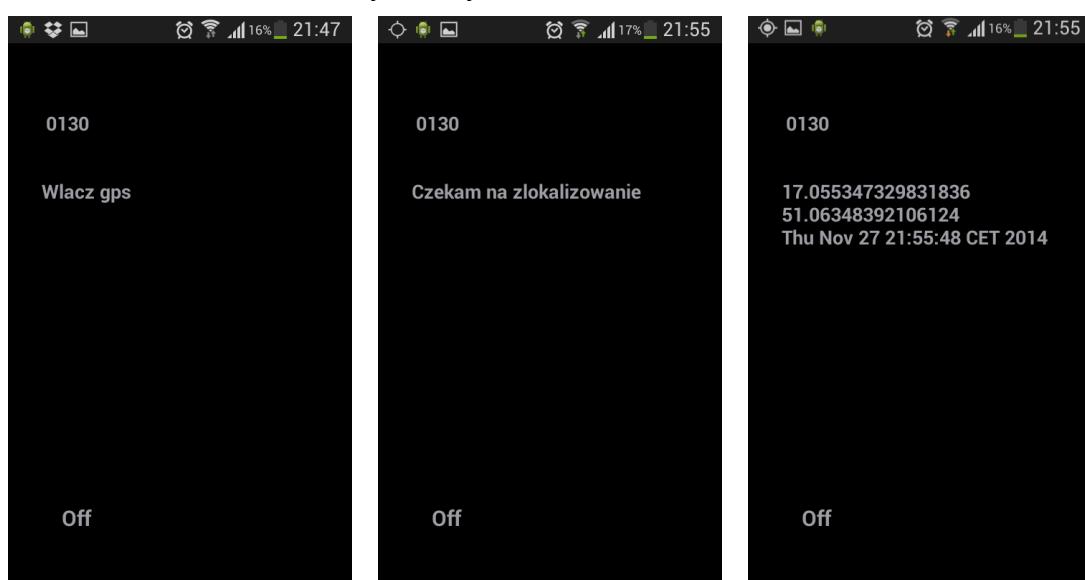


(a) Główny ekran aplikacji, oczekuje na wprowadzenie id kuriera

Włącz dostęp do sieci
Off

wprowadzono prawidłowe ID 0130
Off

0130
mówka lub wifi na ekranie pokazuje tym id
się komunikat i zostaje zablokowane
pole do wprowadzania id



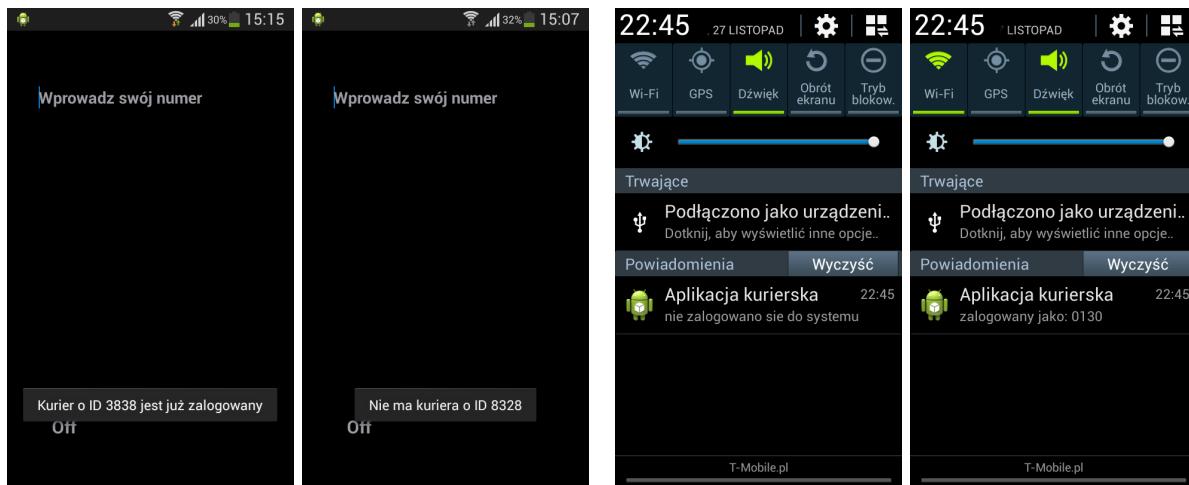
(d) Informacje o niewłączonym GPS (e) System czeka na ustalenie lokalizacji (f) System lokalizuje i wyświetla informacje o lokalizacji i czasie odzysku

Rysunek 2.2: Ekrany Androida

{androidView0

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. 2.3(a)]. Oprócz wyżej wymienionych zachowań aplikacji, w pasku statusu na telefonie wyświetlana jest ikona podczas włączonej aplikacji a także w powiadomieniach [rys. 2.3(b)].

