

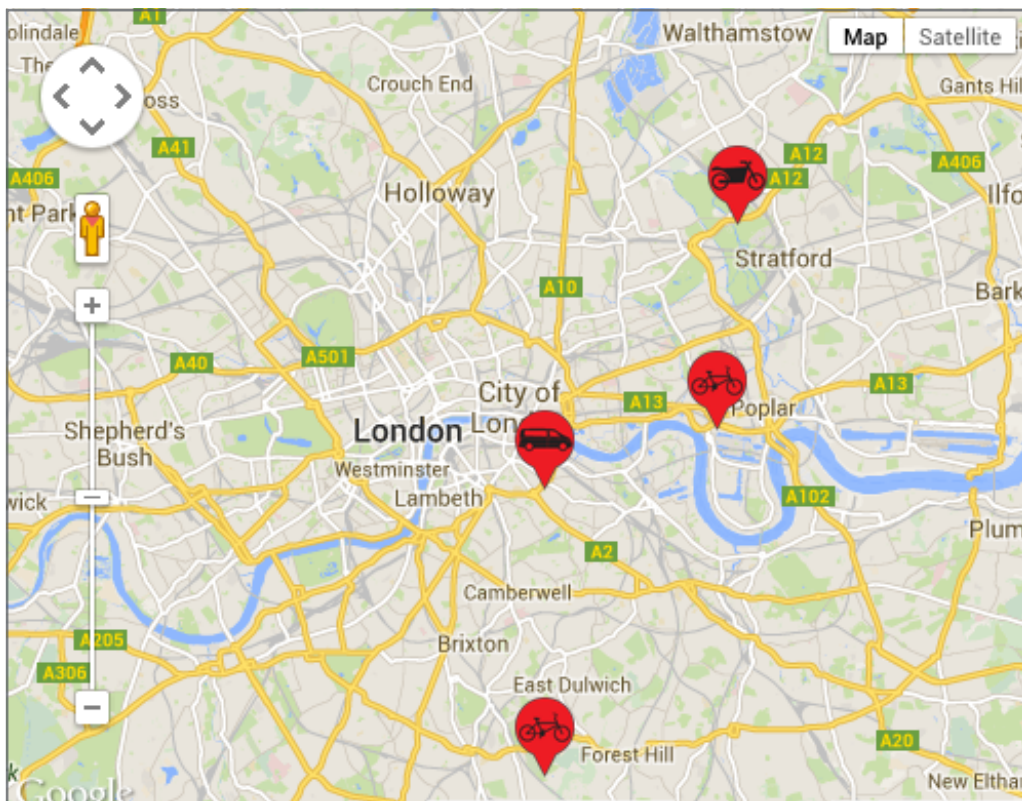
# Online prototype booking system for self-employed couriers

A dissertation submitted in partial fulfilment of the requirements for  
the MSc in Advanced Computing Technologies

by Raitis Cerkasovs

Department of Computer Science and Information Systems Birkbeck College,  
University of London

September 2014



This report is substantially the result of my own work except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. I have read and understood the sections on plagiarism in the Programme booklet and the School's website.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

## **Abstract**

In the project, I've developed an online prototype booking system for self-employed couriers.

Due to a lack of such software results in independently working couriers still being tightened up with commercial organisations, such a Hermes, E-Courier, etc. It ruins the concept of being self-employed. Moreover, it makes additional costs in the delivery process. Companies are acting as middle agents and earning their profit.

If couriers were able to make their location traceable, inform the customer at which stage delivery process is currently due (an approved job, collected or a delivered parcel etc.) and present the history from previously committed jobs, they would be able to do their job independently.

In my project, I've developed a system that would satisfy self-employed couriers' needs. A smartphone was used as a tracking device and web site as a booking platform. Purpose for this system is to decouple couriers from any commercial organisation and make them able to carry on the job without any additional expenses.

## Table of content

1. Introduction	7
1.1 Aim and objectives	7
1.2 Terminology	8
1.3 Application type and approach	9
1.4 Assumed knowledge	9
1.5 Structure of report	9
2. Background	10
2.1 Over all concept of courier activities	10
2.2 Used technologies	10
2.2.1 Google maps	10
2.2.2 Project hosting on Amazon Web Services, scalability and security	11
2.2.3 REST services for android data exchange	13
2.2.4 Android application connectivity issues	14
2.2.5 JQuery	14
3. System Analysis and Design	15
3.1 Problem description	15
3.2 Initial system analysis	15
3.3 High level goals	16
3.4 Use Case Models	17
3.5 Conceptual Class(es) identification	20
3.6 Context Class Diagram design	21
3.7 Identify Design Patterns for web application	24
3.8. Database design	25
4. Implementation	28
4.1 Initial identification of functionality and structure	28
4.1.1 Structure of packages	28
4.1.2 Controllers and main functions	28
4.2 Initial Identification of Views in Web application	29
4.3 Initial identification of Activities in Android application	31
4.4 General appearance of Web application	32
4.5 Users Authentication and Registration	35
4.5.1 User's authentication	35
4.5.2 User's registration	36



4.5.3 User's profile	39
4.6 Job booking console	40
4.6.1 The Google Map	41
4.6.2 Couriers list	43
4.6.3 Collection and delivery addresses	44
4.6.4 Jobs list and create new booking	45
4.7 Android application and REST services.	47
4.7.1 REST services	47
4.7.2 Android application asynchronous tasks.	48
4.7.3 Geolocation	49
4.7.4 MainScreen Activity	50
4.7.5 AllJobs Activity	50
4.7.6 ShowJob Activity	51
<b>5. Testing</b>	<b>52</b>
5.1 Code testing environment configuration	52
5.1.1 Dividing application into profiles	52
5.1.2 Set up Spring to use JUnit tests	52
5.2 Code tests	53
5.3 User tests	54
5.4.1 Registration process	57
5.4.2 Booking process	57
5.4.3 Delivery process	58
5.4.4 Overall experience	58
5.5 Errors observed during testing	59
<b>6. Evaluation and Conclusion</b>	<b>63</b>
6.1 Overall results	63
6.2 Project weaknesses and strengths	63
6.3 Project future improvement	64
6.3.1 Improvements for web application	64
6.3.2 Android application	65
6.3.3 Unimplemented behaviour	65
6.4. Conclusion	66
<b>7. References</b>	<b>67</b>
<b>8. Appendix A - Test results</b>	<b>69</b>
8.1 User tests	69
8.1.1 Client test report 1	69
8.1.2 Client test report 2	70

8.1.3 Client test report 3	71
8.1.4 Courier test report 1	72
8.1.5 Courier test report 2	73
<b>9. Appendix B - Setting up a workspace</b>	<b>74</b>
9.1 Configuration for web application	74
9.1.1 General environment settings	74
9.1.2 Controller configuration	75
9.1.3 View and Tiles configuration	76
9.1.4 Security presets	78
9.1.5 MySql connection, Hibernate, DAO and services	79
9.1.6 Web-flow configuration and purpose	82
9.1.7 JSON data request	83
<b>10. Appendix C - Instructions to run the code</b>	<b>84</b>

# 1. Introduction

This report describes design and development of web-based system to book a self-employed courier [described in chapter 1.2 “Terminology”] for a delivery job. The developed system has two major role players:

- 1) Customer [described in chapter 1.2 “Terminology”] - establishes booking process, has access to existing courier feedbacks, traces a booked job and gathers evidence that the delivery is completed successfully.
- 2) Courier - must provide a tool to receive jobs, to describe courier status (working, not working, on delivery etc.), represent his location, to collect the proof of delivery, translate it in digital format and save for later use. It must also provide an option to organise accounts on previous jobs.

The project describes the weaknesses of being a self-employed courier in a delivery industry and propose and describe a prototype solution. It consists from web and mobile application developed to perform a courier job flow [described in chapter 1.2 “Terminology”].

## 1.1 Aim and objectives

According to research done in chapters 1.2 and 1.3 of the project proposal, there are 100 000 to 130 000 self-employed couriers in the delivery industry in UK for the year of 2013, and due to online shopping industry expansion, the number is continuously growing. This is a significant amount of role players in the delivery business.

However, the culture of the delivery process has been changed. Very rarely a customer would look through advertisements, trying to find a courier, giving him the trust of the content of the parcel and at the end of the job, receiving a paper format signature as proof of delivery. Instead, customers look through well-developed fast booking systems of major delivery companies like DHL [22], UPS [23], Courier Systems [24], etc. and make their job order through them. Customers would trust the well-known company better than an individual courier. Such companies are recruiting self-employed couriers as additional workforce and adjusting the amount of received jobs according to the amount of hired self-employed couriers [according on research in the project proposal, chapter1]. Therefore, all self-employed couriers (with rare exceptions) are tightened up with delivery companies. They are dependent from them and furthermore, companies are monopolising prices, adding costs and ruining nature of self-employed courier as a subject.

Reason for such a cripple hybrid situation with self-employed couriers are:

1. Lack of common online or mobile booking system for self-employed couriers.
2. Lack of customer trust to self-employed individuals.
3. It is impossible to trace job status and courier’s location.
4. Difficulties to obtain proof of delivery.

In this project, I developed a prototype online booking website that resolve all these issues - organise courier and customer accounts, use android device as a mobile equipment replacing PDA devices [Personal Digital Assistant, described in project proposal chapter 1.5]. The smartphone gathers data of courier's location and the job status. Feedback system is evolved to gather history of courier activities.

The scale of the project is adjusted according to demand where top level is presumed for 150 thousand couriers carrying about 30 jobs a day (according to research from proposal chapter1.2).

## 1.2 Terminology

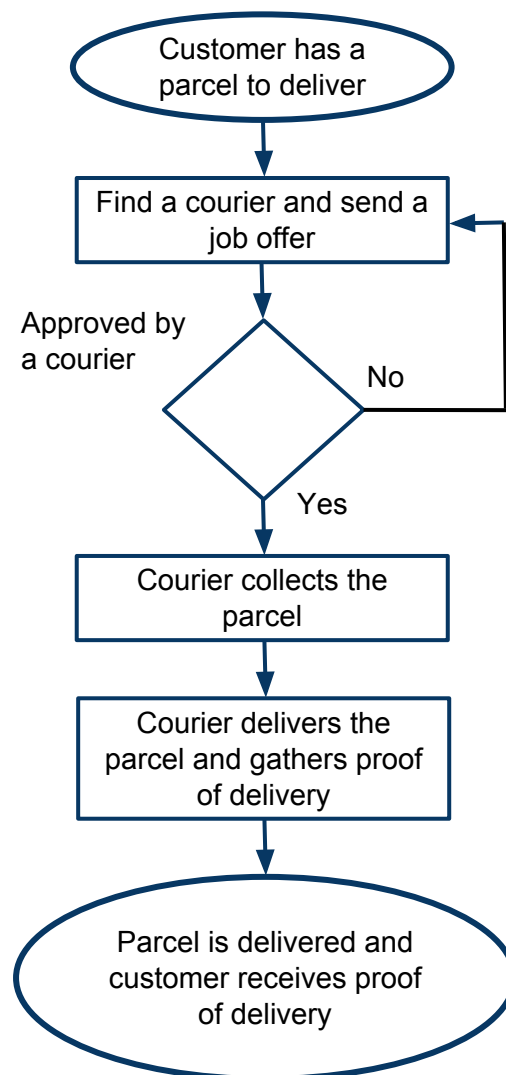
*A customer:* A private person or company who has a single or many *subjects of delivery* to be sent within an area that is agreed with a courier in the scope of a single booking. It also can be a commercial delivery company that would be interested to use self-employed couriers as additional workforce, but this is not considered as an option in the scope of this project.

*Subject of delivery:* It can be any size and amount of parcels our letters.

*A self-employed courier:* The person who delivers a subject of delivery. He works independently and has all necessary legal obligations for doing so (driving licence, insurance etc.).  
Couriers to do their job are using either push bike, motorbike, car or van.

*A delivery job life circle:* The delivery job life circle is a four step process. It involves following steps:

1. Customer finds courier and sends him a job offer.
2. Courier is either Accepting or Rejecting the offer.
3. Courier has completed collection of a parcel.
4. Courier has completed delivery and gathers the proof of delivery.



### **1.3 Application type and approach**

To deliver an easy access to a developed system, it must be a web-based online application. Instead of developing standalone application, that requires a platform compatibility and additional installation, this project will utilise a web browser as a user interface. Resulting product must be an easy to use and self-explanatory.

The mobile application is developed in order to achieve a courier location and status trace. It also includes functionality for couriers to receive new jobs. In the scope of the project, it is an android application only, however for a real life project the I-Phone application also has to be developed. Courier booking is only possible from the website.

### **1.4 Assumed knowledge**

It is assumed that a reader has knowledge, and there is no explanation when fundamental structures of following subjects are used:

1. The concept of world wide web, interaction of client and server platforms, passing parameters, sessions, etc.
2. HTML - marking language, including web forms, requests and response.
3. CSS - cascading styles sheets for formatting HTML tags.
4. JavaScript - for client side programming.
5. Java Servlets - java approach of server side programming.
6. Spring MVC - framework to develop a website by using MVC pattern and letting utilise redeveloped tools.
7. Android development platform - for developing android application.

### **1.5 Structure of report**

In the Background chapter is a description in the greater detail about the platform and technologies used. It includes Google maps, Amazon web services, RESTful services for Android data exchange, Android connectivity issues and JQuery Java Script libraries.

Analysis and design chapter has an analyses of the structure and components of both - Web and Android applications. There is a description of the main goal of the project, use case models, class identification and relations, used design patterns, defined general functions, views and a database design.

Implementation chapter contains implementation and behaviour of components of web and mobile applications. Detailed description of key aspects of the applications code. Description of end user behaviour.

In the following chapter, code testing, user testing and observed errors are discussed. This is followed by the evaluation chapter where it is analysed, if project meets it's objectives, weak and strong points of application. The chapter is concluded with the list of possible improvements in the future. The last chapter of the report is a conclusion.

## 2. Background

Here are described problems and solutions of courier activities. It is followed by a list of technologies that further are not going to be explained in more details. They are used for both - web and android applications.

### 2.1 Over all concept of courier activities

During the period of 2011/02 till 2011/08 I worked as a bike courier in London for “E-Couriers” Ltd [1]. Throughout the period of half year, I had consultations with colleagues holding very different positions in this company. Therefore, I gathered full understanding of weaknesses and strengths in courier’s industry. That information has helped me to develop the whole design for this project. Through the evaluation process, I used a person working for this company to test the developed prototype booking system and a mobile app for self-employed couriers.

The courier’s job flow [described in chapter 1.2 “Terminology”] has been fully explained in the project proposal in chapter 1.4.

Relating issues that were described in this project report in chapter 1.1 and the key points and ideological solutions used in developed prototype system are:

- 1) *Lack of common online or mobile booking system for self-employed couriers* – resolved by developed web-based online booking system where every courier and customer have a separate account to serve their needs.
- 2) *Lack of customer trust to self-employed couriers* - every customer have to leave a feedback. According to that, couriers gain their reputation.
- 3) *It is impossible to trace job status and courier’s location.* - couriers use smartphones which updates their location and status.
- 4) *Difficulties to obtain proof of delivery* - proof of delivery collected from a smart phone as a bitmap image file of scanned signature or photo of a delivered parcel.

### 2.2 Used technologies

MySQL database is used for a storage in the developed system, the reason was explained in the project proposal in chapter 3.4. Spring MVC is used as a development framework [project proposal chapter 3.3]. Jelly Fish 4.1 with a minimum requirement of version 2.2, serves as an android platform [project proposal chapter 3.5.2].

---

#### 2.2.1 Google maps

Google Maps is a desktop and mobile web mapping service application and technology provided by Google, offering map perspectives, as well as functions such as a route planner for traveling by foot, car or bicycle. There are also maps embedded on third-party websites via the Google Maps and a locator for urban businesses and other organisations in numerous countries around the world [2].

Comparing with other providers, like Apple, Bing, Nokia etc., Google provides advanced features: powerful routing (including for walking and bicycling), Street View, 3D buildings, weather, and traffic information. Some of these features are unique to Google and can be used in future to extend the project.

It is possible to embed Google Maps site into an external website, where site specific data can be overlaid by using the Google Maps API. API v.3 the latest version will be used in this project. Longitude and latitude coordinates are used to get a geolocation.

---

### 2.2.2 Project hosting on Amazon Web Services, scalability and security

The web application for hosting uses one instance of Amazon Elastic Compute Cloud service (Figure 2.1). Amazon web services provide resizable computation capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon EC2's simple web service interface allows to obtain and configure the capacity. It provides with complete control of computing resources, (Figure 2.3). Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing to scale capacity quickly, both up and down, when the computing requirements change. Amazon EC2 changes the economics of computing by allowing to pay only for capacity that is actually used. Amazon EC2 provides the tools to build failure resilient applications and isolate from common failure scenarios [3].

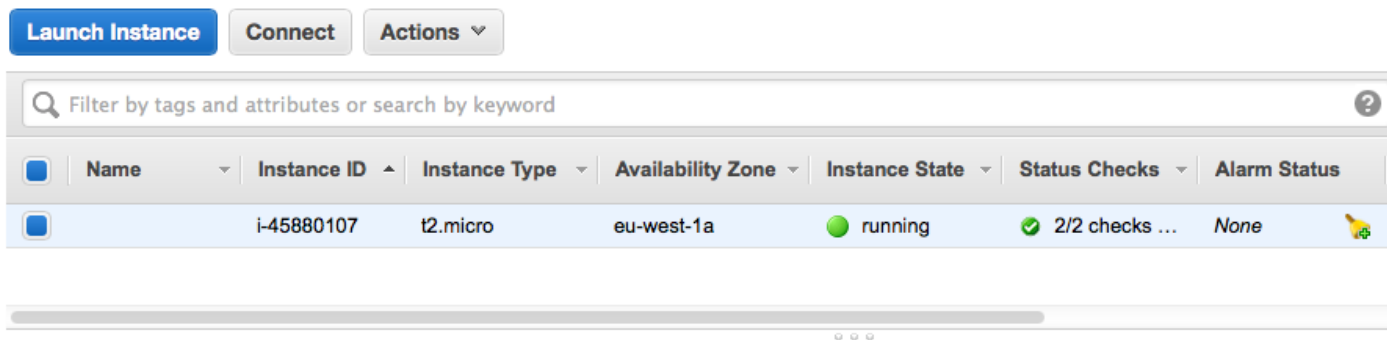


Figure 2.1. Running instance is displayed in AWS EC2 console.

Amazon EC2 enables to increase or decrease capacity within minutes. It is possible to commission many instances simultaneously. Settings can be configured manually or our application can automatically scale itself (on demand instances - let you scale compute capacity by the hour with no long-term commitments) up and down depending on its needs.

Launched single instance for a project is a Linux Ubuntu Server 14.04. It has installation of Apache Tomcat 7 web server and MySQL5. Root access to the instance is established via console. The public IP address is 54.77.11.0 and DNS (domain name service) `ec2-54-77-11-0.eu-west-1.compute.amazonaws.com`.

Amazon Simple Storage Service (Amazon S3) has been deployed to store image and utility data for used instance, (Figure 2.2). It can be used for additional requirements after initial evaluation of the project.

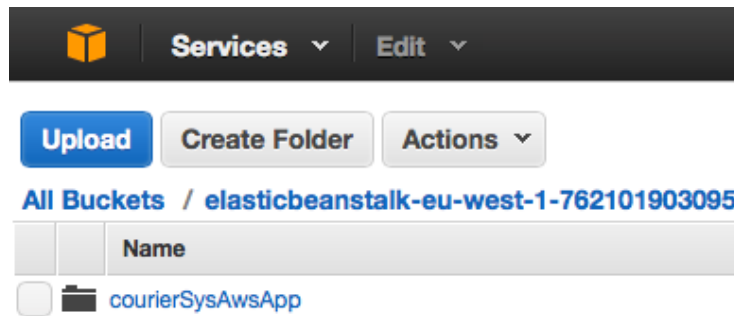


Figure 2.2. AWS Running storage instance.

Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned. The service runs within Amazon's proven network infrastructure and data centres. The Amazon EC2 Service Level Agreement commitment is 99.95% availability for each Amazon EC2 Region. It depends on the configuration which services are posed for internet [3]. Amazon EC2 fully satisfy project needs for designated scale.

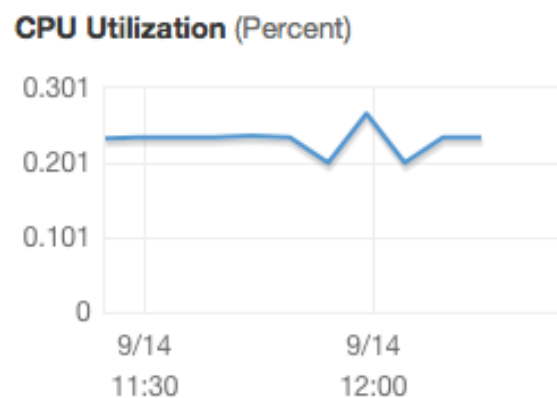


Figure 2.3. Example of AWS resource utilisation monitoring diagram.



