Design Patterns
and Code Structures using JEE

PER ERIKSSON

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2011
Design Patterns and Code Structures using JEE

PER ERIKSSON

Master’s Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2011
Supervisor at CSC was Serafim Dahl
Examiner was Stefan Arnborg

TRITA-CSC-E 2011:045
ISRN-KTH/CSC/E--11/045--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication
KTH CSC
SE-100 44 Stockholm, Sweden
URL: www.kth.se/csc
Design patterns and code structures using JEE

The goal of this thesis is to answer the following question:

For a middle sized web development project of a few or more persons using JEE, what frameworks are good choices and how should you apply them?

To answer the question this thesis is divided in two parts. The first part is a market study made on nine different companies primarily in Stockholm, in where popular JEE frameworks are identified. The market study resulted in frameworks that are used to implement the solution. The market study shows the most popular frameworks to be Spring and Hibernate.

The solution consists of a tutorial in which as much functionality as possible was implemented from the chosen frameworks within the given time frame. The covered functionality is presented in a list after the Swedish translation.
Designmönster och kodstrukturer med JEE

Målet med examensarbetet är att besvara följande fråga:

För ett medelstort projekt på ett par eller fler personer, vilka webbprogrammeringsteknologier är bra att välja och hur skall man applicera dem?


Lösningen består av en manual i vilken så mycket funktionalitet som möjligt implementerades från de utvalda ramverken inom den givna tidsramen. Den täckta funktionaliteten innefattar:

- Spring Dependency Injection
- Spring MVC
- Spring JSP tags and JSTL
- Spring externalized messages/internationalization
- Spring input validation
- Spring exception handling
- Spring view with Apache Tiles 2
- Spring backed DAOs with Hibernate
- Persisting objects with Hibernate
- Spring caching with Ehcache
- Spring transactions with Hibernate
- Spring security
- Spring AOP Logging with Log4J
The idea of this thesis came about one day when I was in the midst of applying for thesis work at different companies around Stockholm. At that time I realized that most of the thesis work either handled some framed abstract data-logical problem (which is often very interesting in itself) or in surprisingly many cases that someone just wanted cheap labor to build some kind of graphical interface or website. What I wanted was very different. I wanted to be able to accurately answer the question: how do you implement a website using state of the art frameworks and design patterns. The hard part was to realize that that was what I really wanted. This meant that I would have to formulate, research and write the thesis all on my own.

I would like to take the opportunity to thank Simon Ragnar and Per Strand Eklund. Simon helped me to formulate this thesis.
Table of contents

1 Problem description.................................................................1
2 Theory..................................................................................1
3 Analysis and Method description...........................................1
4 Solution................................................................................2
  4.1 Introduction........................................................................2
    4.1.1 Tutorial organization..................................................4
    4.1.2 Covered functionality...............................................5
    4.1.3 Excluded functionality..............................................5
      4.1.3.1 Web services......................................................6
      4.1.3.2 Java Management Extensions..............................6
    4.1.4 What is RoadRantz?...................................................6
  4.2 Getting started...................................................................7
    4.2.1 What goes where?.....................................................7
    4.2.2 About debugging.....................................................8
  4.3 Dependency Injection.....................................................8
  4.4 Front End..........................................................................8
    4.4.1 Spring MVC............................................................9
      4.4.1.1 Dispatcher Servlet.............................................9
      4.4.1.2 Context files......................................................10
      4.4.1.3 Handler Mapping..............................................10
      4.4.1.4 Domain model..................................................11
      4.4.1.5 Command class................................................12
      4.4.1.6 Controllers.......................................................12
      4.4.1.7 Declaring beans...............................................14
      4.4.1.8 Service Layer...................................................15
      4.4.1.9 View Resolver...................................................16
      4.4.1.10 JSP and JSTL..................................................16
    4.4.2 Adding new functionality.........................................18
      4.4.2.1 Externalized messages/Internationalization.............18
      4.4.2.1.1 Recap: add jsp:s, controller and bean declarations...22
      4.4.2.2 Input Validation.................................................24
      4.4.2.3 Exception handling..........................................25
      4.4.2.4 Using Apache Tiles2.........................................26
  4.5 Back End..........................................................................31
    4.5.1 Database management..............................................32
      4.5.1.1 Designing the database.....................................32
      4.5.1.2 Data Access Layer...........................................33
      4.5.1.3 Building DAOs.................................................33
      4.5.1.4 Annotating the Domain for Hibernate....................37
    4.5.2 Caching.....................................................................40
    4.5.3 Transactions............................................................43
  4.6 Security..........................................................................45
    4.6.1 Evil requests............................................................45
    4.6.2 Spring security.........................................................46
1 Problem description

This thesis is aimed at people that need to implement a web-application with Java EE and are familiar with basic Enterprise development with JSPs and Servlets in Java EE.