Kolejnym krokiem było nadanie odpowiednich uprawnień aplikacji, do tego służy plik „AndroidManifest.xml”.



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

(b) status telefonu

Rysunek 2.3: Struktura programu aplikacji Android

fest.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

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 2.2].

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

{manifest}

{button}

Listing 2.2: Ustawienie właściwości przycisku "Off"

Na listingu 2.2 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. 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 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. 2.3(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.

{Authentictio

```

31 @Override
32 protected String doInBackground(String... params) {
33     ArrayList<NameValuePair> pairs = new ArrayList<NameValuePair>();
34     pairs.add(new BasicNameValuePair("ID", params[0]));
35     pairs.add(new BasicNameValuePair("activ", params[1]));
36     pairs.add(new BasicNameValuePair("logout", params[2]));
37
38     httpPostAuthentication.setEntity(new UrlEncodedFormEntity(pairs));
39
40     new Thread(new Runnable() {
41         @Override
42         public void run() {
43             httpClient.execute(httpPostAuthentication);
44         }
45     }).start();
46     httpResponse = (new DefaultHttpClient())
47         .execute(httpGetAuthentication);
48     String entityStr = EntityUtils.toString(httpResponse.getEntity());
49     if (entityStr.contains("logIn"))
50         return "logIn";
51     else if (entityStr.contains("busy"))
52         return "busy";
53     else if (entityStr.contains("false"))
54         return "false";
55     else if (entityStr.contains("logOut"))
56         return "logOut";
57     return "";

```

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, któryczytuje 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 informacji o nim), natomiast getSystemService jest do kontroli pobierania lokalizacji. Następnie ustawiono 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 public Location getLocation() {
32     locationManager = (LocationManager) context
33         .getSystemService(Context.LOCATION_SERVICE);
34
35     locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
36         minTime, minDistance, (android.location.LocationListener) this);
37
38     if (locationManager != null) {
39         location = locationManager
40             .getLastKnownLocation(LocationManager.GPS_PROVIDER);
41         if (location != null) {
42             longitude = location.getLongitude();
43             latitude = location.getLatitude();
44         }
45     }
46     return location;
47 }

```

{GpsTrack}

Listing 2.4: Pobieranie lokalizacji

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, 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>

→×tamp=2014-11-25+15:14:55&aktiv=1