### 2.2.3 REST services for android data exchange

Representational State Transfer (REST) [4] is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web [4].

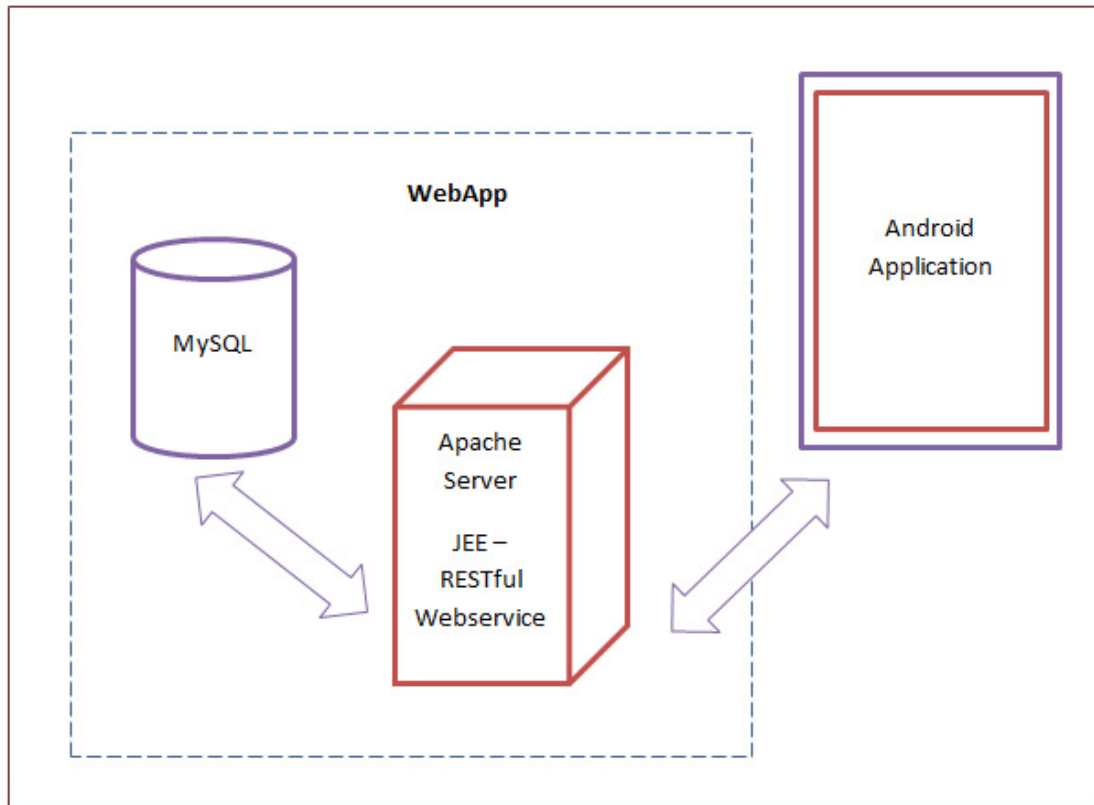


Figure 2.4. Data communication from mobile device to database via web application.

It is a common practice to use REST service to establish data communication from database to mobile device via web application. The data flows from android application to web site and finally is stored in MySQL database. The mobile device is calling HTTP (or SHTTP) request (it can be GET, POST, PUT, DELETE) with or without parameters. Script, executed after HTTP request, returns JSON data structure. In case if request response is data, then it returns an array of results. If request is for creating new entry in database, or update existing database entity, JSON return could be a JSON array with parameter “success” true or false.

---

### 2.2.4 Android application connectivity issues

The common problem with android data communication is connectivity. In this project, android application is periodically updating data with asynchronous background request where connectivity is crucial. Each request uses a separate thread and working in the background. The following code sample is pre executed before every request. Therefore a lack of connectivity is not going to crush or stop the application, only warning must pop up if the communication is lost for a significant period and data is out of date.

```
public boolean hasActiveInternetConnection(Context context) {

    ConnectivityManager cm = (ConnectivityManager)context.
    getSystemService(Context.CONNECTIVITY_SERVICE);

        Networkinfo netinfo = cm.getActivityNetworkInfo();

        if (netinfo != null && netinfo.isConnected()) {
            return true;
        }

    return false;
}
```

---

### 2.2.5 JQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax more simple with an easy-to-use API that works across a multitude of browsers [5]. In this project, JQuery is used to make an asynchronous request to REST services and update HTML DOM objects accordingly of data received from JSON array.

Example:

```
function updatePage() {
    $.getJSON("<c:url value='/getcouriersjson'/>", updateCouriers);
}
```

**updateCouriers** is a function called after a request. It has an argument **data**, which holds returned values.

### **3. System Analysis and Design**

The system is developed more similar to the Rapid application development (RAD) principles. This software development methodology favours iterative development. There is not a large amount of up-front planning. It allows software to be written much faster. The rapid development process starts with the development of preliminary data models that are described in this chapter. It includes description of the main goals of the project, use case models, class identification and relations. They are followed by description of used design patterns, defined functions, views and database structure.

#### **3.1 Problem description**

Clients have to be able to book a job. Then they have to choose a collection and a delivery address for the job. Some delivery or collection addresses can repeatedly be used (for example customer's home address). Consequently, clients have to choose an available courier. The customer should be able to check the courier's previous feedbacks and be able to leave a feedback after the job is completed.

In this application, a courier should see all his previous jobs. A courier has to be able to present his location and status and be able to receive jobs and change their status according to a committed stage.

#### **3.2 Initial system analysis**

It is clear from the statement that application has to be developed in two separate parts - Web application and Android mobile application. Therefore, Android application uses REST services from the web application. It relays on it's design (for example if a courier have to present his location, then the REST service in web application have to handle it).

From problem's definition is clear that application is used by two type of users:

1. Courier.
2. Customer.

As much as every application has a person who's duty is to maintain the application, there is a third player who could be an "Administrator". For example, his duty could be to delete bad language from client feedbacks. It is also an option, that system could have more than one administrator, with different levels of access. During the scope of this project, such player won't be implemented. However, there should be a used design that allows them to be added in a modular way, without rewriting the code.

### 3.3 High level goals

Tasks which must be achievable in a web application

#### ***Unauthorised user tasks:***

- 1) Being able to register either as a Customer or a Courier

#### ***Administrator tasks:***

- 1) Being able to authorise themselves (in the scope of the project)

#### ***Customer tasks:***

- 1) Being able to authorise as a Customer
- 2) Make bookings by choosing collection, delivery addresses and a courier
- 3) Leave and read Courier feedback(s)
- 4) Trace courier location and status
- 5) Trace booked job status
- 6) Delete jobs that no more valuable
- 7) Add, delete, edit a collection address
- 8) Add, delete, edit a delivery address
- 9) Update his/her own account details

#### ***Courier tasks:***

- 1) To be able to authorise as a Courier
- 2) To be able to receive a delivery jobs (via an android application)
- 3) To check his/her previous jobs
- 4) To represent his/her location, status and availability (via an android application)
- 5) To change status of committed jobs (via an android application)
- 6) Update his/her account details

All predefined tasks are the application's initial state. Within the project expansion, the amount of tasks will increase. The initial state of development has a list of challenges achievable until the first evaluation. Therefore, overall design must be modular with a high capacity of prediction in any area of possible growth. In the scope of this project, only initial stage of the application will be developed.

### 3.4 Use Case Models

The first three charts is used by web application. As defined in section 3.2, the application has an Undefined user and two main user types - Customer and Courier.

Unidentified User:

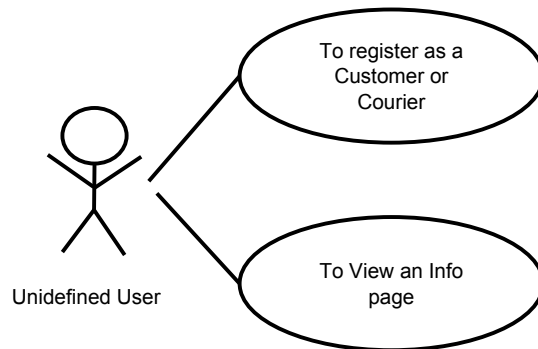


Figure 3.1. Undefined user use case diagram for the web application.

Undefined user only have two choices - register or visit “info” page.

Customer:

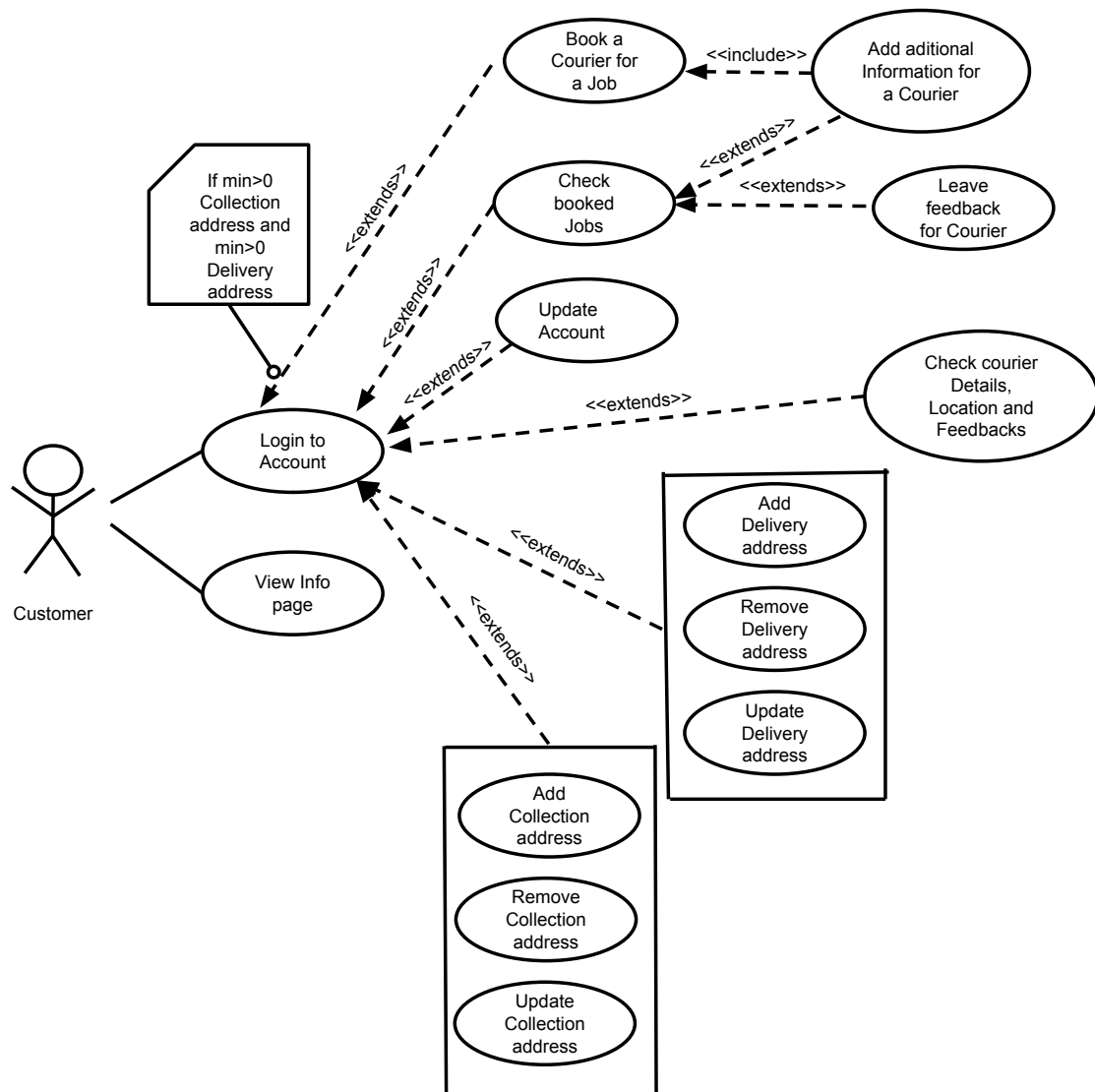


Figure 3.2. Customer's use case diagram for the web application.

The customer has two choices - login to the application or view "info" page. If registration is done, a customer can check booked jobs, update his account, check couriers details, including their locations and feedbacks. He can add, remove, update or view delivery and collection addresses. If at least one collection and delivery address is entered, he can book a courier for the job. Customer can add additional information to the booking by editing or creating a new job. By editing a booked job the customer can leave a feedback for it.

Courier for the Web application:

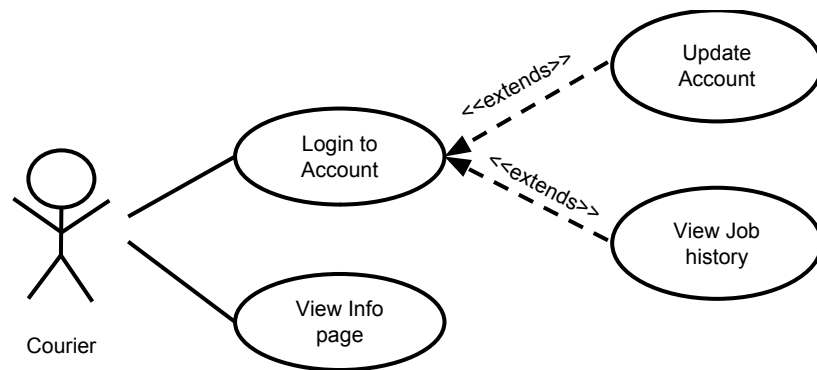


Figure 3.3. Courier's a use case diagram for the web application.

The courier has two choices - login to application or view “info” page.  
After registration he can update his account or explore his job history.

Courier for Android application:

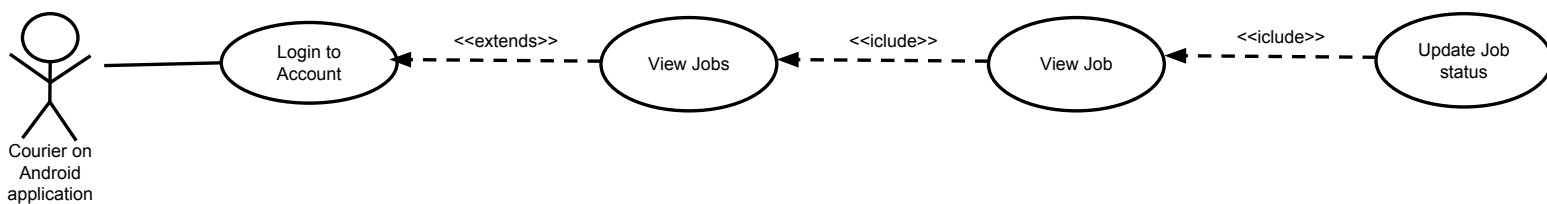


Figure 3.4. Courier's use case diagram for the Android application.

After login into the Android application a courier can view jobs. By selecting any specific job he can view his details. Further he can change his job status.

### 3.5 Conceptual Class(es) identification

It is possible to identify the Conceptual (role player) classes for the web application if nouns are extracted from previously gathered information.

First two main objects are:

- 1) Customer
- 2) Courier

It is obvious that these two concepts have common properties. Both are persons with Name, Surname, etc. In order to optimise data information flow, it is recommended to keep common records in a single table. Therefore, the third concept gathers and stores information from the both of them.

- 3) User

As mentioned in chapter 3.2, there must be a way how to add another player to organize access for maintenance.

- 4) Authorities

The central object of whole system is delivery is **Job**

- 5) Job

Every job has a collection address and a delivery address. Even if presumably some jobs have the Collection address same as other Delivery addresses, to avoid confusion and to reduce an amount of data in one table, it is better to separate them.

- 6) Collection address
- 7) Delivery address

Every **Job** has its **Feedback**

- 8) Feedback



### 3.6 Context Class Diagram design

The class design diagram is divided into separate parts to improve a better readability.

The first entity is a **User**. To achieve authentication application it must employ User, Customer, Courier and Authority classes [from chapter 3.5].

It is clear that Customer and Courier classes are extension of User class, therefore generalisation relationship is taking place.

Every user has some kind of authority (role in application). That is a life time dependency between both classes and represents a composition relationship.

The Courier and Customer classes are types of **Authority** but dependency are weaker. An attribute of dependent class is an instance of **Authority**.

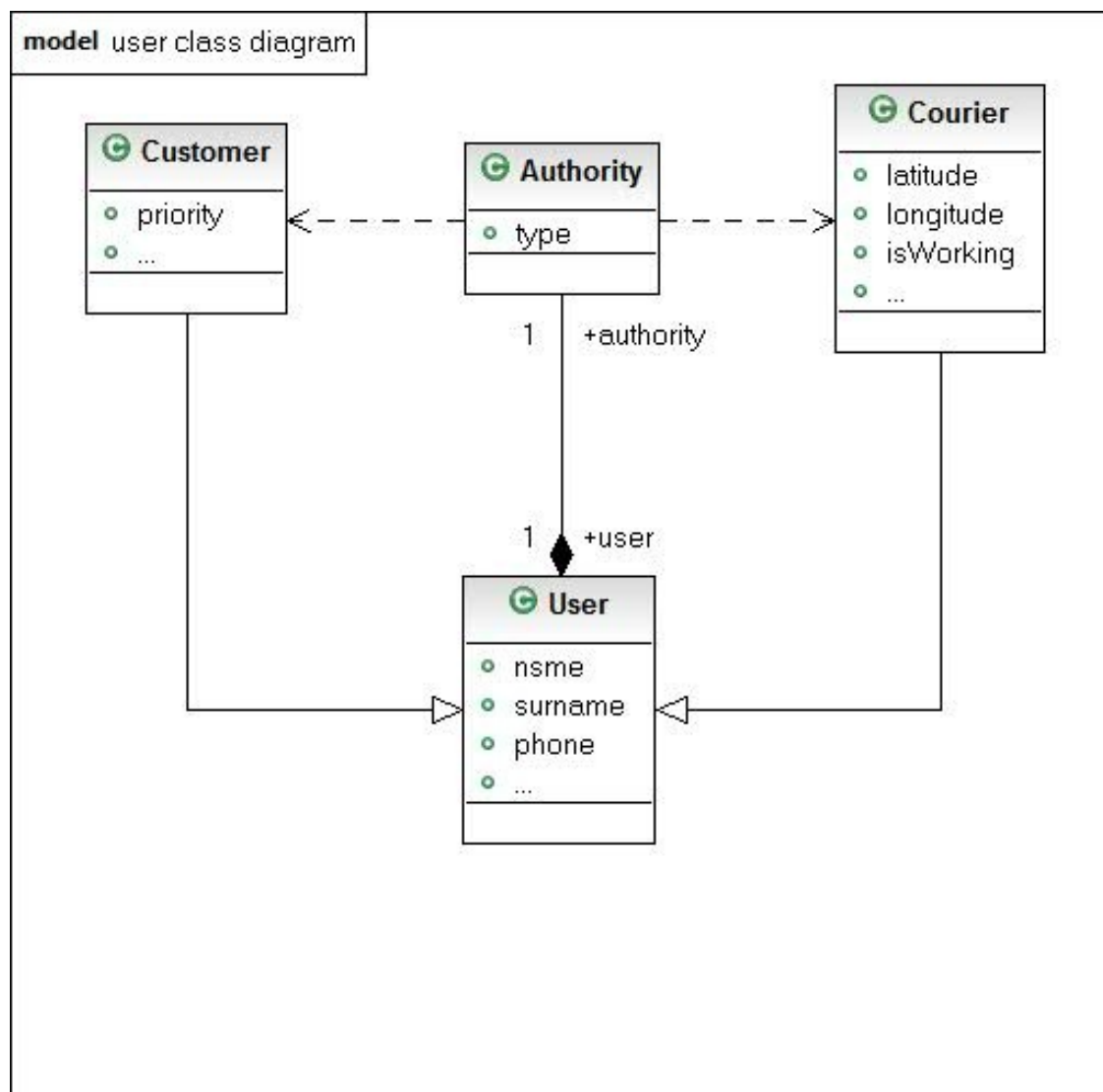


Figure 3.5. User class diagram.

The central entity in application is a **Job**. Every job has at least one collection and one delivery address. This could be any address entered by a user. This represents Aggregation relationship.