In web development there exists an acronym, namely LAMP. This acronym is a “technology stack” meant to help developers choose what technologies to use for their new web-application projects. LAMP stands for Linux, Apache, MySQL and PHP. This philosophy of web development has been around a few years, but I find the idea of LAMP very appealing. Why does not something like this exist in the the Java EE community? I have taken most available courses in web development at KTH but I cannot help but feeling that a piece is missing from courses provided in this regard. Therefore the problem statement of this thesis is as follows:

“For a middle-sized web-programming project with a few or more persons using Java EE technologies, what frameworks are good to choose and how should you apply them?”

This thesis aims to answer this question. The technologies and frameworks involved in this solution are products of a market study that is presented in detail in the “Market study” attachment.

2 Theory

The amount of work that has been put down into MCV frameworks for Java during the years is staggering [1]. Therefore there is also a lot of theory behind these ideas. I tried to the best of my ability to find literature that covers the frameworks that I used but in the end it was a combination of about 50-50 literature and online forums, tutorials and APIs that produced the end result.

The material I used can be found in the “References” chapter.


3 Analysis and Method description

The method I chose for this thesis was to write a tutorial. I have not only taken into account while writing this thesis that it should be read by an examiner and get approved with a grade, but also by developers that really need the information to perform a task. The manifestation of this point of view can perhaps most notably be seen in the chapter “Security” in the solution. In that chapter it felt natural to include a short story about an evil request traveling through the different security filters as the filters are being explained. Whether or not this belongs in a thesis has been of secondary concern.

As for determining which frameworks to include into the solution, there was only one way that I felt comfortable with, and that was to perform a market study. Of course this was not the only possible way. I could have analyzed the web (i.e. forums and alike) for
information regarding which frameworks that deserve attention. However to be sure that I
did not accidentally base my decisions on some zealot (zealot = a person with abnormally
high convictions about something) of an obscure framework or get lost in something that
could turn out to be irrelevant, I saw not better way to be sure than to ask the people that
meddle in these questions for a living, i.e. company employees. More on this in the
“Market study” attachment.

From the start it was not clear that the solution should give such and in depth explanation
of only a few frameworks. The original idea was to do the work in several steps. The first
would be to implement a “Hello world” in all frameworks that seemed popular in the
market study. The second step would be to choose a few that seemed better and
implement a more in depth solution. This would certainly be very interesting, but the time
frame given did unfortunately not allow for such an elaborate scheme. It would have been
quite interesting to implement an identical application in many different frameworks and
compare the differences. Instead what naturally evolved was a large implementation of
the most popular framework, with a few helper frameworks on the side.

## 4 Solution

This chapter contains the solution to the problem statement in the shape of a tutorial.

### 4.1 Introduction

**Further definition of the scope:**

Lets revisit the problem statement one last time:

“For a middle-sized web-programming project with a few or more persons using
Java EE technologies, what frameworks are good to choose and how should you
apply them?”

The section “middle-sized web-programming project with a few or more persons” of the
problem statement is meant to indicate the size of the project. Not how such a project
should be organized or how the collaboration between people in the project should be
organized. Furthermore it does not indicate that any collaboration frameworks or
programs will be presented. For short descriptions about such frameworks or programs,
see the Market Research document.

**About the book Spring in Action:**

The principal source of information and methodologies is the book Spring in Action –
Second edition by Craig Walls. There are a few points here worth mentioning: First the
application that I developed was largely based on solutions in this book. To lessen the
complexity of following the ongoings in this book I did not care for changing the type of
application being developed. The application in the book is called RoadRantz,
consequently my application is called RoadRantz. Secondly, the book was first published
in 2008. A lot has happened since then, especially for Spring which is in its third edition.
Craig Walls RoadRantz is developed with Spring 2.0. My RoadRantz is developed with
Spring 3.0.3. Because of this, not all of the solutions in my application are perfectly up to
date. One obvious example of this is the implementation of controllers. Both Craig and I have implemented controllers which extends the controller classes provided by Spring. However in Spring 3.0 this approach is now deprecated and controllers are now defined using annotations. Personally I do not see any real shortcomings in extending Spring provided controllers but it seems to be the general fashion in the Java community to utilize annotations to its fullest degree.

What have I done that is not in the book?

Likewise there were plenty of solutions in the book that did not work with Spring 3.0. Among the solutions that did not work are Internationalization, Security, Tiles integration, JSTL integration, Hibernate integration and Caching. The problems could range from incompatible JAR files to completely redesigned ways of solving matters. In such cases the book where of no help and I hope that the solutions I provide in this thesis will be of practical help to others.

About design patterns:

When presenting a solution in this document, I will not discuss the reasons why the design patterns (program flows) are as they are. The design patterns utilized in this thesis are the product of many hours of trial and error made by many people during the course of many years. If you are interested in this kind of theory I recommend the book Design Patterns, Elements of Reusable Object-oriented Software by Gang of Four.

My own technology stack:

Just for fun I will now present the acronym for my own personal technology stack on which I have built my solution: AOADTSJTEHL. It is based on the technology stacks from the Java EE companies in the market study and on the needs of the system. The acronym transfers to: Any Operating system, Any Database provider, Tomcat, Spring, JSTL, Tiles, Ehcache, Hibernate, Log4j [1]. I would like to add a 'J' to the acronym representing JUnit, but this thesis does not cover testing eventhough it is an essential part of any modern programming done by anyone, anywhere [2]. If you feel uneasy about the sheer number of frameworks involved in this solution, then all I can say is that they are all there to make your life as a developer easier, not harder.