```

55  private String localhost = "192.168.1.2:8080";
56  private HttpClient httpClient = new DefaultHttpClient();
57  private HttpPost httpPost = new HttpPost("http://" + localhost
58      + "/inzServlet/insert");
59
60
61
62  Date date = new Date(System.currentTimeMillis());
63  textView.setText(lon + "\n" + lat + "\n" + date);
64  ArrayList<NameValuePair> pairs = new ArrayList<NameValuePair>();
65  pairs.add(new BasicNameValuePair("ID", savedId));
66  pairs.add(new BasicNameValuePair("longitude", lon + ""));
67  pairs.add(new BasicNameValuePair("latitude", lat + ""));
68  pairs.add(new BasicNameValuePair("timestamp",
69      new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(date)));
70  pairs.add(new BasicNameValuePair("aktiv", 1 + ""));
71  httpPost.setEntity(new UrlEncodedFormEntity(pairs));
72
73  new Thread(new Runnable() {
74      @Override
75      public void run() {
76          httpClient.execute(httpPost);
77      }.start();
78  })

```

Listing 2.5: Przygotowanie wiadomości do wysłania na serwer oraz wysłanie jej

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

2.2 Serwer

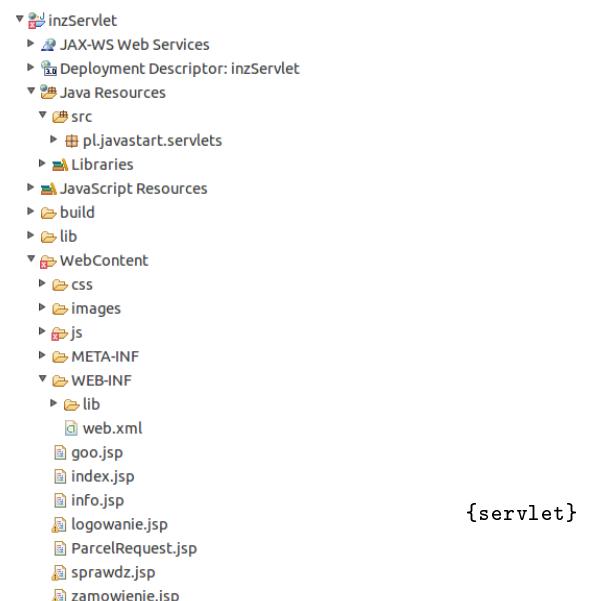
W niniejszej pracy skorzystano możliwości rozszerzenie języka Java o funkcje serwera WWW. Przez takie wykorzystanie języka Java takie serwer nazywany jest wtedy Serwletem - aplet Javy. Zadaniem serwera było obsłuzenie klientów firmy kurierskiej, którzy wpisując numer swojej przesyłki mogili sprawdzić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

Rozpoczęcie pracy z servletem wymagało zainstowanie 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 prystąpić do utworzenia nowego dynamicznego projektu sieciowego (*Dynamic Web Project*). W strukturze projektu najważniejsze są dwa foldery [rys.2.4], jeden zawierający w sobie klasy Javy, drugi zawierający obsługę, wygląd stron servletu. W tymże folderze znajduje się plik „web.xml”, w którym definiuje zachowanie serwera w zależności od tego jaki adres został wpisany w przeglądarkę.



Rysunek 2.4: Struktura Servletu

{webxml}

```

1 <welcome-file-list>
2   <welcome-file>index.jsp</welcome-file>
3 </welcome-file-list>
4
5 <servlet>
6   <servlet-name>Sprawdz</servlet-name>
7   <servlet-class>pl.javastart.servlets.Sprawdz</servlet-class>
8 </servlet>
9 <servlet-mapping>
10  <servlet-name>Sprawdz</servlet-name>
11  <url-pattern>/sprawdz</url-pattern>
12 </servlet-mapping>
13
14 <servlet>
15  <servlet-name>Index</servlet-name>
16  <jsp-file>/index.jsp</jsp-file>
17 </servlet>
18 <servlet-mapping>
19  <servlet-name>Index</servlet-name>
20  <url-pattern>/index</url-pattern>
21 </servlet-mapping>

```

Listing 2.6: Przykładowy

Na powyższym 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 linii jest załadowanie klasy Javy (`pl.javastart.servlets.Sprawdz`) do serwletu 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>`).

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 żadania jakie powstały w nagłówku http - pobiera parametry i przetwarza je - obsługuje żadania typu `request`. Metoda `doPost` również obsługuje żadanie powstałe z uzupełnienia formularzy.

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 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 ustawieniu 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`.

{checkisexist}