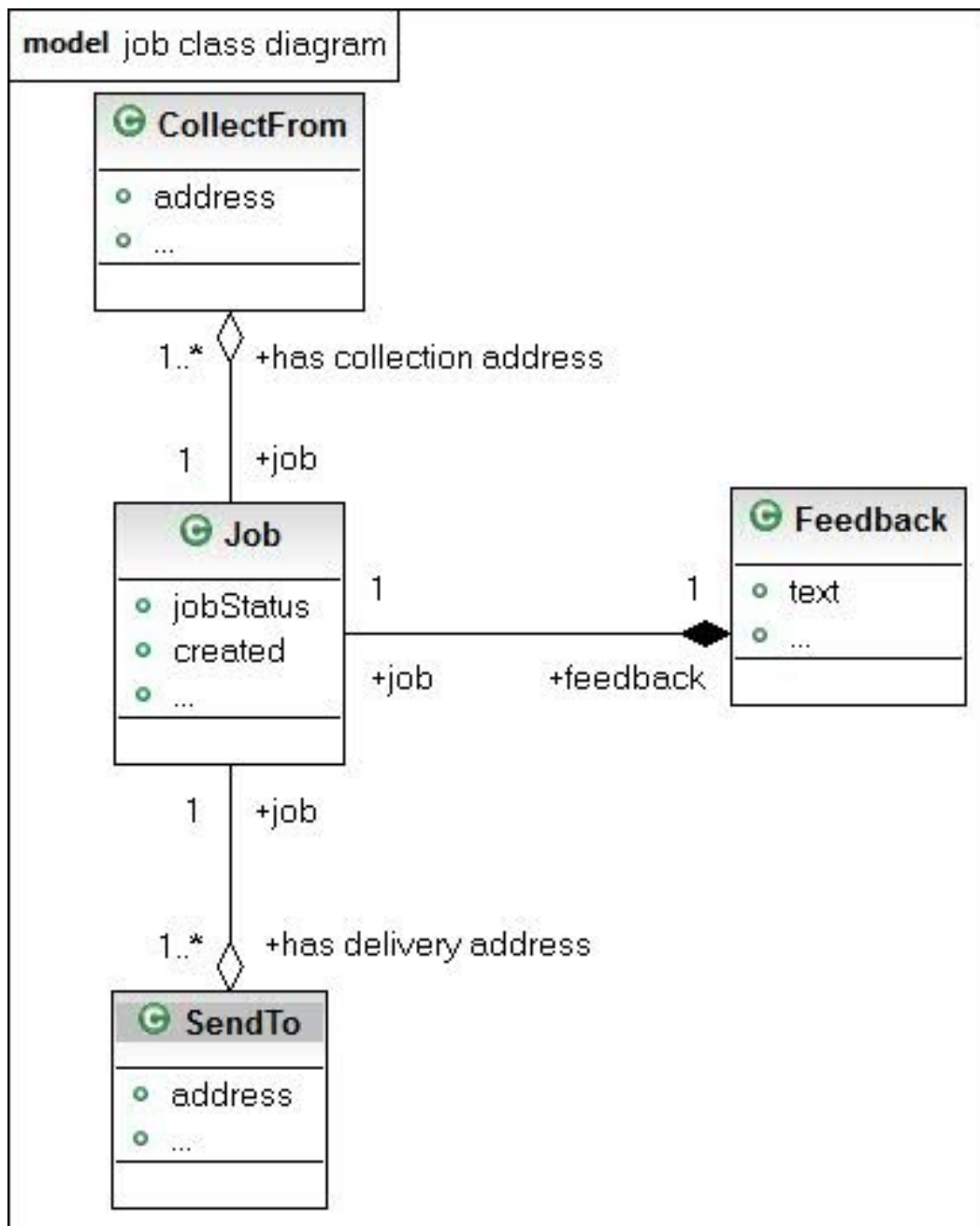


Figure 3.6. Job class diagram.

Every **Job** has only one **Feedback**, therefore it is a Composition relationship.

By combining both diagrams together, there have to be added Aggregation relationship between **Courier** and **Job**, and **Customer** and **Job**. **Courier** can have any **Job** and **Customer** can have any **Job**. **Couriers** have **Feedback** for every **Job** and every **Customer** must leave the **Feedback** for every **Job**. That is a weaker form of dependency represented in a diagram.

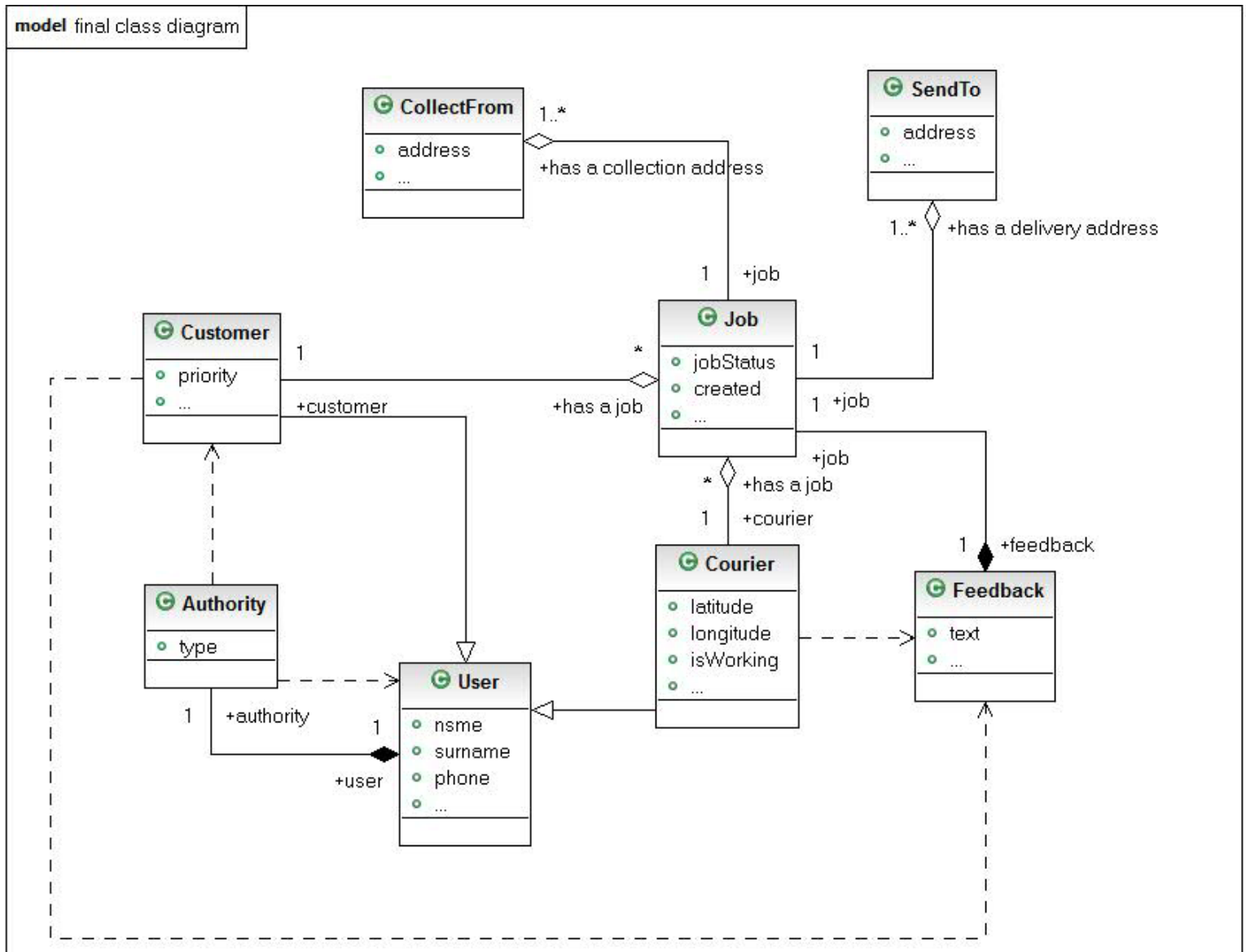


Figure 3.7. Final class diagram.

### 3.7 Identify Design Patterns for web application

A design pattern is general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into the code. The web application deploys the widely used MVC as a structural and DAO as data access pattern.

**Model View Controller (MVC):** the project is developed on Spring MVC framework that is designed to commit every request through one Dispatch servlet via Controller which forwards results to designated View. This is by default MVC pattern behaviour. Although MVC pattern is so common and widely used that is not going to be explained in this report.

**Data Access Object (DAO):** t to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data. The DAO implements the access mechanism required to work with the data source. This is the place to use Hibernate functions [7]. The business component that relies on the DAO, uses the simpler interface exposed by the DAO for its clients. In the project, this component is called service. The DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes [6].

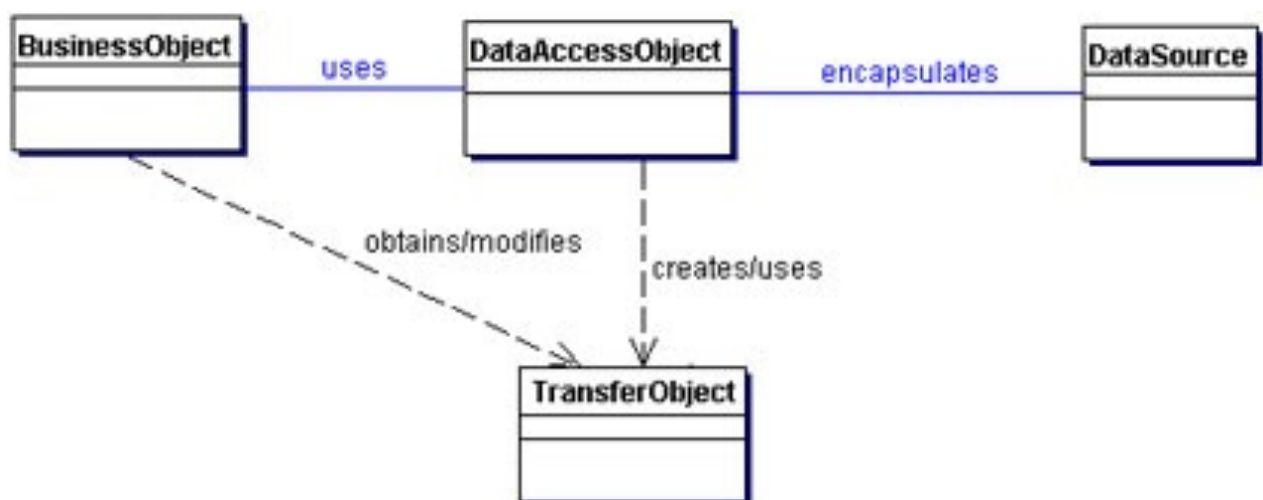


Figure 3.8. DAO pattern diagram.

#### *BusinessObject*

The BusinessObject represents the data client. It is the object that requires access to the data source to obtain and store data. In the project, this is one of context class objects or a list of the objects.

#### *DataAccessObject*

The DataAccessObject is the primary object of this pattern. The DataAccessObject abstracts the underlying data access implementation for the BusinessObject to enable transparent access to the data source. The BusinessObject also delegates data load and stores operations to the DataAccessObject [6].

### *TransferObject*

This represents a Transfer Object used as a data carrier. The DataAccessObject uses a Transfer Object to return data to the client. In this project, Transfer object function is delegated to three services:

1. users service - all data flow related with users (login, registration etc.)
2. bookings service - all data related to jobs
3. address service - only serves collection and delivery address objects

## **3.8. Database design**

After database analysis in the project proposal chapter 3.4, it is decided to use MySql database. Every contextual class represents one table.

Four tables are needed to establish an authorisation: Users, Couriers, Customers and Authorities. They all have one to one relationship because one user can only be just one authority, and it can be either Courier or Customer (in the scope of the project).

Table users:

A field "username" is a primary key. The field "iscourier" distinguishes couriers from customers. The field "enabled" represents a situation if a user is authorised to use the system after registration. Can be used in case for email conformation after registration.

Table couriers:

A field "username" is a primary key. A field "isworking" represents a state of the courier, "isavailable" shows if he is currently doing a job.

Table client:

A field "username" is a primary key. A field "bankid" is left for future extension to add an online banking. This option is not going to be implemented in the project. Text fields are used for utility information and debugging.

Table authorities:

*A field* "username" is a primary key.

An application is not going to delete any entries therefore field "visible" represents the publicity. Date and time fields are representing time for certain activity.

Every Job have one delivery and one collection address. It also have one Feedback. That are one to one relationships. However, every Customer can have many Jobs as well as every Courier can have many Jobs. They can have many deliveries and collection addresses. Couriers can have many Feedbacks and Customer can leave many Feedbacks. Those tables have “one to many” relationship.

Table jobs :

A field **id** is a primary key.

A field **sendto** is a foreign key for table “Sendto”.

A field **sendfrom** is a foreign key for table “Sendfrom”.

A field **courier** is a foreign key for table “Couriers”.

A field **client** is a foreign key for table “Clients”.

Table feedbacks:

A field **id** is a primary key.

A field **courier** is a foreign key for table “Couriers”.

A field **client** is a foreign key for table “Clients”.

A field **job** is a foreign key for table “Jobs”.

Table sendto:

A field **id** is a primary key.

Table sendtfrom:

A field **id** is a primary key.

The full list of database table fields and relations are displayed in Figure 10.

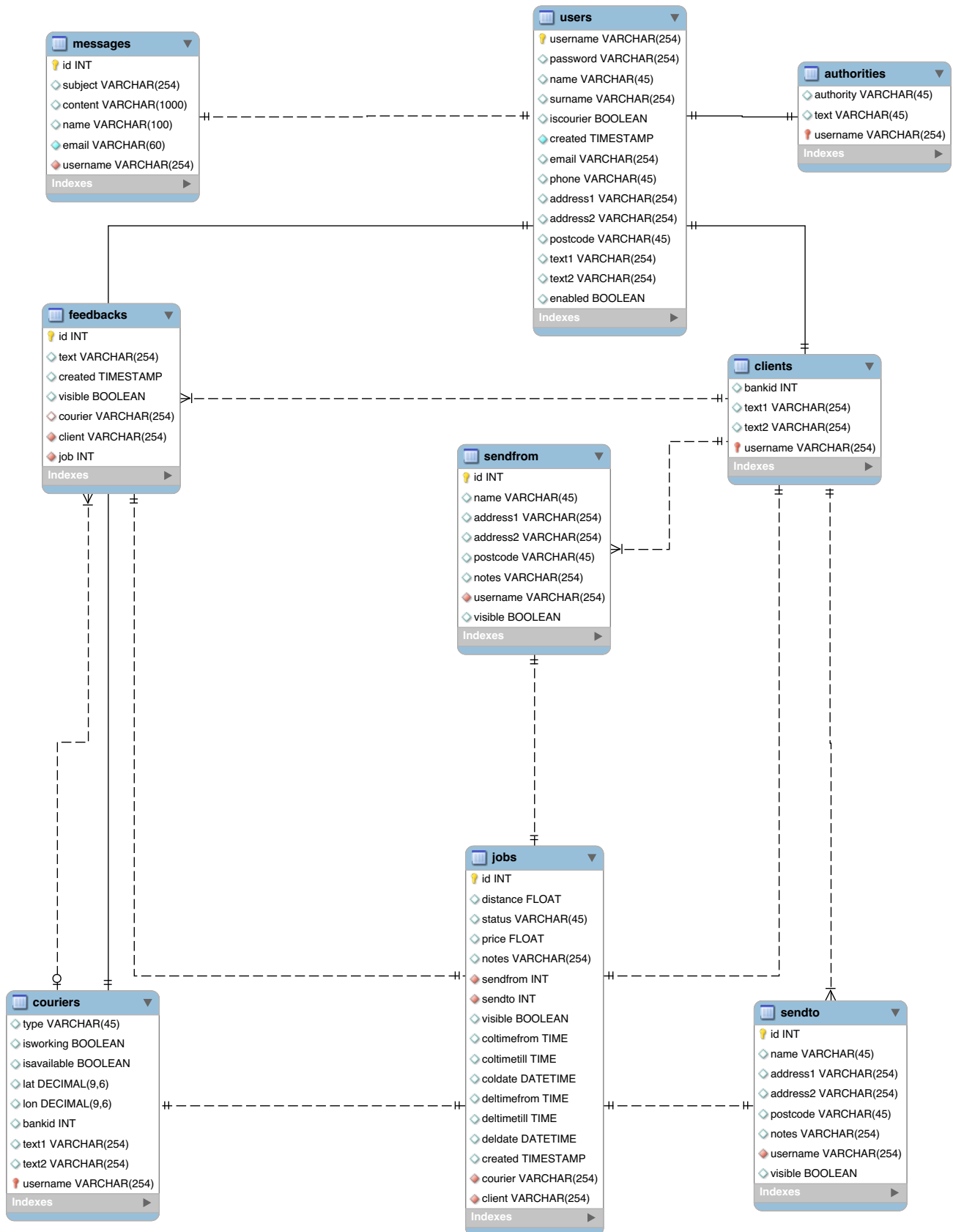


Figure 3.9. The database table relationship diagram.

## 4. Implementation

The initial definition of application components are presented in the beginning of this chapter. It is followed by steps of the actual development process of both - Web and Android Applications. It begins with the description of programming the login process then moves to user controls, job booking and maintaining. The last part is delegated to the Android application development. All configuration and settings of the web application development environment components (Tiles, controllers, web-flows, JSON server etc.) are used from various tutorials (mainly “The Java Spring tutorial” from John Purcell [25]), and Spring documentation. They are explained in details in appendix B.

### 4.1 Initial identification of functionality and structure

Because web application is maintaining REST functions for the Android application those both are not going to be separate.

---

#### 4.1.1 Structure of packages

The project should maintain separate project packages for unrelated tasks and concepts.

- 1) Package for the project configuration files
- 2) Package for custom validators
- 3) Package to accommodate services. Initially there will be three services: addresses, bookings and users
- 4) Package for contextual classes and DAO request functions.
- 5) Package for controllers

---

#### 4.1.2 Controllers and main functions

The project separates the following controllers:

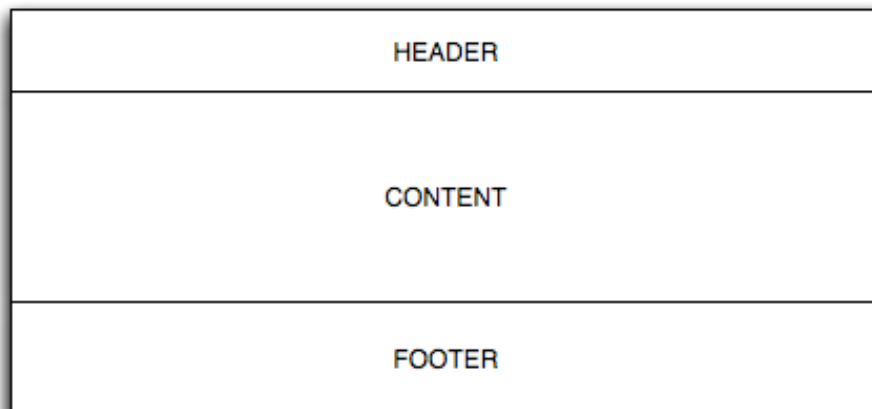
- 1) Home controller - to render main and info page and first page for couriers to follow their job list
- 2) Login controller - to complete login and logout actions, edit user account.
- 3) Booking controller - main controller to establish new job, edit, remove, update job and generate main console.
- 4) Courier controller - create update courier details (including feedbacks), generate JSON data for asynchronous requests to change courier date in the main console.
- 5) Sendto controller - separate controller to create, update and edit delivery addresses. The main reason is to separate this controller is to increase code readability and release amount of functions inside one controller.
- 6) Sendfrom controller - separate controller to create, update and edit collection addresses.
- 7) Rest controller - controller to handle android REST requests. Update android job list, update courier's location, update courier's and job's status, authorise android user.



## 4.2 Initial Identification of Views in Web application

To allow defined page fragments that can be assembled into complete pages, at runtime this project will be using Apache Tiles. These fragments (or "tiles") can be used as "includes" in order to reduce the duplication of common page elements, or embedded within other tiles to develop a series of reusable templates [8].

Complete set of tiles within one page is a layout. This project initially is going to use only one configuration of layout which will consist of header content and footer.



Footer holds a copyright information and remain the same, header contains login information and have two states:

- 1) If user is authorised
- 2) If user is not authorised

Content is the main "tale" which, according on use case diagrams, have following states:

every user must have access to two views

- 1) main view - holds links to registration and downloads android application
- 2) info view - general information

courier or customer registration is separated in three parts. The first view is a common courier and customer information, so one view for both

- 3) user registration view

separate view for customer related information and courier related information

- 4) customer related information view
- 5) courier related information view

separate view for additional information for customer and courier. Most fields are added after evolution

6) additional customer related information view

7) additional courier related information view

page to edit user information uses one form for all parts, so it needs separate view

8) edit user view

page for user authorisation

9) login view

main page for couriers to see the previous jobs

10) job list view

the main page for a customer is a map and presented courier locations, status and availability. It holds such information as where to check booked jobs and commit a new booking. Keep track on collection / delivery addresses. This view is a main console where to work with an application.

11) in this project it is called availability view

page to add feedback for the job

12) add feedback view

page to view specifically courier who is doing a current job, his location on the map

13) courier details view

to update job instructions

14) update job view

create new or edit existing collection address

15) update collection address view

create new or edit existing delivery address

16) update delivery address view

utility view to display error messages, connection problems etc.

17) error view

### **4.3 Initial identification of Activities in Android application**

In android application the “main” activity is establishing authorisation. It has a view for the entry point into application

1) login view

After authorisation a courier must be able to see list of jobs. In the android application it employs two views for one list activity. One for jobs list and one for single item

2) jobs list view

3) list item view

Application must have separate activity to present the job details and change it's status

4) show job view

## 4.4 General appearance of Web application

The project web application is accessible from the public domain at <http://ec2-54-77-11-0.eu-west-1.compute.amazonaws.com:8080/CourierSysAWSv8/>

The first page of web application holds links to Courier and Customer registration, and a link for a general information page - "Information". It also has download links for four types of mobile applications. iPhone - Customer and Courier mobile app (both are not included in the project), a download link for Customer android mobile application (not included in the project) and a download link for Courier's mobile android application. This link has a basic "apk" file source, because Google Marketplace, that is commonly used for this purpose, has financial costs involved.

Links and submit buttons in the site has a light blue background and dark blue text colours.

Every page in the application has a yellow square in the left side with general information and guide lines for the purpose of page components.

Top of the page holds information about user's authentication status.

Site is tested with Safari 6.0.7. Microsoft Mozilla has few known issues.

You have to be registered to use an application.

There are two types of registration - one for couriers and one for clients. Both registrations are 4 step process to fill the related information.

To use an existing registration to log-in as a courier:  
username: **testcourier1**  
password: **testcourier1**

To log in as a client:  
username: **testclient1**  
password: **testclient1**

Courier's mobile application can be downloaded from the related link.

After authorisation "Bookings" link appears for clients and "My Jobs" for couriers

After authorisation "Edit Account" link appears at the top of the page to edit user's details.

Menu "Information" holds only general information about the project.

Hello testclient1! You are authorized as a customer. [Edit Account](#) [Logout](#)

MSc ACT Course Project. Couriers Online System. Proposed to organize couriers online job flow, including registration, tracing status and proof of delivery.

Registration for Clients

Download Client's App:

  
not included in scope of this project

  
not included in scope of this project

Registration for Couriers

Download Courier's App:

[download link for android app](#)

  
not included in scope of this project

Bookings

Information

© 2013/14

Figure 4.1. The first page of web application.

After authentication application has two main states, one for Courier and one for Customer. The additional button appears on the top of “Information” link in the main page. In Courier case, it is “My Jobs”, and it leads to courier accounts for committed and current jobs, shows in Figure 4.2.

<b>Courier:</b>	Ray Standers, tel: 07562793514, email: ray@ray.com		
<b>From:</b>	Walthamstows library, 91 Hoe street London, E17 3EH		
<b>To:</b>	my work address, 34 Beresfield , E4 3DD		
<b>Notes:</b>			
<b>Status</b>	<b>Created</b>	<b>Price</b>	<b>Distance</b>
Sent to Courier	2014-07-27 16:46:55.0	0.0	0.0

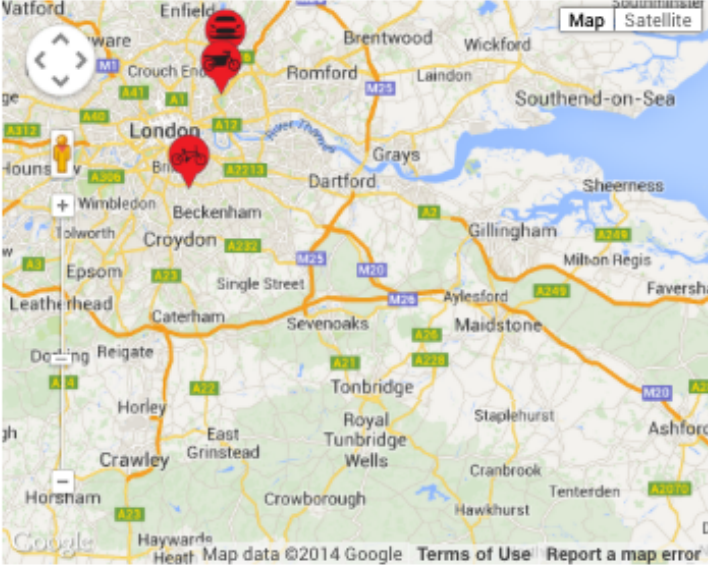
<b>Courier:</b>	Ray Standers, tel: 07562793514, email: ray@ray.com		
<b>From:</b>	Walthamstows library, 91 Hoe street London, E17 3EH		
<b>To:</b>	my work address, 34 Beresfield , E4 3DD		
<b>Notes:</b>			
<b>Status</b>	<b>Created</b>	<b>Price</b>	<b>Distance</b>
Sent to Courier	2014-07-27 16:47:25.0	0.0	0.0

<b>Courier:</b>	Ray Standers, tel: 07562793514, email: ray@ray.com		
<b>From:</b>	area area area area		

Figure 4.2. “My Jobs” page for couriers with a list of his jobs.

If the authenticated person has Customer credentials, the additional link “Bookings” appears, which leads to main jobs booking console, shows in Figure 4.3.



### Available Couriers

Info	Vacant	Nickname
	✓	rckerka01
	✓	testcourier2
	✗	testcourier3

### My Booked Jobs

	Collection Address	Delivery Address	Courier	Job Status	Created
<a href="#">view</a> <a href="#">remove</a> <a href="#">feedback</a>	Euro Parts Totenenham	my work address	testcourier2	Sent to Courier	2014-08-17 19:07:09.0
<a href="#">view</a> <a href="#">remove</a> <a href="#">feedback</a>	Walthamstows library	Birkbeck	testcourier1	Delivered	2014-08-10 16:02:19.0
<a href="#">view</a> <a href="#">remove</a> <a href="#">feedback</a>	njam njam	Birkbeck	testcourier1	Approved	2014-08-13 16:34:50.0

### New Job

Collection Addresses:
Delivery Addresses:
Available Couriers:

Walthamstows library
my work address
rckerka01
To Book

### My Collection Addresses

	Name	Postcode	Address
<a href="#">edit</a> <a href="#">remove</a>	Walthamstows library	E17 3EH	91 Hoe street London
<a href="#">edit</a> <a href="#">remove</a>	Euro Parts Totenenham	EN9 3EF	12 Albion drive Edmonton
<a href="#">edit</a> <a href="#">remove</a>	njam njam	E12 W14	vietinja ko nezin neviens

Create Collection Address

### My Delivery Addresses

	Name	Postcode	Address
<a href="#">edit</a> <a href="#">remove</a>	my work address	E4 3DD	
<a href="#">edit</a> <a href="#">remove</a>	Birkbeck	EC1 1AH	

Create Destination Address

Figure 4.3. “Bookings”, customers jobs booking console.

## 4.5 Users Authentication and Registration

### 4.5.1 User's authentication

According on design, the program has four states of authentication. Unregistered user, Customer, Courier and Administrator, which is not included in this project. The authentication progress is displayed in a web page header.

Unregistered user has a link to “login page”:

[Login](#)

Registered users have a welcome information accordingly of their status (Courier or Customer) and links, to logout or edit profile:

Hello testclient1! You are authorized as a customer. [Edit Account](#) [Logout](#)

The project uses the Spring built-in security system which is designed by using security filters. To get the entry point in the application, the authentication provider configuration must be added to “security-context.xml” file (The full structure of security configuration is described in a chapter 4.1.4).

```
1. <security:authentication-manager>
2.   <security:authentication-provider>
3.     <security:jdbc-user-service
        data-source-ref="dataSource"
        users-by-username-query="select username, password, 'true'
from users where binary username=?"
        authorities-by-username-query="select username, authority
from authorities where binary username=?"
        role-prefix="ROLE_" />
4.     <security:password-encoder ref="passwordEncoder">
5.     </security:password-encoder>
6.   </security:authentication-provider>
7. </security:authentication-manager>
```

User credentials are gathered from **users** table and role added from Authorities table. Tables are joined with “username” field which is a primary key. Password is encoded with SHA-256 algorithm. To get a password in encoded format, “password-encoder” must be included into configuration.

The login form is a customised version of the Spring built in login form. It is located in “/tiles/login/login.jsp” file. The authentication is performed by “*j\_spring\_security\_check*” action

```
<form name='f' action='${pageContext.request.contextPath}/
j_spring_security_check' method='POST'>
```

The custom validator checks for error parameter(s), and if they exist, displays message “Wrong credentials”.

```
<c:if test='${param.error != null}'>
  <span class="error">Wrong credentials</span>
</c:if>
```

#### 4.5.2 User's registration

User's registration is divided into four parts. In the first part (Figure.4.4), user must have to leave credentials (password, username, email). The second part (Figure.4.5) is designed for common information, like phone number, address, etc. The third part (Figure.4.6) is different for Couriers and Customers where they have to leave related information (vehicle type for Couriers or special requirements for Customers, etc). The fourth part (Figure.4.7) is meant to be for banking information (it is not included in the project).

Username:

Name:

Surname:

Email:

Password:

Confirm password:

**Step 2**

Figure 4.4. The first part of registration process.

Phone:

Address:

Address:

Postcode:

**Step 3**

Figure 4.5. The second part of registration process.

Vehicle type:

Other: ...

**Step 4**

Figure 4.6. The third part of registration process.

Please enter your bank details.

Not going to be implemented in scope of this project

**Finish**

Figure 4.7. The fourth part of registration process.



User's information is stored into four tables. The common information is stored in a "Users" table, Couriers related - "Couriers", Client related - "Clients" and access roles are stored in the table "Authorities".

User's registration is a four step process. To allow users jump from one step to another by pressing "back" or "forward" buttons, a Spring web-flows technology is used. The configuration and a setup process is described in a chapter 10.1.6 of Appendix B. The registration has two separate web-flows, one for Couriers registration, located in "courier-reg.xml" file, and one for Customers registration, located in "customer-reg-flow.xml".

The web-flow activities are defined states with an attribute "name". It can be a "view" state or "action" state. A chain of activities are performed according on configuration from xml file.

The example of action state:

```
<action-state id="a_createuser">
    <evaluate expression="userService.saveUser(user, 'CUSTOMER') "></evaluate>
    <transition to="createuserform2"></transition>
</action-state>
```

The example of view state:

```
<view-state id="createuserform" model="user">
    <transition on="createuser" to="a_createuser"></transition>
</view-state>
```

If the action uses a conceptual class model, and in this case it does by storing information into Users table, it must be defined at the beginning of xml file

```
<var name="user" class="com.project.dao.User" />
```

To move from one state to another, the "transition" closure is defined into xml tuple. The attribute "to" defines the "name" of the next state. In this example after action "a\_createuser" web-flow jumps automatically to state "createuserform2". In the case if this is a "view" state, additional attribute "on" defines condition to trigger the next move. Such condition is a form's submission. The form, instead of having defined "action", have to include additional hidden input fields:

```
<input type="hidden" name="_flowExecutionKey" value="$
{(flowExecutionKey)}" />
<input type="hidden" name="_eventId" value="createbank" />
```

The input "\_eventId" field "value" attribute matches with a web-flow transition "on" attribute. The input "\_flowExecutionKey" utilises the information needed for a web-flow engine.

The web-flow xml file can use defined services without additional configuration. That is because service classes has an annotation with service name.

In this example "userService" saveUser() function was used. It was accessible because UsersService class has an annotation:

```
@Service("userService")
public class UsersService {
    public void saveUser() {
        ...
    }
}
```

In the first step to save information into Users table saveUser() function creates a new object from Authorities class and saves its value according on “role” argument passed from the request. Then “User” properties are added and saved:

```
1. public void saveUser(User user, String role) {
2.     Authorities authorities = new Authorities();
3.     authorities.setAuthority(role);
4.     authorities.setUsername(user.getUsername());
5.     if (role.equals("COURIER")) {
6.         user.setIsCourier(true);
7.     }
8.     user.setAuthorities(authorities);
9.     user.setEnabled(true);
10.    usersDao.save(user);
11. }
```

The next step is followed by saving Couriers or Customers information by using the same principle.

Most of fields in the registration process has Spring built in server side validators. They are added via annotations on a top of the parameters of the entity class.

For example some of Users class validators are:

```
@Size(min=8, max=60)
@Pattern(regexp="^[\\w]{8,}")
private String username;

@Size(min=8)
@Pattern(regexp="^[\\S+$")
private String password;
```

These two validators checks size and regular expression - username must consist of only letters and numbers, password cannot have spaces. Validation messages are defined into “com/project/messages/messages.properties” file.

To check if password matches with confirmation password, there is a Java Script client side validator. It uses passwordMutch() function. If the both strings match and return value “true”, then form with id “details”, submit function is enabled.

```
$( '#details' ).submit(passwordMutch);
```

---

### 4.5.3 User's profile

After user is successfully registered and logged in to the account he can edit his information by clicking “Edit Account” link in the header.

**Edit profile:**

Username:	<input type="text" value="testclient1"/>
Name:	<input type="text" value="Coben"/>
Surname:	<input type="text" value="Dallas"/>
Email:	<input type="text" value="real@gmail.com"/>
Phone:	<input type="text" value="0234234324"/>
Address 1:	<input type="text" value="224 Carterhatch rd."/>
Address 2:	<input type="text" value="Enfield"/>
Postcode:	<input type="text" value="EN3 5DJ"/>

Figure 4.8. “Edit Account” page.

It doesn't use web-flow. Instead it is a single function executed from LoginController, that uses “UserService” service as a wrapper class to call UsersDAO Hibernate function:

```
session().saveOrUpdate(user);
```

Here user is a “user” bean name that matches with annotation given into entity **User** to “users” table:

```
@Table(name="users")
public class User implements Serializable {
...
}
```

## 4.6 Job booking console

After successful registration, the Customer is able to display a booking page. That is the main part of a web application. At the top of the page Google map with icons representing available couriers is situated. Next to the map a table with courier usernames and link to additional courier related information (phone number, email, feedbacks etc.) is displayed.

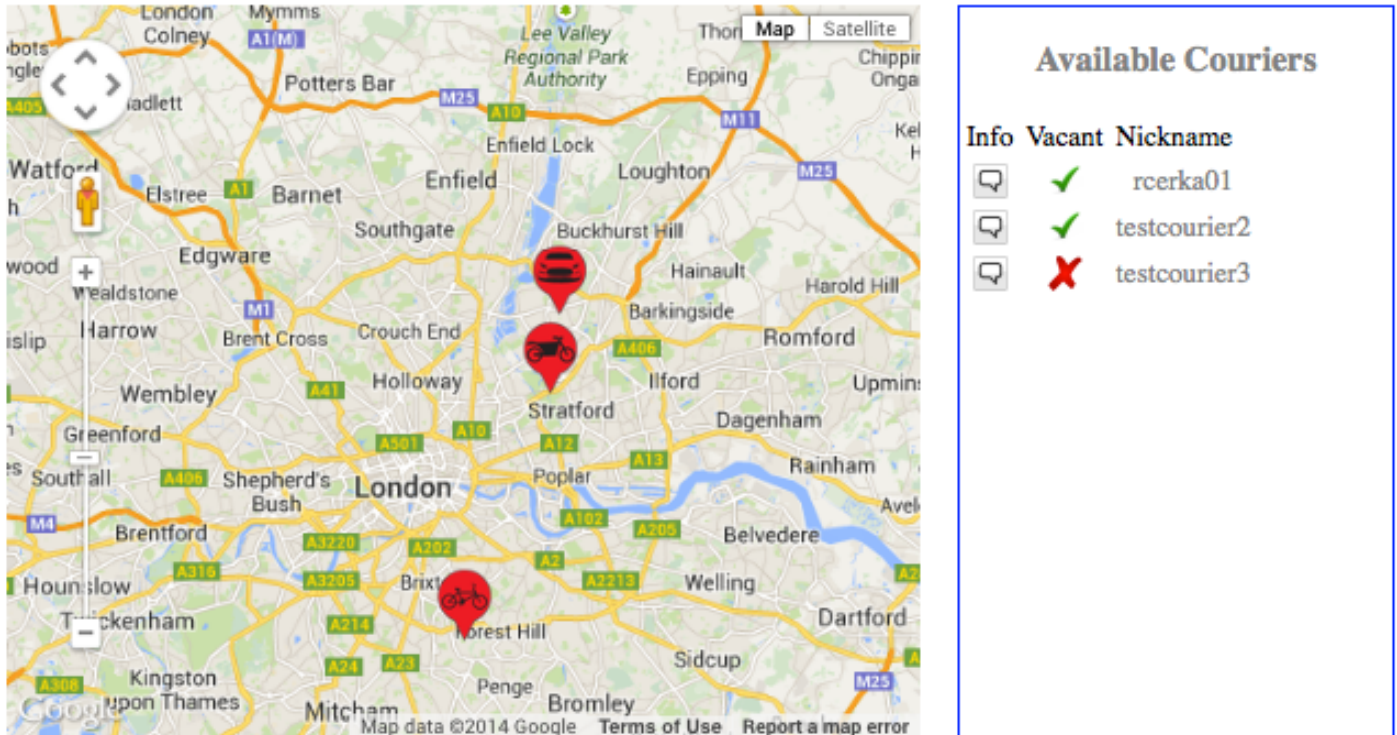


Figure 4.9. The Google map and a courier's table.

Down below there is main area in a red square for current and previously booked jobs. New bookings can be made by selecting collection, delivery addresses and the courier from drop-down boxes. Confirmation of booking is done by pressing "To Book" button.

### My Booked Jobs

	Collection Address	Delivery Address	Courier	Job Status	Created
<a href="#">view</a> <a href="#">remove</a> <a href="#">feedback</a>	Euro Parts Totenenham	my work address	testcourier2	Sent to Courier	2014-08-17 19:07:09.0
<a href="#">view</a> <a href="#">remove</a> <a href="#">feedback</a>	Walthamstows library	Birkbeck	testcourier1	Delivered	2014-08-10 16:02:19.0
<a href="#">view</a> <a href="#">remove</a> <a href="#">feedback</a>	njam njam	Birkbeck	testcourier1	Approved	2014-08-13 16:34:50.0

### New Job

Collection Addresses:  

Walthamstows library

Delivery Addresses:  

my work address

Available Couriers:  

rcerka01

To Book

Further below there are two tables with entered collection and delivery addresses, together with links to delete or update them (Figure 4.10).

### My Collection Addresses

	Name	Postcode	Address	
<a href="#">edit</a> <a href="#">remove</a>	Walthamstows library	E17 3EH	91 Hoe street	London
<a href="#">edit</a> <a href="#">remove</a>	Euro Parts Tottenham	EN9 3EF	12 Albion drive	Edmonton
<a href="#">edit</a> <a href="#">remove</a>	njam njam	E12 W14	vietinja	ko nezin neviens

[Create Collection Address](#)

### My Delivery Addresses

	Name	Postcode	Address	
<a href="#">edit</a> <a href="#">remove</a>	my work address	E4 3DD		
<a href="#">edit</a> <a href="#">remove</a>	Birkbeck	EC1 1AH		

[Create Destination Address](#)

Figure 4.10. Tables with collection and delivery addresses.

#### 4.6.1 The Google Map

The Google map represents courier type (via icon). Courier can be a bike, a push bike, a car or a van driver. It also represents its geolocation and information window with couriers nickname (it can be extended to display more information, like phone number, email, etc.).