```

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

```

```

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

```

Listing 2.7: Przykład połączenia z bazą danych na przykładzie metody sprawdzającej istnienie kuriera oraz jego stan

Projekt dążył do ustawienia przyjaznego dla użytkownika-klienta widoku lokalizacji przesyłki. Do osiągnięcia tego celu użyto zmodyfikowanego przez autora darmowego szablonu pobranego z internetu [cite]. W zakładce „SPRAWDZ PRZESYŁKE” wpisywany jest numer przesyłki jaką klient chce sprawdzić [rys. 2.5(a)]. 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. 2.5(c)]. Gdy użytkownik wprowadzi poprawny numer przesyłki zostaną wyświetlane takie informacje, jak nadawca, odbiorca, czas nadania i czas odbioru oraz ostatnie zczytane położenie przesyłki. Ponadto wyświetlana jest mapa pokazująca jak trasę pokona przesyłka - znaczniki A - B. Na mapie dodatkowo pokazany jest znacznik, który wyznacza pozycję kuriera z daną przesyłką - znacznik „kurier”.

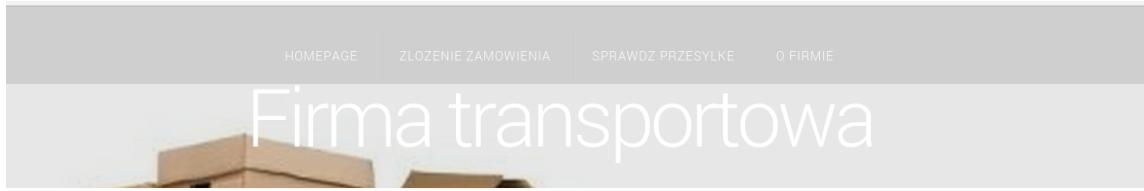
Istotną częścią projektu było zaprojektowanie strony, na której klient może sprawdzać status swojej przesyłki zrealizowane jest to poprzez utworzenie pliku *.jsp, i przypisanie jej adresu http.

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

blblb

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



Wpisz nr przesyłki

sprawdź

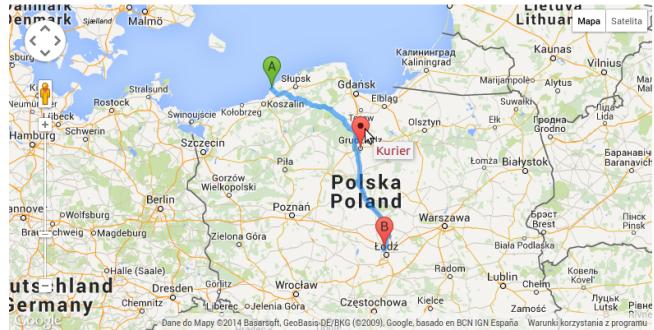
(a) bla



Wpisz nr przesyłki

sprawdź

Numer przesyłki	200
Nadawca	Anna Kwiatkowska
Długa 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 położenie	2014-11-21 15:55



(b) labb



Wpisz nr przesyłki

sprawdź

Nie mamy przesyłki w bazie o numerze: 6423

(c) babal

Rysunek 2.5: Struktura programu aplikacji Android

Rozdział 3

Przygotowanie i uruchomienie aplikacji

Że trzeba stworzyć androida, i servlet i bazę danych

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 serwera, który łączy wszystkie elementy w jedną spójną całość. Głównym językiem programowania wykorzystanym w projekcie jest język Java, dzięki któremu zrelizowano 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. *logo Javy*

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 stostunku do C++) o arytmetykę wskaźnikową, struktury, unie, przeciążanie operatorów itd.
- 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 sytuacjom, w których może błęd nastąpić oraz sprawdzaniu błędów podczas działania programu
- Bezpieczny - Java może służyć do zastosowań sieciowych, z tego powodu zadbane o możliwie 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 ścisłe określone rozmiary typów danych i nie ma możliwości zmiany rozmiaru przez programistę przez co nie następuje np. zmiana kolejności bajtów
- Interpretowany - ...
- 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

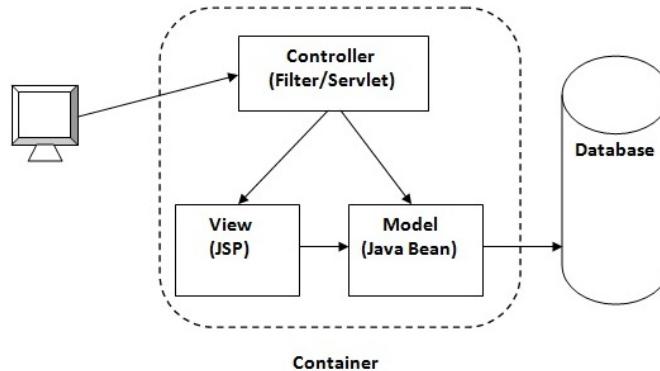
- Dynamiczny - obiekty w Javie można zmieniać w zależności od zmieniającego się środowiska oraz możliwy jest wgląd we wszystkie obiekty, a nawet dodawać nowe metody

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) [4].

4.1.1 podpodrozdział

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. [6]



Rysunek 4.1: Schemat systemu Model-View-Controller model 2[7]

{MVC2}

Autor w swojej pracy użył modelu MVC2 [rys. 4.1] 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. Kontenerem z [rys. 4.1] jest opisany w kolejnym podrozdziale Servlet.

4.3 Servlet

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

open source'owy i korzysta z licencji Apache. Serwer Apache obsługuje www za pomocą protokołu W projekcie skorzystano z serwletu Tomcat Apache [rys. 4.3], który jest http, jest otwarty, zapewnia wiele wątkowości, skalowalność, bezpieczeństwo, kontrolę dostępu [2].



Rysunek 4.2: Logo Apache i Apache Tomcat [3]

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

4.4 JSP - Java

4.5 Technologie internetowe

W przedstawionym w tej pracy projekcie korzystano z technologii internetowych, które obsługiwają interakcję z użytkownikiem oraz 'strony www'. Skorzystano z takich technologii jak:

- JavaScript - jest to skryptowy język programowania stosowany do tworzenia stron internetowych, zapewnia interakcję z użytkownikiem[8]
- XML - jest to język znaczników przeznaczony do reprezentowania danych w strukturyzowany sposób [10]
- Protokół http -
- ...

4.6 db - MySQL

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

Systemem, do zarządzania bazą danych w projekcie był MySQL. Jest opensource'owe rozwiązanie do zarządzania relacyjnymi bazami danych. Charakteryzuje się takimi cechami jak szybki, wielowątkowy dostęp z możliwością obsługi dużej ilości użytkowników. Serwer MySQL może być stosowany do systemów, w których znajdują się dane o znaczeniu krytycznym lub te systemy są mocno obciążane. [9]

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. Najnowszą wersją systemu jest Android Lollipop 5.0.

Rozpoczęcie pracy z Androidem 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 usługę oraz logikę aplikacji, ostatnim etapem jest testowanie aplikacji. [1] *napisac o tym, ze android oddelegowuje zadania, taki obrazek ze strony*

4.8 Google apps - mapy

Firma Google udostępnia korzystanie deweloperom ze swoich produktów [5]. W swoim projekcie korzystałem 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(?) / zmienienie 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 urzynają kolektorów danych.

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] Apache Org. <http://apache.org/>. [Online; dostęp 16.11.2014].
- [4] dokumentacj Javy. <https://docs.oracle.com/javase/specs/jls/se8/html/index.html>. [Online; dostępny 16.11.2014].
- [5] Google. <http://developer.google.com>. [Online; dostęp 16.11.2014].
- [6] Java MVC. <http://www.oracle.com/technetwork/articles/javase/index-142890.html>. [Online; dostępny 26.11.2014].
- [7] Java MVC. <http://www.javatpoint.com/model-1-and-model-2-mvc-architecture>. [Online; dostępny 26.11.2014].
- [8] JavaScript wiki. <http://pl.wikipedia.org/wiki/JavaScript>. [Online; dostępny 16.11.2014].
- [9] MySQL. <http://dev.mysql.com/doc/refman/5.6/en/introduction.html>. [Online; dostępny 26.11.2014].
- [10] xml wiki. <http://pl.wikipedia.org/wiki/XML>. [Online; dostępny 16.11.2014].

Spis rysunków

1.1	podpisisi	2
2.1	Struktura programu aplikacji Android	3
2.2	Ekrany Androida	4
2.3	Struktura programu aplikacji Android	5
2.4	Struktura Servletu	7
2.5	Struktura programu aplikacji Android	10
4.1	Schemat systemu Model-View-Controller model 2[7]	13
4.2	Logo Apache i Apache Tomcat [3]	13

Spis tabel

Listings

2.1	Nadanie uprawnień aplikacji Android	5
2.2	Ustawienie właściwości przycisku “Off”	5
2.3	klasa AuthenticationDeliverer metoda doInBackground	5
2.4	Pobieranie lokalizacji	6
2.5	Przygotowanie wiadomości do wysłania na serwer oraz wysłanie jej	7
2.6	Przykładowy	8
2.7	Przykład połączenia z bazą danych na przykładzie metody sprawdzającej istnienie kuriera oraz jego stan	8