The information for google map is gathered from `getcouriersjson()` function in couriers controller. It returns JSON data which includes all working couriers, number of working couriers and their related jobs.

```
{
  "couriers": [
    {
      "username": "rckerka01",
      "type": "bike",
      "isworking": true,
      "isavailable": true,
      "lat": 51.55018,
      "lon": -0.017344
    },
    {
      "username": "testcourier1",
      "type": "car",
      "isworking": true,
      "isavailable": true,
      "lat": 51.495064,
      "lon": -0.082397
    },
    {
      "username": "testcourier2",
      "type": "car",
      "isworking": true,
      "isavailable": false,
      "lat": 51.507656,
      "lon": -0.024496
    },
    {
      "username": "testcourier3",
      "type": "push bike",
      "isworking": true,
      "isavailable": false,
      "lat": 51.435066,
      "lon": -0.082397
    }
  ],
  "number": 7,
  "jobs": [
    {
      "id": 5,
      "created": 1406476157000,
      "status": "Sent to Courier",
      "notes": "",
      "distance": 0.0,
      "price": 0.0,
      "visible": true,
      "coltimefrom": null,
      "coltimetill": null,
      "deltimfrom": null,
      "deltimetill": null,
      "coldate": null,
      "deldate": null,
      "clientUsername": "testclient1",
      "courierUsername": "testcourier1"
    },
    {
      "id": 4,
      "created": 1406476111000,
      "status": "Sent to Courier",
      "notes": "",
      "distance": 0.0,
      "price": 0.0,
      "visible": true,
      "coltimefrom": null,
      "coltimetill": null,
      "deltimfrom": null,
      "deltimetill": null,
      "coldate": null,
      "deldate": null,
      "clientUsername": "testclient1",
      "courierUsername": "testcourier1"
    }
  ]
}
```

Data collection function `getcouriersjson()` is called asynchronously from JavaScript function `updatePage()`.

```
function updatePage(){
    $.getJSON("<c:url value='/getcouriersjson'/>", updateCouriers);
}
```

Then `updatePage()` function, after page is loaded, is called periodically every 5 seconds. That according on tests becomes the shortest most stable interval.

```
function onload() {
    updatePage();
    window.setInterval(updatePage, 5000);
}
```

The actual data update on Google map is achieved through the `updateCourier()` function. In the beginning it wipes off all previous markers and then loops through all couriers in the JSON data array and, in the case, if a courier status is “working”, position a marker on the map according on it’s latitude and longitude coordinates. It also represents a vehicle type that courier is using. Information window can contain more data, but that will be decided after initial evaluation.

```
1. function updateCouriers(data){
2.
3.     deleteMarkers(data);
4.
5.     or (var i = 0; i < data.couriers.length; i++) {
6.         if (data.couriers[i].isworking) {
7.             latLng = new google.maps.LatLng(data.couriers[i].lat,
data.couriers[i].lon);
8.             marker[data.couriers[i].username] = new
google.maps.Marker({
9.                 position: latLng,
10.                map: map,
11.                title: data.couriers[i].username,
12.                icon: icons[data.couriers[i].type]
13.            });
14.
15.            markerName = marker[data.couriers[i].username];
16.            info = "Hi, my name is <strong> " +
data.couriers[i].username + "</strong>. <br> Find my feedbacks and
contact information <br>under relative icon in the list.";
17.
18.            addInfoWindow(markerName, info, map);
19.
20.        }
21.    }
22. }
```

This code is written by the sample taken from Google Map JavaScript v.3 API website [15].

---

#### 4.6.2 Couriers list

The couriers list has the same function as Google maps - it represents couriers current status and links to additional information, like phone number, email and feedbacks. If the courier starts to work, it is triggered through the boolean value "isworking" from couriers table in the database, and couriers information appears in the couriers list. If a courier is working but is on delivery, then his status in the courier's list field "vacant" changes to red cross. Couriers status in the database is updated through the android device and is described in chapter 4.5. Additional information about a courier is accessible through the "info" link next to his nickname in the couriers list (Figure 4.11).



Figure 4.11. Courier information page.

The couriers list is generated into HTML table with id attribute "courierstable".

```
<table id="courierstable">
<tr><th>Info</th><th>Vacant</th><th>Nickname</th></tr>
</table>
```

The table content has been updated asynchronously using the same function which uses Google Maps - updateCouriers(). Therefore couriers list is also updated every five seconds. It uses JQuery function to remove all previous table rows

```
$("#courierstable").find("tr:gt(0)").remove();
```

Then data for new values are gathered into the same loop which uses Google Map markers

```
1. if (data.couriers[i].isavailable) {
2.   available = "<img width='20' height='20' src='$
{pageContext.request.contextPath}/static/img/true.jpg'>"; }
3. else {
4.   available = "<img width='20' height='20' src='$
{pageContext.request.contextPath}/static/img/false.png'>"; }
5.
6. feedback = "<a style='font-size:11px;' href='<c:url value='/
courierdetails?uid=" + data.couriers[i].username + "'/>'><img width='20'
height='20' src='${pageContext.request.contextPath}/static/img/
Feedback2.jpg'></a>";
```



And finally by using JQuery function, new rows of data are added to the table.

```
$("#courierstable").append( "<tr><td>" + feedback + "</td><td>" +  
available + "</td><td>" + data.couriers[i].username + "</td></tr>" );
```

---

#### 4.6.3 Collection and delivery addresses

The list of entered collection and delivery addresses is located at the bottom of the page. The buttons “Create Collection Address” and “Delete Location Address” links to related form. Next to each address are links to “edit” and “remove” address.

Edit address page:

Name: Walthamstows library  
Address: 91 Hoe street  
Address: London  
Postcode: E17 3EH  
Notes:   
Save

If a customer removes address, it becomes invisible rather than removed from the database. Reason for this is, if a user wants to remove address which is related to existing job, it will bring SQL error of deleting data being related with existing record. Next possibility, if a user removes both - the job and the address but a courier still have this job in his accounts, it will return the same error. Therefore “remove” option is making record invisible rather than deleting it. This is achieved via “isvisible” boolean database field.

When address lists are populated from the database the following Hibernate criteria is taking place:

```
1. @SuppressWarnings("unchecked")  
2. public List<Sendfrom> getSendfrom(String username) {  
3.     Criteria crit = session().createCriteria(Sendfrom.class);  
4.     crit.add(Restrictions.eq("username", username));  
5.     crit.add(Restrictions.eq("visible", true));  
6.     return crit.list();  
7. }
```

In this example are gathered data from “sendfrom” table with an authenticated customer username and a field boolean having value “true”.

To make an address invisible the following expression in a “Sendfrom” or “Sendto” controllers are updating “isvisible” field:

```
1. @RequestMapping("/removeaddressdeliver")  
2. public String removeAddressDeliver(@RequestParam("id") Integer id) {  
3.     Sendto sendto = addressesService.getSendto(id);  
4.     sendto.setVisible(false);  
5.     addressesService.createOrUpdateAddressDeliver(sendto);  
6.     return "redirect:availability";  
7. }
```

Hibernate function is used to update record, wrapped into AddressService class.



---

#### 4.6.4 Jobs list and create new booking

The area in the page with a red border represents booked jobs and offer to book a new one. To accomplish a booking at least one collection and one delivery address must be made. If it isn't, the booking process is protected by a validator:

**To create job you must have at least one "collection" and at least one "delivery" address.**

It is controlled via JavaScript function blockJobCreation() which disables booking form's submit button that has an id "tobook".

```
function blockJobCreation() {  
    $('#validjob').show();  
    $('#tobook').attr("disabled", true);  
}
```

Condition has been checked with java jstl "if" tag which is verifying returned parameter from the controller. It must contain at least one collection and one delivery address.

```
<c:if test="${empty sendfrom}"><script type="text/  
javascript">blockJobCreation();</script></c:if>
```

There is no need for "enable" function because there is no option to add addresses asynchronously. After entering a new address, a page will be reloaded.

To book a new job is a two-step process. At first part there must be selected collection and delivery addresses as well as as a courier from the drop-down boxes. The next step is to add an additional information for the job in the step two (Figure 4.12). This page also represents a chosen courier on the map, his contact details and collection, delivery addresses.

Here initially are added only two fields - "price offer" and "notes". After evaluation and consultation with testers more items if necessary will be included.

After booking is done every job has additional links to "view" a job, leave "feedback" and "remove" a job from the list. View a job link leads to the same page where step two for a booking takes place. Here customer can trace a courier location on the map and update additional information. In this way a simple information exchange with a courier is maintained.

Collection: Walthamstows library, 91 Hoe street London, E17 3EH  
 Delivery: my work address, 34 Beresfield , E4 3DD  
 Courier: Ray Standers, tel: 07562793514, email: ray@ray.com  
 Price offer:

Notes:

[Save and Send](#)

Ray Standers location

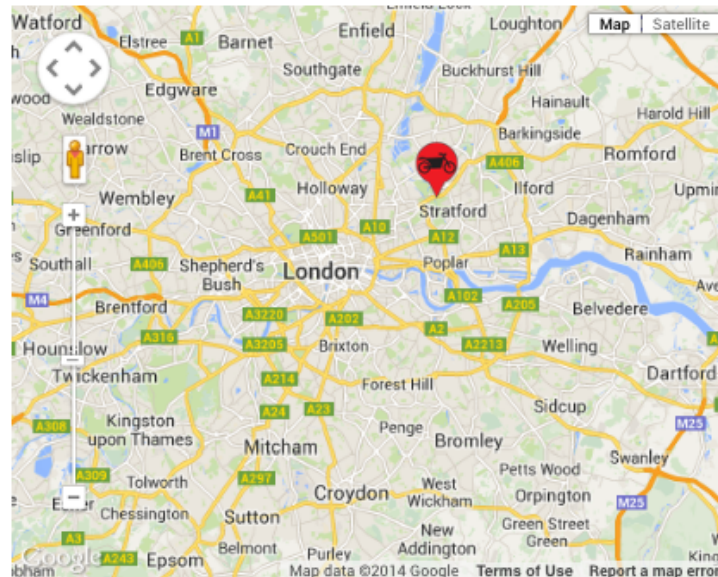


Figure 4.12. “Step two” job booking page.

Link, to leave a feedback, is connected with a listed job. However, all feedbacks are going to be related with a courier who has committed that job. To remove job (similarly to remove address), instead of deleting the record, makes it invisible.

The job status is updated asynchronously with the the same updateCouriers() JavaScript function that is used for the map markers and couriers list. In HTML table every job status cell have an equal “id” to the jobs “id” in the database.

```
<td id="{jobs.id}" style="font-weight: bold;"><c:out value="{jobs.status}"></c:out></td>
```

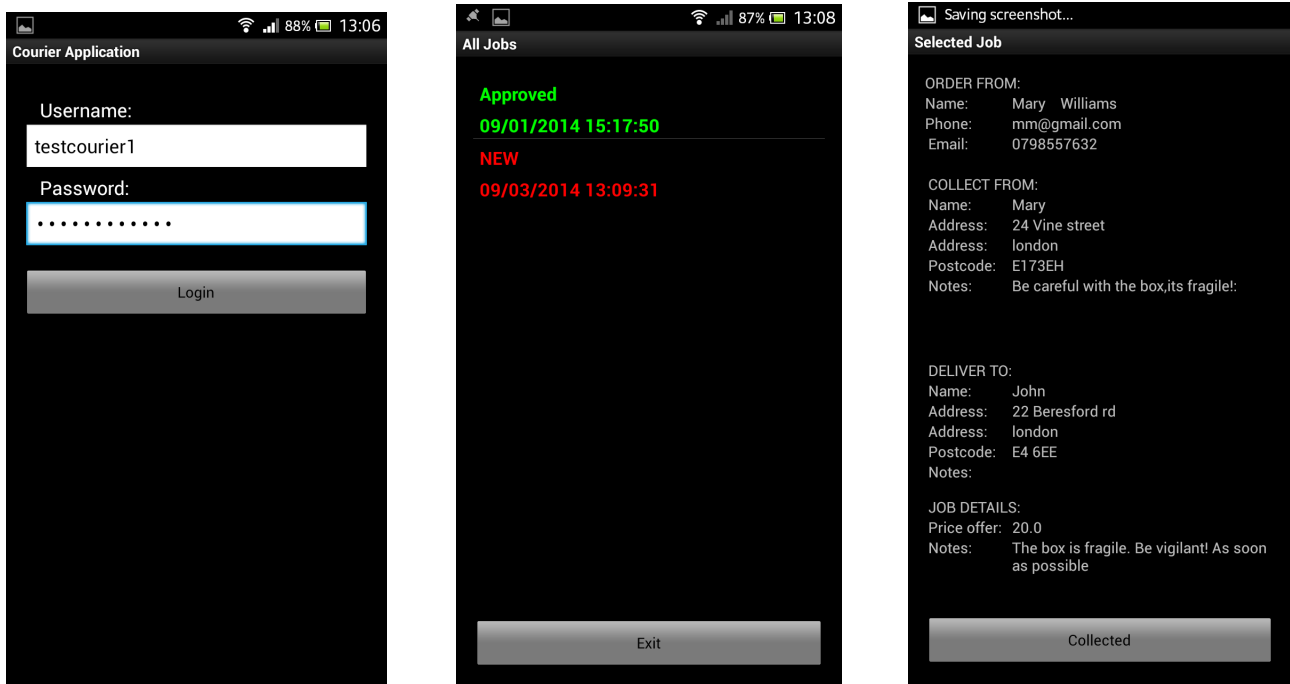
The job status is updated every five seconds and for better visibility different job statuses have different colours - “Sent to courier” is grey, “Approved” is green, “Rejected” is red, “Collected” is yellow and “Delivered” is brown.

```
for (var i = 0; i < data.jobs.length; i++) {
  $("# + data.jobs[i].id).html(data.jobs[i].status);
  if (data.jobs[i].status == "Approved")
  {setGreen(data.jobs[i].id);}
  ...
}
```

The colours are changed by adding and removing css classes where each has only one value which is a text colour. This is delegated to JavaScript functions setGreen(), setYellow(), setRed() and setBrown().

## 4.7 Android application and REST services.

The Android application is relatively simple and compact, however, it serves all predefined tasks. It consists of three main activities and four classes. The first activity serves the courier identification; the second one holds the list of active jobs. If the job is completed it disappears from the list. Jobs are in different colours which represent their statuses, red - for new job, green for approved job, yellow - after collection is done. Every list item (job) is clickable, and it opens the third activity where all job details are presented. At the bottom of the third activity is a button which changes the job status. It can be either “approved” or “rejected”. Then it changes to “collected” and ends with “delivered”.



The main challenge in android application is connection with database and to perform an asynchronous periodic updates. This is achieved with REST services (described in chapter 2.2.3).

### 4.7.1 REST services

All functions that respond on android application HTTP request are located in the web application RestController class. For a simplicity it is always considered as a GET request.

Values to update or add data to database are passed through HTTP as GET parameters. This controller contains five functions for different purposes. They are **loginJson()** and **stopcourierjson()** to login and logout from application, **updateStatusJson()** that responds on button pressed from ShowJobActivity to change the job status. This function uses “id” parameter to locate job by its id, and the “status” parameter, containing value to update the selected job’s status. The function **getDetailsJson()** gathers information for ShowJobActivity. It returns JSON data with job details, matched with job id passed as a parameter. The function **getjobsJson()** returns JSON array with every job related to a couriers username.

There is a major security leak by passing plain (not encoded) password in a **loginJson()** function as a GET parameter. This is not going to be done in the scale of this project, but steps to resolve this issue would be to use POST rather than GET method, encode the password before sending it, and / or use SHTTP over HTTP for communication.

All functions that serve android HTTP requests return JSON data. It can be array of required information or, like in the case with **loginJson()** function, acknowledgement with “success” value of 1 or 0.

To get a function to return JSON data, *produces*=“*application/json*” value into annotation “Request Mapping” must be added. Annotation “Respond Body” also must be added. Parameters are passed in the same manner as in the rest of all application, by “Request Parameter” annotation. The returned data is a HashMap with values of an object.

Example:

```
1. @RequestMapping(value="/getdetailsjson", produces="application/json")
2. @ResponseBody
3. public Map<String, Object> getDetailsJson(@RequestParam("id") Integer
id){
4.     ...
5.     Map<String, Object> data = new HashMap<String, Object>();
6.     data.put("job", job);
7.     ...
8.     data.put("success", 1);
9.     return data;
10. }
```

---

#### 4.7.2 Android application asynchronous tasks.

To call any HTTP request in android application has been used AsyncTask class. This class allows to perform background operations without having to manipulate threads and/or handlers. AsyncTasks are used for short operations (a few seconds at the most.). An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the main thread. An asynchronous task is defined by separate class and has a 4 steps, called *onPreExecute*, *doInBackground*, *onProgressUpdate* and *onPostExecute* [18].

Definition of class is extending AsyncTask class.

```
class Login extends AsyncTask<String, String, String> {
```

The first step in android application is activating progress bar window.

```
1. protected void onPreExecute() {
2.     super.onPreExecute();
3.     pDialog = new ProgressDialog(MainScreenActivity.this);
4.     pDialog.setMessage("Loggin. Please wait...");
5.     pDialog.setIndeterminate(false);
6.     pDialog.setCancelable(true);
7.     pDialog.show();
8. }
```

The second step is to run an actual thread which (if necessary) collects request parameters. To perform a request and format its parameters according on its type (GET or POST), *JSONParser* class is used.

```

1. protected String doInBackground(String... params) {
2.     runOnUiThread(new Runnable() {
3.         public void run() {
4.             try {
5.                 List<NameValuePair> params = new ArrayList<NameValuePair>();
6.                 params.add(new BasicNameValuePair("uid", usernameString));
7.                 ...
8.                 JSONObject json = jsonParser.makeHttpRequest(url_login,
"GET", params); ...

```

Then according on return, action is committed. For example, it could show a message if JSON return from REST service is 0, or moving to next activity if return is 1.

```

1. if (success == 0) {Toast.makeText(MainScreenActivity.this,
"Wrong credentials. Try again.", Toast.LENGTH_LONG).show();}
2. if (success == 1) {Intent intent = new
Intent(MainScreenActivity.this, AllJobsActivity.class);
3.             intent.putExtra("username", usernameString);
4.             startActivity(intent);
5.             finish();
6.     }
7. ...

```

The third step onProgressUpdate has never been used.  
And the forth step terminates progressDialog window.

```

protected void onPostExecute(String file_url) {
    progressDialog.dismiss();
}

```

This structure of AsyncTask is used to perform all five possible HTTP requests types.

---

#### 4.7.3 Geolocation

The activity, where couriers receive new jobs is always open and is working as the main console of the android application. It is AllJobsActivity. It implements LocationListener interface. Further **getLocation()** method is responsible for updating location coordinates.

Getting user location in Android works by means of callback. Code must require to receive location updates from the LocationManager ("Location Manager") by calling **requestLocationUpdates()**, passing it a LocationListener. LocationListener must implement several callback methods that the Location Manager calls when the user location changes or when the status of the service changes [19].

The location is updated by GPS and Network location providers. Both are updating "last known location" every five seconds. It is recommended by a Android developer site not to update location more frequent than one minute, because it is costly procedure for resources and battery usage. However, to test the program stability on critical circumstances before initial evaluation this frequency is left very high. Five seconds is also frequency for JavaScript webpage asynchronous update.

```

1. locationManager
=(LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
2. locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
5000, 10, this);
3. locationManager.requestLocationUpdates(LocationManager.
NETWORK_PROVIDER, 5000, 10, this);
4. Location locationGPS =
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
5.      Location locationNet =
locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
6.
7.      if (null != locationGPS) { GPSLocationTime =
locationGPS.getTime(); }
8.      if (null != locationNet) { NetLocationTime =
locationNet.getTime(); }
9.
10.     if ( 0 < GPSLocationTime - NetLocationTime ) {location =
locationGPS; } else {location = locationNet; }
11.
12.     if(location!=null) {
13.         lon = (float)location.getLongitude();
14.         lat = (float)location.getLatitude(); }

```

This strategy of gathering location is not meant to be the optimal for performance, but most thorough to perform initial evaluation and test stability.

---

#### 4.7.4 MainScreen Activity

The MainScreenActivity class is an entry point into the application. It collects courier's username and password via login form and by committing HTTP request calls web application service "loginJson". That, according on returned "success" value 0 or 1, closes MainScreen activity and opens AllJobsActivity.

Web service is called as an Asynchronous Task which is described in a chapter 4.5.2. The task name is Login, and it is executed by a command:

```
new Login().execute();
```

It passes entered username and password as GET request parameters and according on returned JSON data - in the case of 0 it generates message of incorrect credentials, in case of 1, it opens AllJobs activity and closes itself, therefore preventing for user to press a back button.

---

#### 4.7.5 AllJobs Activity

The AllJobsActivity class acts as the main console for android application. It populates the list with jobs related to registered courier (excluding the finished ones). The list is populated via asynchronous call (described in a chapter 4.5.2) for a "LoadAllJobs" service by passing username as a "uid" parameter. It also updates courier coordinates by passing "lat" and "lon" parameters, gathered from **getLocation()** function (described in chapter 4.5.3). Every list item is linked with a ShowJob activity which explores the details of selected job.

To update the list (populate with new jobs), and periodically update the location of courier, the LoadAllJobs task is called every 30 seconds. It is achieved by running "updateTimerThread".

```

1. private Runnable updateTimerThread = new Runnable()
2. { public void run()
3.     { getLocation();
4.       jobsList = new ArrayList<HashMap<String, String>>();
5.       new LoadAllJobs().execute();
6.       customHandler.postDelayed(this, 30000);
7.     }
8. };

```

This runnable is executed from **onCreate()** function.

```
customHandler.postDelayed(updateTimerThread, 0);
```

The “LoadAllJobs” is also called from the **onRestart()** function.

```

1. protected void onRestart() {
2.     super.onRestart();
3.     jobsList = new ArrayList<HashMap<String, String>>();
4.     new LoadAllJobs().execute();
5. }

```

Therefore list will be updated also after returning from ShowJob activity.

At the bottom of a layout “Exit” button is located. It triggers execution of Exit task. This task passes “uid” parameter to “stopcourierjson” service, which changes in database “isworking” value from 1 to 0, therefore a courier disappears from working couriers list.

To despite the usage of battery, to increase convenience for a courier and avoid problems from application returning from a screen saver, the screen lock is blocked with a following line in **onCreate()** function.

```
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

---

#### 4.7.6 ShowJob Activity

This activity is designed to represent job details and change a job status. Asynchronous task “GetJobDetails” is executed from **onCreate()** function. It passes job “id” value to “getdetailsjson” service, which returns all job related information from “jobs”, “sendto”, “sendfrom” and “users” tables (the task execution principle is described in a chapter 4.5.2). It also sets the correct button to change job status, according on its current status. Background task “UpdateJobStatus” is executed after a button for change status is pressed. This task executes “updatestatusjson” service which changes job “status” field with “status” parameter value passed by a service.

## 5. Testing

In this chapter code testing and the testing environment configuration is explained. It is followed by a user test description and analyse. The errors observed while the user tests are summarised and discussed.

### 5.1 Code testing environment configuration

---

#### 5.1.1 Dividing application into profiles

The additional testing database has been involved to perform the tests. This database is exact replica of existing one, used in application development process. To perform tests the new dependency “org.springframework spring-test,” was added to pom.xml file. It has the same version as a Spring core, which is 4.0.5.

To separate tests from production code, the application is divided into two profiles, “production” and “development”. To achieve that in the dao-context.xml file data source configuration must be surrounded by a “beans” tag with the attribute “profile” (this behaviour is present only from Spring version 3.1).

```
1.    <beans profile="production" >
2.        <jee:jndi-lookup jndi-name="jdbc/ray" id="dataSource"
3.            expected-type="javax.sql.DataSource">
4.        </jee:jndi-lookup>
5.        ...
6.    </beans>
```

To get application to work for this profile, here must be set up system item property or added a context parameter. This application uses a context parameter which is inserted into the web.xml file.

```
1.    <context-param>
2.        <param-name>spring.profiles.actice</param-name>
3.        <param-value>
4.            production
5.        </param-value>
6.    </context-param>
```

---

#### 5.1.2 Set up Spring to use JUnit tests

Instead of using “src” source folder, the tests are separated into “test” source folder. This folder contains “com.project.test” package with two sub packages “com.project.test.config” for configurations and “com.project.test.tests” for actual tests.

To use JUnit tests into Spring, the Junit dependency with the current latest version of 4.11 must be added to the pom.xml file. Instead of using Apache Tomcat for tests a “dbcp” connection is used. To use that commons-dbcp, dependency with the latest version of 1.4, is added to the pom.xml file.

The testing environment configuration consists of two configuration files, datasource.xml and jdbc.properties. The datasource.xml file holds the data source configuration for “development” profile. Instead of using “jndi” connection (like production profile), it uses



"jdbc". The data source configuration is surrounded by "beans" tag with "development" profile attribute. The property placeholder tag contains path to the second configuration file "jdbc.properties" where credentials for test database access is located.

```
<context:property-placeholder
    location="com/project/test/config/jdbc.properties" />
```

The next bean in datasource.xml file is "jdbc" datasource configuration

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.driver}"></property>
    <property name="url" value="${jdbc.url}"></property>
    <property name="password" value="${jdbc.password}"></property>
</bean>
```

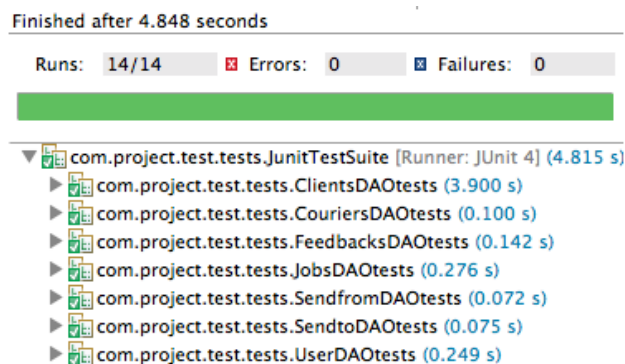
Following configuration is the same as in "production" profile. It has settings for transactional state and Hibernate. The actual tests are located into "com.project.test.tests" package. The tests must hold three annotations: "ActiveProfile" presents a profile name; "ContextConfiguration" holds path to configuration files and "RunWith" holds a JUnit class name.

```
1. @ActiveProfiles("development")
2. @ContextConfiguration(locations = {
3.     "classpath:com/project/config/dao-context.xml",
4.     "classpath:com/project/config/security-context.xml",
5.     "classpath:com/project/test/config/datasource.xml" })
6. @RunWith(SpringJUnit4ClassRunner.class)
7. public class UserDAOTests {
8.     ...
9. }
```

## 5.2 Code tests

The code tests are located inside "com.project.test.tests" package. The tests are simple and include only DAO functions. They should be more thorough and cover controller's behaviour and security, but due a limited time to develop the project amount of tests are minimised.

There are 14 tests that covers at least some aspect of every DAO function.



The Dao tests were very handy to use for casual code refactoring, to keep a track on all variety of data requests during development.

### 5.3 User tests

The software was tested by five persons on the 7th of September. It took place in Walthamstow area in London. Everyone committed one task each. It followed by the completion of feedback form. Every person was in a different age group and from different background and occupation. Three persons were simulating client side of operations, and two became couriers.

The courier activities were performed by one professional courier, who works for the delivery company E-Courier for two years, and an individual who has never been a courier but agreed that he could become one for that day.

The Customer activities were tested by three persons who had used courier services before or agreed they would have an interest do it in the future.

All people were supposed to be in the same area at the same time for testing. Due to previous introduction and discussion of common courier pitfalls and general use of software, the main three points of discussion became - registration process to test how user friendly it is, delivery experience (for couriers) and booking experience (for clients) and the overall conclusion.

The tests had the common format. The two most important feedbacks are from Arvils Kalnberzs, who works as a courier for two years (Figure 5.1), and Linda Madison who uses couriers services daily (Figure 5.2). All reports are included in Appendix B of this report.

**Courier name:** Arvils Kalnberzs

**Age:** 32

**Occupation:** Courier, E-courier Ltd.

**Task description:** *Please write brief description of your experience using Couriers Online System. Please mention are you testing as a client or as a courier.*

#### **Registration process**

Registration into website went smoothly. After registration I was able login into android application and wait for the jobs to appear on my screen.

#### **Delivery process**

I did two deliveries within a web project program, developed by Raitis Cerkasovs. I was using Sony Xperia S Android smartphone.

The android application seems very simple, however it was not very intuitive. The one major disadvantage was lack of sound notifications for incoming job. I had to constantly look in the phone, to be sure that I did not miss any new job assignment.

Another important feature would be to use a map within this phone app as a navigation tool for setting up route and directions to different destinations. It would also be very useful to have a link with a phone number which would allow me to call the customer, or use any alternative built in communication tool, messaging via chat, perhaps. The application crashed once when I got an incoming call.

#### **Overall feedback**

Despite a few inconveniences, the app allowed me to complete the job, to do a delivery and communicate with a customer. The developer must improve a few details for the app, but I can see it becoming useful tool for people working in my profession.

Figure 5.1. Courier test report example

**Clients name:** Linda Madison

**Age:** 36

**Occupation:** Marketing executive in Investment company Eurix IMC.

**Task description:** *Please write brief description of your experience using Couriers Online System. Please mention are you testing as a client or as a courier.*

#### **Registration process**

I tested a client side of the program. I've opened the Couriers Online System and registered myself as a client. I had to write my username and create a password along with other details to complete my registration process first. The system didn't accept my username at first, since it needed to be at least 8 letters long. The system didn't show it on the page where I had to enter it, I think it should be improved.

#### **Booking process**

After completing the registration process I got to the booking page where I had to enter the collection address, delivery address and send the job to my chosen courier from the drop down menu and press a button to book. System brought me to the next page where I had to enter the payment, see delivery details and enter the job description. I wrote: Please pick up a box of Magnum ice-cream from Shelbourne Newsagent. When all information was filled in, system allowed me to send it to courier for acceptance. This page also displayed a map with an icon of chosen courier and sees his/her current location, which I really liked as an extra feature. I would suggest that the delivery address and pick up address also would be shown on a map as a small dots or any other symbol.

#### **Overall feedback**

I could see on the booking page that my package has been collected which I found very useful and also on which stage of delivery process the courier was at the moment. Courier's feedbacks are another good feature.

An overall positive experience - I received my box of dessert on time!

Figure 5.2. Client test report example

## 5.4 User test results

---

### 5.4.1 Registration process

The registration process for a courier and a customer are very similar. However, couriers additionally have to download the Android application. One person who was performing courier activities experienced problems with an app download link. From his report:

*“After registration, I downloaded the android app for testing. I would rather use Google Play to download an app.”*

Unfortunately, there are financial costs involved to deploy any android application into Google Play. It would be a logical step forward if the project were used in a real life environment.

Nearly every tester had inserted an invalid entry into the field that has a validator.

*“The system didn’t accept my username at first since it needed to be at least eight letters long. The system didn’t show it on the page where I had to enter it, I think it should be improved.”*

A fix for that would be the usage of client side validation along (or subsidised) with the server side validation. That would inform the user after entry of each character if the field is valid or not.

---

### 5.4.2 Booking process

During the booking process everyone agreed that feedback is a great thing to have, and it certainly gained a trust to a courier. However, it raises the question, what to do if the courier just started to work and haven’t got feedbacks? Unfortunately, this application cannot give answer to that. Probably couriers have to work in an area where are a lot of potential jobs and he / her will be picked as the only available person, and earning his / her potential first feedbacks.

At the booking process step 2 was confusion around job booking optional fields. It was not self-explanatory that customer has to put price offer just if he wants to.

*“I was confused what payment to give to the courier since I don’t have any experience how much these services cost.”*

The two-step job booking system was too confusing and raised a lot of questions. It has to be redesigned in one-step booking where it is clear what is optional and what is compulsory.

Another good improvement is to make a calculator, which would return a distance from Google map, and then multiple it with a designated coefficient and offer suggested price for delivery.

To help to trace a courier position relatively to his collection or delivery points was recommended to put two markers in the map, and probably a suggested route from the Google

*“I would suggest that the delivery address and pick up address also would be shown on a map as a small dots or any other symbol.”*

Suggestion was also made to involve an option to transfer address entered on user registration used as a collection or delivery address. This can be common collection or delivery place.

*“In the next step, I had to provide addresses. My collection address was the same as my home one. It will be more convenient if there is a drop down menu with existing information to choose.”*

---

#### 5.4.3 Delivery process

As a main missing feature was a lack of acknowledgement for an incoming booking.

*“The one major disadvantage was lack of sound notifications for incoming job.”*

It is relatively easy to fix. There has to be added a ringer if a new job is coming. Also job object must have ringer parameter that on creating job would be true and after first update false. Otherwise, the phone will be ringing on every asynchronous data request, until a courier will change job status to “approved” or “rejected”.

Both couriers were not happy for lack of maps or navigation included into app.

*“Another important feature would be to use a map within this phone app as a navigation tool for setting up route and directions to different destinations.”*

Also, customer phone number must be done as a link to make a call option straight from the app. Messaging system must be included.

*“It would also be very useful to have a link with a phone number which would allow me to call the customer, or use any alternative built in communication tool, messaging via chat, perhaps.”*

This certainly must be included if the application is going to be used in the production. In the scope of this project these features were not enough time to integrate.

One of couriers experienced crash of the application during an incoming call to the phone when he was using the app.

*“The application crashed once when I got an incoming call.”*

---

#### 5.4.4 Overall experience

Overall experience for persons simulating couriers or clients was more or less equal. Everyone agreed that all system must be improved. There are a lot of areas with lack of intuitive usage design and convenience. However, if details would be improved, all system would be usable and functional.

## 5.5 Errors observed during testing

Along with the testing there were two major problems monitored with the android application.

The first, in areas with a poor network cover the application was crashing while not being able to do an asynchronous data update.

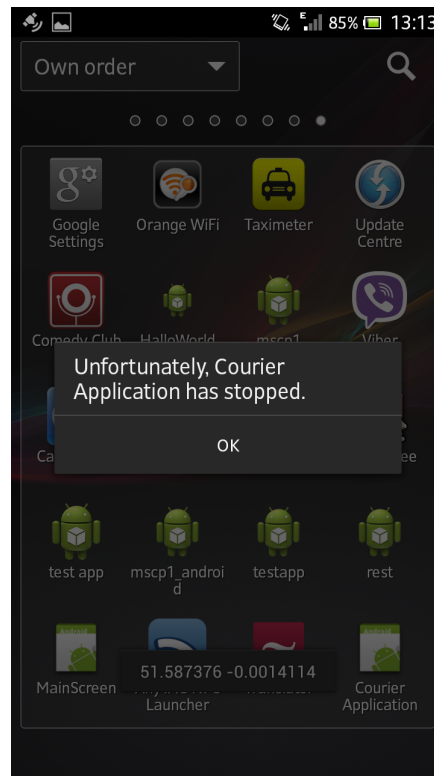


Figure 5.1. The Android error message.

This behaviour was expected, because due a lack of time to finish the application, the control over data access quality was eliminated.

The second issue was related with positioning. Android development has a wide area of choice of the best strategy for location sensing. The project was designed for couriers who are most of the time outside. Therefore the “GPS” location was used over “Network” location sensing. However, tests proved that was not the best practice.

For the courier performing a job in an open area with no obstacles, to do the delivery, everything was fine, and his location was detected correctly (Figure 5.2).



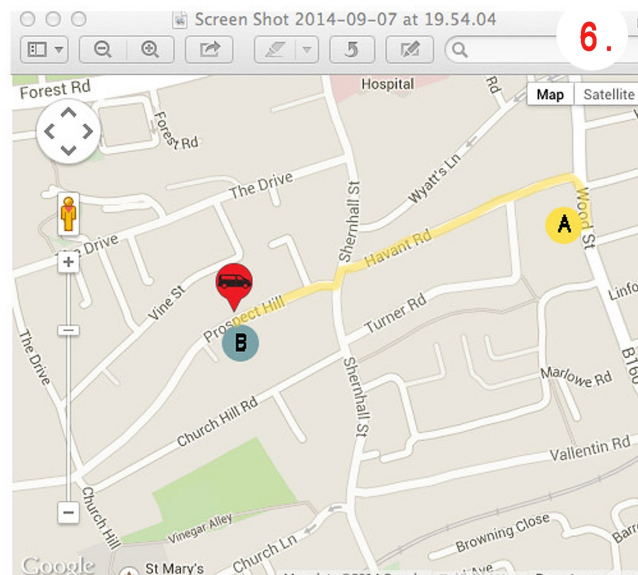
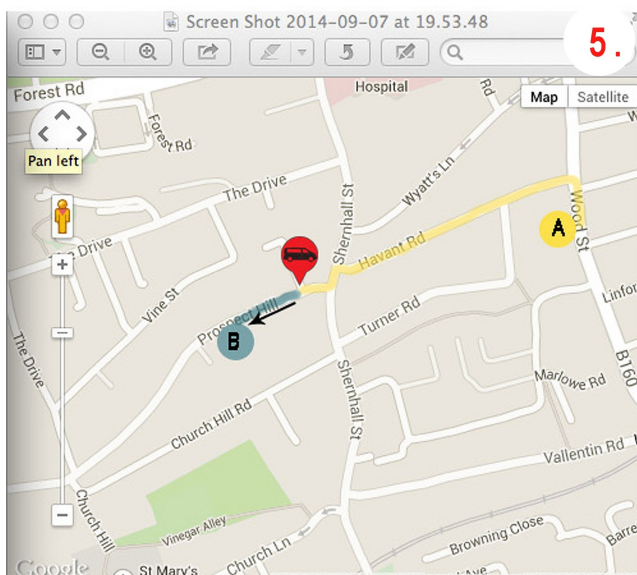
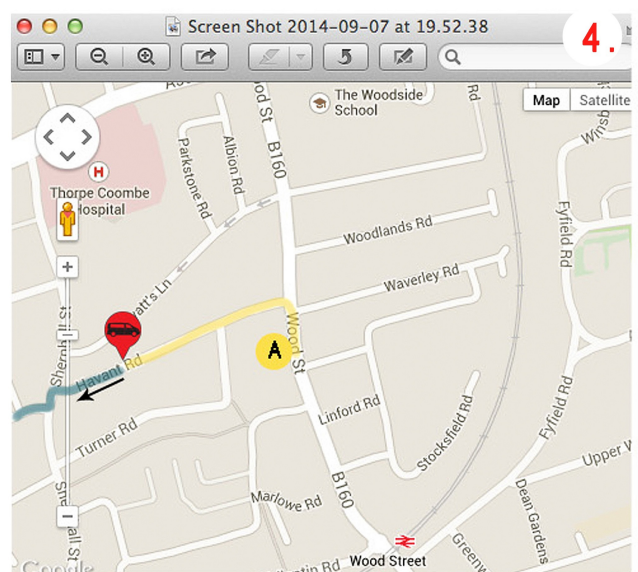
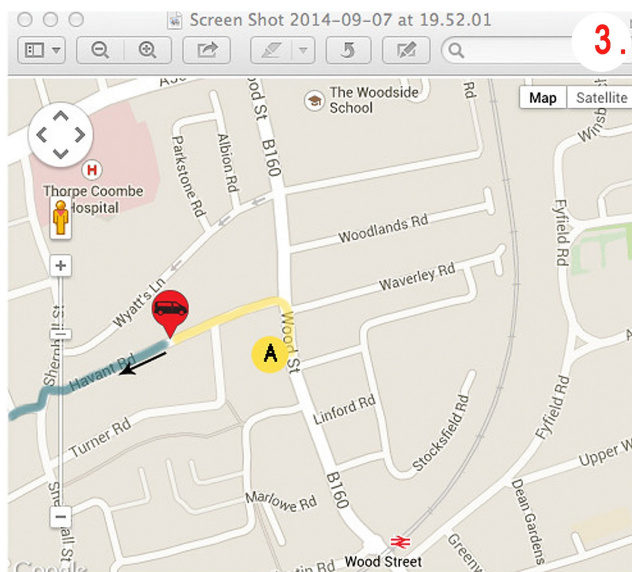
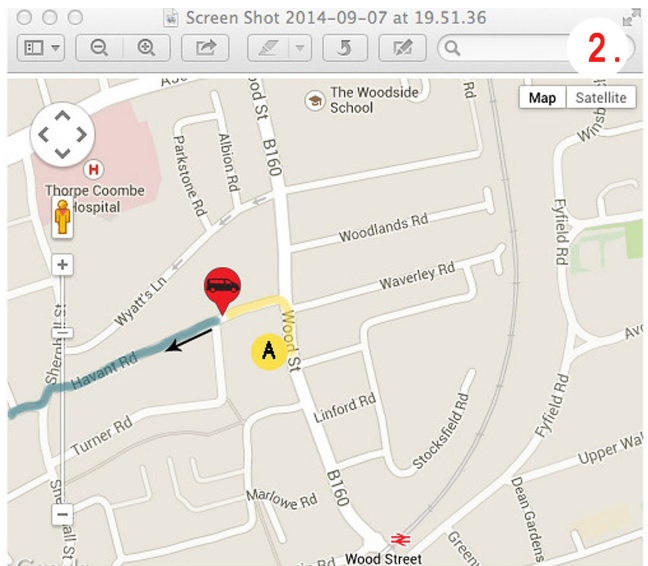
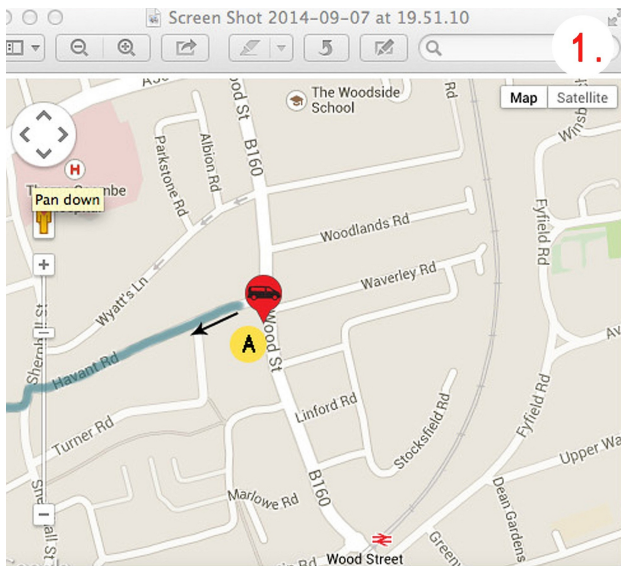


Figure 5.2. Maps with courier position detections where results are satisfied.



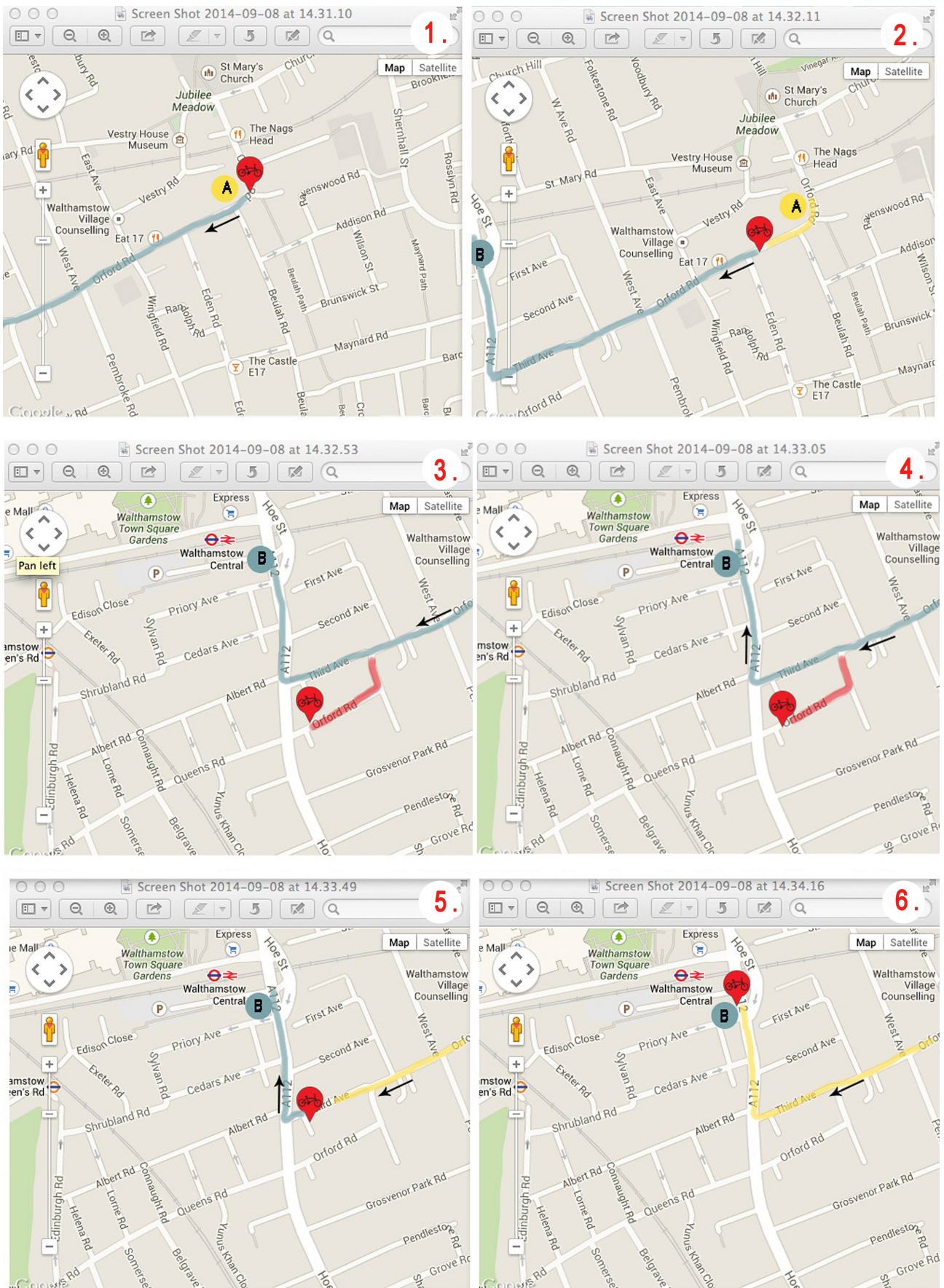


Figure 5.3. The map with a courier position where errors (marked with a red line) were detected.

When a task to deliver an envelope to the bank was performed, which is situated within an area with tall buildings, the location detection was not that precise anymore (Figure 5.3).

The location detection is crucial, because a customer who will be tracking couriers don't have much information how accurate a location is sensed, and can decide that a courier is doing something wrong.

The current accuracy of location detection by one of testers was described as "erratic."

*"His movement was a bit erratic. But in general I could follow up, and all the updates (collected or delivered) I could receive on the booking page."*

During the testing, when a courier started his job, we discovered a small issue. The courier was added to the map and a courier's list but not appeared in the drop-down box to choose a courier. It showed up only after reloading the page. Drop-down box is not updating asynchronously.

Stress testing to check the program performance beyond the limits of normal performance were not executed. Reason is a lack of time for developing the project. However, the software uses AWS service that has automatic tools to handle an increasing capacity. Stress tests would test AWS performance that is reliable in any case.

## 6. Evaluation and Conclusion

In this chapter is discussed if project met its objectives, weak and strong points of application, and is concluded with the list of possible improvements in the future.

### 6.1 Overall results

The project objective was to develop an online based courier booking system which would allow for couriers to carry jobs independently from courier companies.

It was supposed to include following tasks:

1) *organise courier and customer accounts*

The couriers have an option to log-in to their accounts and check all previous and current jobs. The data formation must be improved in the future, probably some report generation technology (like windward [22]) can be involved.

The client side has a wide variety of features how to organise the job list.

2) *using android device as a mobile equipment where smartphone gathers data of couriers location and the job statuses.*

The android application collects geolocation from the GPS or Network providers, passing it through the REST service to the main application, which stores it for later use. The android device is used as a tool to trigger job status that can move from being sent to a courier, being rejected or approved, further to collected and delivered.

3) *feedback system must be evolved to create history for courier activities.*

Every customer has an option to leave feedback for any booked job. Feedbacks are presented altogether with a list of courier's information.

4) *the scale of the project must be adjusted according on demand where top level is presumed for 150 thousand couriers carrying about 30 jobs a day.*

This is an absolute maximum of transactions potentially being populated. The application is deployed to the AWS cloud service. It has a mechanism of automatic resources adjustment according on data volumes. It can automatically increase or decrease amount of running instances therefore adjust used resources against the scale.

### 6.2 Project weaknesses and strengths

The main project weakness is its development stage. The application only satisfies basic needs to carry out the work. All its components are developed to a level where they function together without any additional features, which would involve stability, ergonomic or the comfort for the user. To bring the application to the production stage the improvements listed in the chapter 6.3.1 and 6.3.2 must be made.

The weak point of application is location sensing strategy, that doesn't function absolutely right, and the issues with the android connection loss, that must be resolved.

The positive decision was to deploy application to the Amazon Web Services. In the future it can save a lot of financial costs if the project is carried out to production level.

The SpringMVC seems a very stable platform and has a lot of pre-developed and tested tools. Huge amount of documentation which allows easy and fast development even by starting from very basic level of knowledge. The choice of Spring MVC as a development framework is a strong point.

The testing team in general terms agreed with introduced functionality to resolve problems of self-employed couriers. They liked idea of feedbacks, and enjoyed to follow courier location changes on the map. Everyone agreed that this software could be useful.

### 6.3 Project future improvement

As the project covers different areas of development (mobile, location sensing, communication, web, database etc.) and brings together different technologies, the areas of its improvement are endless. The main need to make this application alive for production level should be a developing of iPhone version of mobile application. Due to a growing popularity of using internet from mobile devices, the mobile version of booking application must be developed as well.

The good feature which would invoke unused technologies, would be saving courier location updates into NOSql database. The courier data and a courier location is forming a perfect example of paired data used in NOSql tables. It could be retrieved and analysed with MapReduce algorithms and gathered valuable information of location statistics (for example, best possible routes according on time of the day, etc.).

To improve performance of existing technologies, to lift application to a level to step into production, the following list of minimum improvements should be done.

---

#### 6.3.1 Improvements for web application

The web application has satisfied the minimum requirements of being able to handle courier operations at its basic level. The following list includes only minimum improvements necessary for application to be used in production.

- 1) The overall design and CSS must be improved. By following suggestions from testers, the job booking structure must be optimised.
- 2) Payment method for customers to pay a courier for the job, must be introduce.
- 3) Registration process must be improved. User's email can be delegated as a username, and conformation of account through email, could be developed.
- 4) Autocompletion for address fields (address detection from postcode and house number).
- 5) Phone number confirmation through "sms".
- 6) Persons registered as couriers could also be customers. This case must be considered.
- 7) Administrator profile must be developed, to maintain the website (possible task could be to edit feedbacks).
- 8) Client side validation (as suggested from the testers).
- 9) Must be involved validator for a unique username (currently it brings sql error).
- 10) Function to reset password through the email.
- 11) For each job delivery addresses can be more than one.
- 12) Feature to display courier's distance.
- 13) Feature for the home address being automatically transferred as a collection or delivery address.
- 14) There must be collected "collection" and "delivery" timestamps for each job.

- 15) Courier feedbacks should introduce five star valuation for the job.
- 16) Clients must be forced to leave one feedback. It can also be a standard three word sentence from the drop-box, like “Excellent job, thanks” or “Reliable person, good work”.
- 17) Optimised list view for feedbacks and couriers in the table if they don't fit in one page.
- 18) Sorting must be introduced for the job list.
- 19) Courier icons in the map could hold more information, like phone number, email, etc
- 20) Messaging must be developed to establish better communication between courier and customer (and probably between couriers).

---

### 6.3.2 Android application

Android application is developed till beginning stage. The following list represents the minimum improvements needed for it, to be used in production.

- 1) REST functions have a huge security hole, where password is passed as a HTTP GET parameter in a plain format. Password should be encoded and SHTTP used outer HTTP.
- 2) Ringer must be included if a new job arrives. Also, if job notes are updated without acknowledgement.
- 3) Collection and delivery timestamps must be collected.
- 4) Connectivity problems and error handling must be resolved.
- 5) Some strategy must be introduced how to extend the battery life.
- 6) Location detection must be improved.
- 7) Design of application interface must be improved.

---

### 6.3.3 Unimplemented behaviour

Due to limited time for developing the project, it is not implemented method for proof of delivery. However, the environment for implementation is prepared and all needed is one additional activity, where user could be transferred from job's “delivered” status change. This activity should execute photo taking, uploading via HTTP and registering its file name to related job. It wouldn't bring much additional and interesting aspects of programming.

Other implementation of proof of delivery was a signature scanning. That also would result with an image file upload process. Signature can be taken by using any open source software. As an example could be a code snippet from [mysamplecode.com](http://mysamplecode.com) website “android capture signature using canvas” [26].

## 6.4. Conclusion

This report presents the development of web based system to book a self-employed courier for a delivery job. The application has two parts - web application and mobile application. Both parts mainly satisfied raised objectives. The fact of collection of proof of delivery is excluded due a deadline for completion of the project.

The application is developed till Alfa stage. It has only necessary features to carry on a courier job. Initial testing of a small group of people has been done. However, this project has the wide range of technologies - android programming, including location sensing and asynchronous background requests. Spring MVC framework for web development, JSON data servers, REST service to establish a connection between android and web application. JavaScript to process asynchronous requests, MySql relation database and Amazon WebServices.

The main work was done on SpringMVC framework which was new technology for me, for Android development I had a basic pre-knowledge. This project was useful in two ways - to get till a working stage the application that can be usable for someone in the future, and secondly - in relatively short period of time learned all mentioned technologies, and by using them - developed an application that works.

## 7. References

- [1] E-courier Ltd. company website.  
<https://www.ecourier.co.uk>
- [2] Google Map, Wiki  
[http://en.wikipedia.org/wiki/Google\\_Maps](http://en.wikipedia.org/wiki/Google_Maps)
- [3] Amazone Web Services  
<http://aws.amazon.com/ec2/>
- [4] RESTful services, Wiki  
[http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer)
- [5] JQuery, Wiki  
<http://en.wikipedia.org/wiki/JQuery>
- [6] Data Access Object pattern description from Oracle website  
<http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>
- [7] Hibernate official site  
<http://hibernate.org>
- [8] Apache Tiles homepage  
<http://tiles.apache.org/index.html>
- [9] Class Diagrams, Wiki  
[http://en.wikipedia.org/wiki/Class\\_diagram](http://en.wikipedia.org/wiki/Class_diagram)
- [10] Eclipse Web Tool Plugin  
<http://www.eclipse.org/webtools/>
- [11] Apache Maven homepage  
<http://maven.apache.org>
- [12] Apache Tiles homepage  
<https://tiles.apache.org/faq.html>
- [13] The Spring Web-Flow, Wiki  
[http://en.wikipedia.org/wiki/Spring\\_Web\\_Flow](http://en.wikipedia.org/wiki/Spring_Web_Flow)
- [14] Simple Spring WebFlow App, by David Winterfield, 2008  
<http://www.springbyexample.org/examples/simple-spring-web-flow-webapp.html>
- [15] Hibernate homepage  
<http://hibernate.org>
- [16] JSON wiki  
<http://en.wikipedia.org/wiki/JSON>
- [17] Google Map JavaScript API v.3 website  
<https://developers.google.com/maps/documentation/javascript/examples/marker-simple>
- [18] Android developer guide, AsyncTask  
<http://developer.android.com/reference/android/os/AsyncTask.html>

[19] Android Geolocation Strategies, by Android Developer website  
<http://developer.android.com/guide/topics/location/strategies.html>

[20] Stress testing, Wiki  
[http://en.wikipedia.org/wiki/Stress\\_testing\\_\(software\)](http://en.wikipedia.org/wiki/Stress_testing_(software))

[21] Windward, Java report generation technology  
<http://windward.net>

[22] DHL company website  
<http://www.dhl.co.uk/en.html>

[23] UPS company website  
<http://www.ups.com/content/gb/en/index.jsx>

[24] Courier Systems company website  
<http://www.couriersys.co.uk>

[25] The Java Spring tutorial, from John Purcell  
<https://www.udemy.com/javaspring/#/>

[26] Android capture signature using canvas, from mysamplecode.com  
<http://www.mysamplecode.com/2011/11/android-capture-signature-using-canvas.html>



## 8. Appendix A - Test results

### 8.1 User tests

---

#### 8.1.1 Client test report 1

07.09.2014.

**Clients name:** Jan Marcin

**Age:** 28

**Occupation:** Barmen, All Star Lines

**Task description:** *Please write brief description of your experience using Couriers Online System.  
Please mention are you testing as a client or as a courier.*

#### Registration process

I was asked to test the system as a client. At the registration process I had to write down my username and all the other relevant information. Then I pressed to get to step two, but it returned me back to the same page. The requirement to have eight numbers or letters for username should be mentioned at the beginning of the registration otherwise it is a waste of time to go back and correct it.

#### Booking process

Booking page looks fine with all the courier icons listed on the map. I could find their feedbacks right next to the available couriers. This information is extremely important to be able to trust a courier. I entered my collection and delivery addresses. Then the page asked me to choose a courier but didn't show available ones in the drop down menu of the booking page. After refreshing the page, I was able to book it and the program took me to the next page. There was a map and an empty box to enter my payment details and I could write down package description. I was confused what payment to give to the courier since I don't have any experience how much these services cost. In the field for job notes I wrote: Please, deliver an envelope to the Barclays Bank and send it for approval. I got rejected. Now I had to look for a new courier. It would help a lot if there would be a column showing next available couriers in my area. If the program knows the collection address, it could automatically suggest me couriers around this area. Would save time and money. When I got approved by second courier, I followed his steps in the map.

#### Overall feedback

Two-page system to overlook the process of delivery is not comfortable. It would be more useful if the current state of the package ( accepted, collected, delivered) and a couriers movement on the map were on one page. Otherwise, it is a useful program.

---

### 8.1.2 Client test report 2

07.09.2014.

**Clients name:** Linda Madison

**Age:** 36

**Occupation:** Marketing executive in Investment company Eurix IMC.

**Task description:** *Please write brief description of your experience using Couriers Online System. Please mention are you testing as a client or as a courier.*

#### **Registration process**

I tested a client side of the program. I've opened the Couriers Online System and registered myself as a client. I had to write my username and create a password along with other details to complete my registration process first. The system didn't accept my username at first, since it needed to be at least 8 letters long. The system didn't show it on the page where I had to enter it, I think it should be improved.

#### **Booking process**

After completing the registration process I got to the booking page where I had to enter the collection address, delivery address and send the job to my chosen courier from the drop down menu and press a button to book. System brought me to the next page where I had to enter the payment, see delivery details and enter the job description. I wrote: Please pick up a box of Magnum ice-cream from Shelbourne Newsagent. When all information was filled in, system allowed me to send it to courier for acceptance. This page also displayed a map with an icon of chosen courier and sees his/her current location, which I really liked as an extra feature. I would suggest that the delivery address and pick up address also would be shown on a map as a small dots or any other symbol.

#### **Overall feedback**

I could see on the booking page that my package has been collected which I found very useful and also on which stage of delivery process the courier was at the moment. Courier's feedbacks are another good feature.

An overall positive experience - I received my box of dessert on time!

---

### 8.1.3 Client test report 3

07.09.2014.

**Clients name:** Ilva Kalnberza

**Age:** 28

**Occupation:** I work as a graphic designer.

**Task description:** *Please write brief description of your experience using Couriers Online System. Please mention are you testing as a client or as a courier.*

#### **Registration process**

I received an invitation to test a Courier Online job flow program as a client. I was told to create my imaginary delivery that could be delivered to me by a courier.

First of all I had to create an account as a client.

The registration went smoothly except a minor issue with the postcode. I've entered it incorrectly, and the program proceeded to the next step. Had to go back and correct it. The address insertion should be automatic and if the program would also offer a list of existing addresses to choose from, it would make life more simple.

#### **Booking process**

Then I was taken to the booking page. The map with available couriers is in full view that helps when you want to book the nearest available courier to your location. In this way, the distance between addresses would be shorter, and the delivery fee would be cheaper. I like it!

In the next step, I had to provide with addresses. My collection address was the same as my home one. It will be more convenient if there is a drop down menu with existing information to choose.

When this stage was done, I had to choose an available courier from the drop down menu. I needed a folder to be delivered from my home address to work. It's a small package. It could be delivered by a courier on the bike. Very useful feature that all icons show what kind of courier (bike, car, van) is operating in the area. In this way, I can decide on the most appropriate vehicle for delivery and save some money. When I had chosen a courier, I checked the feedback that gave me a better assurance that my folder would be in safe hands.

Next step, I booked a chosen courier and was taken to a different page. There was a map with just one visible courier and fee offer for this job I had to write myself. I wasn't sure how much would it cost. A chart with advisable price tag for one mile or some relevant information would be very useful. Otherwise, I got stuck. How much shall I offer? After that, I entered some details about delivery. And send it for the courier to approve. Then I could follow him on the map. His movement was a bit erratic. But in general I could follow up, and all the updates (collected or delivered) I could receive on the booking page. That's useful.

#### **Overall feedback**

If few of the features changed to get it more user-friendly, I would use this service in the future.

---

### 8.1.4 Courier test report 1

07.09.2014.

**Courier name:** Arvils Kalnberzs

**Age:** 32

**Occupation:** Courier, E-courier Ltd.

**Task description:** *Please write brief description of your experience using Couriers Online System. Please mention are you testing as a client or as a courier.*

#### **Registration process**

Registration into website went smoothly. After registration I was able login into android application and wait for the jobs to appear on my screen.

#### **Delivery process**

I did two deliveries within a web project program, developed by Raitis Cerkasovs. I was using Sony Xperia S Android smartphone.

The android application seems very simple, however it was not very intuitive. The one major disadvantage was lack of sound notifications for incoming job. I had to constantly look in the phone, to be sure that I did not miss any new job assignment.

Another important feature would be to use a map within this phone app as a navigation tool for setting up route and directions to different destinations. It would also be very useful to have a link with a phone number which would allow me to call the customer, or use any alternative built in communication tool, messaging via chat, perhaps. The application crashed once when I got an incoming call.

#### **Overall feedback**

Despite a few inconveniences, the app allowed me to complete the job, to do a delivery and communicate with a customer. The developer must improve a few details for the app, but I can see it becoming useful tool for people working in my profession.

---

### 8.1.5 Courier test report 2

07.09.2014.

**Courier name:** Hans Snieder

**Age:** 44

**Occupation:** Catering manager, La Siesta

**Task description:** *Please write brief description of your experience using Couriers Online System. Please mention are you testing as a client or as a courier.*

#### **Registration process**

My name is Hans Snieder, and I will be a program tester as a courier. I don't have any experience as a courier, but I give it a go.

To test an android application, first I had to register as a courier on PC. I picked a push bike as my initial vehicle for deliveries. After registration, I downloaded the android app for testing. I would rather use Google Play to download an app.

#### **Delivery process**

The testing could start! I got outside with my android phone and a push bike. Logged in the application and waited for a job to receive. I had to check my phone at all times, because it didn't make any noise when the job offer was sent to me. Some sound or bleep is missing to make it more user-friendly. Consequently, I accepted a job to deliver a box of ice-cream from newsagent to client's address. But I didn't know where to go, the application is missing google maps to see my location and also a route to get to the destination. I had to open google maps that were already installed on my mobile and search for the address myself. When I was sure where to go, I started my journey.

The signal wasn't strong on my mobile; I got logged off twice from the application. It wasn't easy to log in and move at the same time. I didn't want to stop to be time efficient.

I reached my destination and went to buy a client a box of ice-cream, but the newsagent had many flavours, and I wanted to call a client to make sure to get the right one. The application was also missing a call button to get in touch with the client. It was a bit of a downside as I wanted to be fast. Everything else went well.

#### **Overall feedback**

If all mentioned issues improve, I think this application would be useful not only for couriers, but also for clients.

## 9. Appendix B - Setting up a workspace

Many different technologies must be preconfigured to start to develop the web based application. Especially a lot of configuration work must be done for Spring MVC framework (Tiles, controllers, web-flows, JSON server etc.). All of them are gathered from various tutorials (mainly “The Java Spring tutorial” from John Purcell [25]), and Spring documentation. In this appendix, are explained in detail the most important key aspects of configuration for Spring MVC components.

### 9.1 Configuration for web application

To develop a Web application the following main technologies are used - Spring MVC framework (discussed in project proposal chapter 3.2), Apache Tomcat7 as a web server, MySQL as a relational database (project proposal 3.4), Eclipse version Luma with WTP plugin [10], as a working environment.

---

#### 9.1.1 General environment settings

To begin with the web application development the configurations of framework must take place. As a first step to integrate Spring MVC framework into Eclipse environment the necessary dependencies are added with Maven, which is a software project management and comprehension tool [11]. Maven is included with a standard package of Eclipse.

To work with a Spring MVC framework It is necessary to include following libraries:

*spring-core, spring-context, spring-beans, spring-web, spring-web, spring-mvc.*

To establish a connection with MySQL database libraries must be included:

*spring-jdbc and MySQL connecto.*

The dependencies are added through “pom.xml” file. The latest version of libraries are used. The common style to add dependencies is via three rows of xml tuple. It includes dependencies group, id and version.

```
1. <dependency>
2.     <groupId>org.springframework</groupId>
3.     <artifactId>spring-core</artifactId>
4.     <version>4.0.5.RELEASE</version>
5. </dependency>
```

Spring MVC is routing all requests through one dispatch servlet. In the project, this servlet is a “dispatch-servlet.xml” file. To push all requests to use this servlet, the configuration must be made in “web.xml” file. Settings must include servlet name and origin class.

```
1. <servlet>
2.     <description></description>
3.     <display-name>dispatch</display-name>
4.     <servlet-name>dispatch</servlet-name>
5.     <servlet-class>org.springframework.web.servlet.DispatcherServlet</
servlet-class>
6.     <load-on-startup>1</load-on-startup>
7. </servlet>
```

To include CSS files, images, Java Script files etc., the file location must be defined in a dispatch servlet. In the project this location is within a “resources” folder and definition is:

```
<mvc:resources location="/resources/" mapping="/static/**" />
```

---

### 9.1.2 Controller configuration

The controllers in the project are located in a separate package “com.project.controllers”. The controller is a simple Java class that has to have an annotation “@Controller”.

Example:

```
@Controller
public class CourierController {
    ..
}
```

To get Spring MVC to look for the controller in the controllers package the following closure must be added into a dispatch servlet:

```
<context:component-scan base-package="com.project.controllers"></
context:component-scan
```

To read via annotation, it must be followed by:

```
<context:annotation-config></context:annotation-config>
```

If the function in the controller are delegated to to the routed request, then function must be annotated with “RequestMapping” annotation and have to have a routed path as an argument.

Example:

```
@RequestMapping("/courierdetails")
    public String courierDetails() {
        ...
    }
```

Parameter can be passed to controllers in many different ways. Annotations are used for that purpose, in the project.

Example:

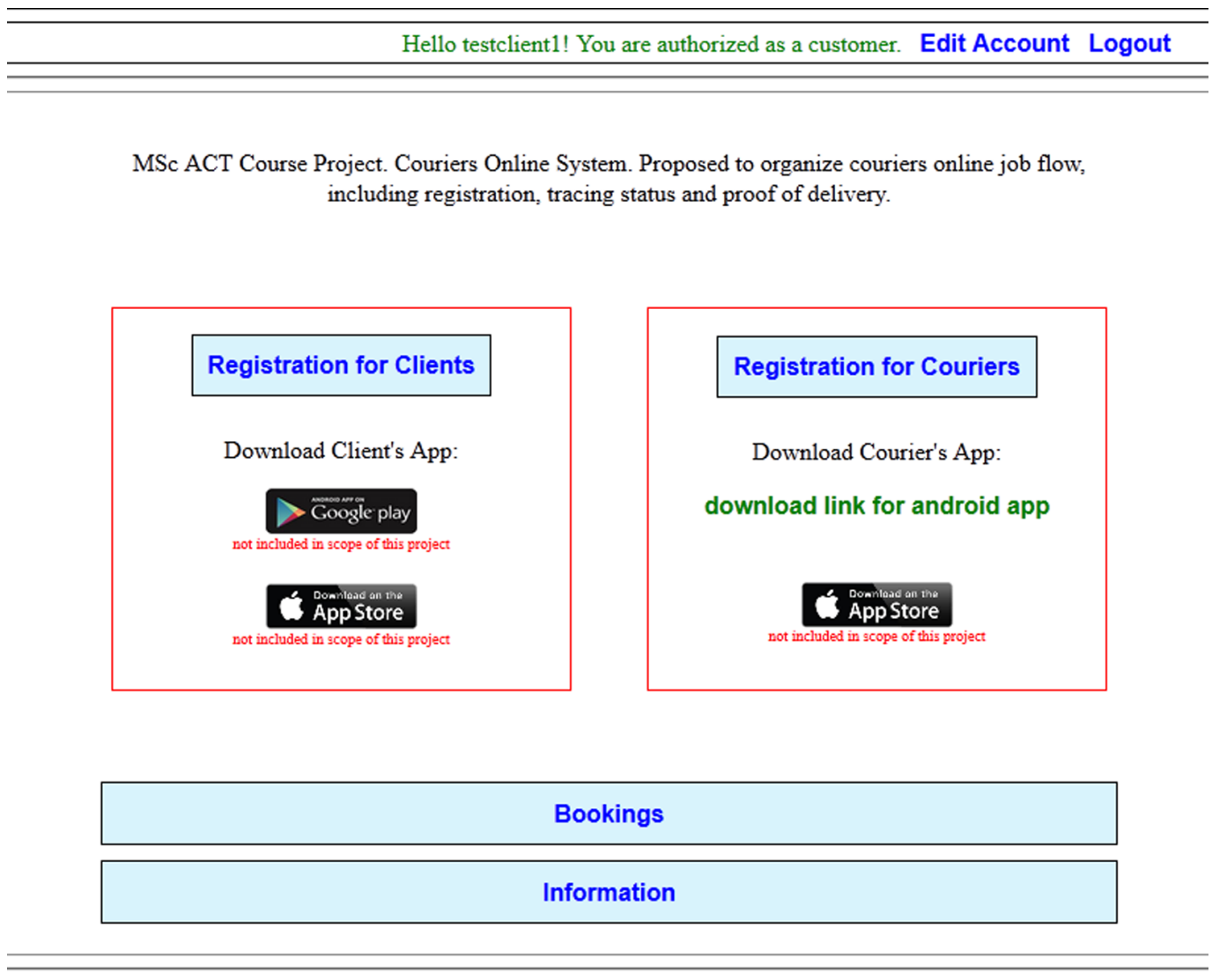
```
@RequestMapping("/courierdetails")
public String courierDetails(Model model, @RequestParam("uid") String
username) {
    ...
}
```

Controller’s function must return string value that represents the name of designated view.

---

### 9.1.3 View and Tiles configuration

“Tiles” is a view layer framework for Java EE applications that allows to separate pages into reusable pieces, according to the Composite View design pattern [12]. In the project, the view layer is divided in three independent pieces. Header, that has been changed according to user’s authentication status. Footer, that remains consistent. Content, that has been updated via all other user activities and stored in designated “jsp” files. The template is stored in “WEB-INF/templates/default.jsp” file. The “tiles” are located into “WEB-INF/tiles/” directory.



© 2013/14

---

Figure 9.1. Screenshot from the start page of application, holds header, footer and content parts.



In order to use Apache Tiles, the following dependencies must be included into “pom.xml” file:

*tiles-jsp, tiles-servlet, tiles-extras*

According on configuration suggestions from “Tiles” website the following closures must be added in the dispatch-servlet:

```
1. <bean id="tilesWievResolver"
class="org.springframework.web.servlet.view.tiles2.TilesViewResolver">
2. </bean>
3.
4. <bean id="tilesConfigurer"
class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
5. <property name="definitions">
6. <list>
7. <value>/WEB-INF/layouts/default.xml</value>
8. </list>
9. </property>
10. </bean>
```

The relation between a string, returned from Controller function, and a View is defined in “WEB-INF/layouts/default.xml” file. Initial setting of default.xml is:

```
1. <definition name="home.main" template="/WEB-INF/templates/default.jsp">
2. <put-attribute name="title" value="Couriers Jobs Distribution System"></put-
attribute>
3. <put-attribute name="header" value="/WEB-INF/tiles/home/header.jsp"></put-
attribute>
4. <put-attribute name="content" value="/WEB-INF/tiles/home/main.jsp"></put-
attribute>
5. <put-attribute name="footer" value="/WEB-INF/tiles/home/footer.jsp"></put-
attribute>
6. </definition>
```

The “value” attribute represents physical location of View content, that is a a separate “jsp” file. Additional definitions change the content of location holding attribute “name”, relating it to definition attribute with the same name, which is returned from a Controller.

Example:

```
1. <definition name="information" extends="home.main">
2. <put-attribute name="title" value="Couriers Online System"></put-attribute>
3. <put-attribute name="content" value="/WEB-INF/tiles/home/information.jsp"></put-
attribute>
4. </definition>
```

Here location of “title” has a value of string and “content” has a value from included “jsp” file.

In the project, “jsp” files are located under “tiles” folder and are separated in following subfolders: “booking”, “home”, “login”, “utility”.

---

#### 9.1.4 Security presets

The security in a Java EE application can be implemented in many various ways. This project is using a Spring security filters. Filters sit in-between the request and a dispatch servlet and according on preset roles, allow or reject the access to resources. To add a Spring security filter to the project following dependencies must be implemented:

*spring-security-core, spring-security-config, spring-security-taglibs*

A Spring Security Filter Chain class has to be declared in the “web.xml” file.

```
1. <filter>
2.     <display-name>springSecurityFilterChain</display-name>
3.     <filter-name>springSecurityFilterChain</filter-name>
4.     <filter-class>
5.         org.springframework.web.filter.DelegatingFilterProxy
6.     </filter-class>
7. </filter>
```

A filter mapping is needed to resolve which url's is going through security. In this project, all requests are secured with a pattern “/\*”

```
1. <filter-mapping>
2.     <filter-name>springSecurityFilterChain</filter-name>
3.     <url-pattern>/*</url-pattern>
4. </filter-mapping>
```

The rules of security are defined in the configuration package (com.project.config) “security-context.xml” file. To get framework to read that file inside a “web.xml”, a context load listener must be declared:

```
1. <listener>
2.     <listener-class>
3.         org.springframework.web.context.ContextLoaderListener
4.     </listener-class>
5. </listener>
```

The path to “security-context.xml” file is set as a context parameter:

```
1. <context-param>
2.     <param-name>contextConfigLocation</param-name>
3.     <param-value>
4.         classpath:com/project/config/security-context.xml
5.     </param-value>
6. </context-param>
```

Ideology of security access in this project is: *everything is forbidden if it is not allowed*. All roles in the file has been read from up to down. Therefore, role must be added at the bottom:

```
<security:intercept-url pattern="/" access="deniedAll" />
```

Any other role has two attributes, pattern - request url and access - an access role.

```
<security:intercept-url pattern="/updatejob"
access="hasRole('ROLE_CUSTOMER')"/>
```

---

### 9.1.5 MySQL connection, Hibernate, DAO and services

In the project, connection to database configuration is separated in “dao-context.xml” file. It increases readability of configuration files. To get the framework to read this file, it’s class path must be added to “web.xml” file Context Listener as a parameter value:

```
<param-value>
    classpath:com/project/config/dao-context.xml
</param-value>
```

To get it working via annotations the following line must be included in “dao-context.xml” file:

```
<context:annotation-config></context:annotation-config>
```

To establish connection with MySQL database, the data source must be defined:

```
<jee:jndi-lookup jndi-name="jdbc/ray" id="dataSource"
    expected-type="javax.sql.DataSource">
</jee:jndi-lookup>
```

And related resource with the same name in Apache Tomcat “context.xml” must be included with relevant connection credentials and settings:

```
<Resource name="jdbc/ray"
    auth="Container"
    type="javax.sql.DataSource"
    maxActive="100"
    maxIdle="30" maxWait="10000"
    username="..."
    password="..."
/>
```

To involve Hibernate [15] in the project, the following dependencies must be added:

*hibernate, hibernate-validator, hibernate-core*

This project uses Hibernate version 3.5.

In the “dao-context.xml” file the Annotation Session Factory Bean and relevant Hibernate dialect for MySQL, must be defined (this configuration works only with Hibernate version 3.x that is used). It is followed by validation group definitions.

```

1. <bean id="sessionFactory"
   class="org.springframework.orm.hibernate3.annotation.AnnotationSession
   FactoryBean">
2.     <property name="dataSource" ref="dataSource"></property>
3.     <property name="hibernateProperties">
4.         <props>
5.             <prop key="hibernate.dialect">
6.                 org.hibernate.dialect.MySQL5Dialect
7.             </prop>
8.             <prop key="javax.persistence.validation.group.pre-persist">
9.                 com.project.dao.PersistenceValidationGroup
10.            </prop>
11.            <prop key="javax.persistence.validation.group.pre-update">
12.                com.project.dao.PersistenceValidationGroup
13.            </prop>
14.            <prop key="javax.persistence.validation.group.pre-remove">
15.                com.project.dao.PersistenceValidationGroup
16.            </prop>
12.        </props>
13.    </property>
17.
18.    <property name="packagesToScan">
19.        <list>
20.            <value>com.project.dao</value>
21.        </list>
22.    </property>
23. </bean>

```

The Hibernate is not working until it is declared as transactional. Following configuration must be added:

```

1. <bean id="transactionManager"
   class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
2.     <property name="dataSource" ref="dataSource">
3.         </property>
4. </bean>
5. <tx:annotation-driven />

```

In the project, the DAO pattern (described in chapter 3.7) for data access has been used. According on a design, every Contextual Class is related to one table. It represent one row in the table. The Contextual Class name is singular form of name of the related table, and holds all its fields as parameters. Getter and Setter functions adds values to parameters.

The Class tableName + DAO deals with the data requests. To get it to work with Hibernate, it must use session factory (declared in the configuration file) and through the session update database with Hibernate functions (Code example 10.1).

```
1.  @Autowired
2.  private SessionFactory sessionFactory;
3.
4.  public Session session() {
5.      return sessionFactory.getCurrentSession();
6.  }
7.
8.  @Transactional
9.  public void saveOrUpdate(Client client) {
10.      session().saveOrUpdate(client);
11.
12.  }
```

#### Code example 9.1. Use of session factory

To completely separate data access, the middle (wrapper) class is used as a service. The DAO classes have a separate package “com.project.dao”. All data access functions are divided between three services - users, bookings and addresses. The service classes are located within a package “com.project.services” that is declared in the “service-context.xml” file, which further is added, in the same way as other configuration files, to “web.xml” Context Loader Listener.

---

### 9.1.6 Web-flow configuration and purpose

The Spring Web captures navigational rules allowing the Spring Web Flow execution engine to manage a conversation and the associated state. At the same time, a *web flow* is a reusable web application module [13]. The project registration process has separate states adding different information to different tables therefore it is appropriate to use a web-flow.

To use Spring web-flow the *spring-webflow* dependency must be added to “pom.xml” file.

The both flows (Courier and Customer registrations) are located under “flows” folder and registered in the dispatch servlet.

```
1. <webflow-config:flow-registry id="flowRegistry"
    base-path="/WEB-INF/flows"
    flow-builder-services="FlowBuilderServices">
2.
3.   <webflow-config:flow-location id="courier-registration"
        path="courier-reg-flow.xml">
4.   </webflow-config:flow-location>
5.
6.   <webflow-config:flow-location id="customer-registration"
        path="customer-reg-flow.xml">
7.   </webflow-config:flow-location>
8. </webflow-config:flow-registry>
```

According on Spring Web Flow configuration an example from [springbyexample.com](http://springbyexample.com) [14] the following lines must be included in the dispatch servlet to get a flow engine running:

```
1. <webflow-config:flow-executor id="flowExecutor"
    flow-registry="flowRegistry">
2. </webflow-config:flow-executor>
3.
4. <bean id="flowHandlerAdapter"
    class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
5.   <property name="flowExecutor" ref="flowExecutor"></property>
6. </bean>
7.
8. <bean id="flowHandlerMapping"
    class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
9.   <property name="flowRegistry" ref="flowRegistry"></property>
10.  <property name="order" value="0"></property>
11. </bean>
12.
13. <webflow-config:flow-builder-services
    id="FlowBuilderServices"
    view-factory-creator="mvcViewFactoryCreator"
    validator="validator" />
```

---

### 9.1.7 JSON data request

JSON is an open standard format that uses human-readable text to transmit data objects that consists of attribute–value pairs. It is primarily used to transmit data between a server and the web application [16]. In the project, it is used to establish data connection with Andoid device (via RESTful service as a response to HTML request) and to update courier information asynchronously with Java Script.

To start using JSON the following dependencies must be added to “pom.xml” file.

*jackson-core-asl, ackson-core-asl*

To get the function return JASON format data, it must have an additional parameter “produces=“application/json” within a Request Mapping annotation. It also must has a Response body annotation to return data in an appropriate format.

Example:

```
1. @RequestMapping(value="/stopcourierjson", produces="application/json")
2. @ResponseBody
3. public Map<String, Object> stopcourierjson(@RequestParam("uid") String
username) {
4.         Map<String, Object> data = new HashMap<String, Object>();
5.         ...
6.         return data;
7. }
```

JSON data returning example:

```
jobs":[{"id":5,"created":1406476157000,"status":"Sent to Courier","notes":"","distance":0.0,"price":
0.0,"visible":true,"coltimefrom":null,"coltimetill":null,"delttimefrom":null,"deltimetill":null,"cold
ate":null,"deldate":null,"clientUsername":"testclient1","courierUsername":"testcourier1"},

{"id":4,"created":1406476111000,"status":"Sent to Courier","notes":"","distance":0.0,"price":
0.0,"visible":true,"coltimefrom":null,"coltimetill":null,"delttimefrom":null,"deltimetill":null,"cold
ate":null,"deldate":null,"clientUsername":"testclient1","courierUsername":"testcourier1"}]}
```

## 10. Appendix C - Instructions to run the code

The web application code is uploaded to Amazon Web Service and is accessible via HTTP:

<http://ec2-54-77-11-0.eu-west-1.compute.amazonaws.com:8080/CourierSysAWSv8/>

The copy of “CourierSysAWSv8.war” file, along with the code, is copied into the root folder of “web-app” directory in the CD.

The android application can be downloaded from the front page of the website by following “download link for android app”. The copy of “CourierAndroidApp.apk” file is copied into the root of the “android-app” folder in the CD.

Alternatively, “war” or “apk” files can be generated from the supplemented code.

The CD included in the project report contains the following folders:

- 1) **web-app** - with the web application code.
- 2) **android-app** - with the android application code.
- 3) **mysql** - contain two “sql” files:
  - a) “db.sql” - data base table export
  - b) “data.sql” - test data export
- 4) **report** - contains PROJ\_CerkasovsR.pdf file.

To run an application, on other server than preconfigured AWS, the following steps have to be made:

- 1) “CourierSysAWSv8.war” must be deployed to the web server. It is tested with Tomcat 7.
- 2) By using db.sql file the MySql database must be generated.
- 3) Test data can be inserted from data.sql file.
- 4) To establish connection with MySql database, code from Figure 10.1 must be added to Tomcat context.xml file (url, username and password must be relative to the settings).
- 5) To get REST services connected with a new web server, the following variables in the Android code must be changed - url\_login, url\_job\_detials, url\_update\_courier\_status, url\_all\_jobs, url\_stop\_courier. Variables have to hold the “uri” string values related to domain name used by a new web server.

```
<Resource name="jdbc/ray"
    auth="Container"
    type="javax.sql.DataSource"
    maxActive="100"
    maxIdle="30" maxWait="10000"
    username="root"
    password=""
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/webEE3"/>
```

Figure 10.1. Code for Tomcat context.xml file, to establish connection with MySql.