This acronym probably won't become as popular as LAMP, but as you will see, the application that we are about to develop will be highly modularized and easily extended with the functionality that you need for your application (see Covered functionality).

About server, database and operating system:

So why so unspecific when it comes to OS, and DB provider? As for DB providers, just use one of MySQL or PostgreSQL and you will be fine. Operating systems are outside the scope of this thesis but Windows or Linux both do their jobs. My choice fell on PostgreSQL and Windows. Servers on the other hand is a chapter of its own. Before starting a new project I highly recommend that you read up on the different server choices out there and choose the one closest to your needs. My need for this thesis was simplicity and support. Since I knew that I did not need to use EJBs the choice of Tomcat came quite naturally (Tomcat is also included in the Eclipse IDE).
Recommended practices not covered in this tutorial:

My methods of developing might differ from the general methods in the way that I did not use a build framework like Ant or Maven. I simply wrote my code and downloaded the necessary JARs manually. If you are truly trying to start a serious project, I recommend that you start by downloading Maven and incorporate it into your development environment. Secondly (as I previously mentioned) I will not cover testing. But as any modern programmer will tell you, [2] testing is the core of successful software. To this end I recommend that you download and use JUnit to write unit and integration tests to all your classes and controllers.

A few final words:

There are many functionalities that I could have implement in this application. However the application itself is not the interesting part about this thesis, the technology is. That is why there might be almost a hundred lines of code to produce only a single digit on the web page. A lot of what we are will to do is going to be overkill, but in return we will get an application that scales well and has a sound logical structure and, although not empirically proven, is probably free of bugs.

Lets get started:

So where do we start? Well, since we are about to develop a web-application, why not start with the entry point of all browser requests to web-applications developed using Spring-MVC, namely the Dispatcher Servlet. This begins in the Front End chapter. If you feel uneasy about how to configure the development environment, I will present a short tutorial on of what I used to develop my application in the Getting started chapter. Also since Spring relies heavily on Dependency Injection I have written a Dependency Injection Chapter that brings up a more detailed description on the subject than the one found in the Market Study document.

[1] Converted to the same level of complexity as the LAMP acronym this would of course be AOASADJ, where J would stand for Java EE.
[2] Experience both from the market study and from researching the solution.

4.1.1 Tutorial organization

This tutorial is organized in four parts:

• Front end
  Handles the mechanics of communication between web page and application

• Back end
  Handles the mechanics of data handling with databases

• Security

• Cross Cutting Concerns
  Goes through how to use AOP to implement cross cutting concerns
4.1.2 Covered functionality

This list explains what functionality is covered in this tutorial and in the right order:

- Spring Dependency Injection
- Spring MVC
- Spring JSP tags and JSTL
- Spring externalized messages/internationalization
- Spring input validation
- Spring exception handling
- Spring view with Tiles
- Spring backed DAOs with Hibernate
- Persisting objects with Hibernate
- Spring caching with Ehcache
- Spring transactions with Hibernate
- Spring security
- Spring AOP Logging with Log4J

4.1.3 Excluded functionality

The following list displays examples of Spring functionality that have been excluded from the solution in this thesis. Solutions to all but cloud computing can be found in Spring in action – Second Edition. The decision to exclude a certain functionality is based on the necessity of certain functionality for the implementation of a basic web page and/or time deficiency.

- Spring Cloud Computing
  - [http://www.cloudfoundry.com/] 21/10-2010
- Spring Java Message Service (JMS) support
- Spring Web Services support
  - Explained below
- Spring Java Management Extensions (JMX) support
  - Explained below
- Spring Non HTML output
- Spring Email
- Spring Scheduling
• Spring Web Flow with forms
• Spring Testing with mock objects

### 4.1.3.1 Web services

One major concern for me in this theses is that I did not manage to find a way to work in Web Services to the application in a reasonable time frame. However Web Services are widely used in today's business world. Therefore I have summarized a list of what remote service technologies that Spring supports:

- **Synchronous Communication**
  - Remote Method Invocation (RMI)
  - Caucho's Hessian and Burlap
  - Spring's own HTTP invoker
  - Web services using SOAP and JAX-RPC
  - Spring's own contract first Web Services (Spring WS)
  - EJB

- **Asynchronous Communication**
  - Java Message Service

### 4.1.3.2 Java Management Extensions

Spring supports JMX that is a standard part of the Java 5 distribution. So what is JMX? Suppose that once your application has been built you would like to be able to manage, monitor and configure the application while it is in production and running. In Spring you can make use of JMX in combination with, for example, RMI to expose methods and attributes inside your application. This would enable you to change key attributes to configure the behavior, and to call functions that help you manage your application. What's more is that with the help of JMX you can enable your application to send notifications when a certain event occurs. An example of this would be to send an SMS to your cellphone when the one millionth user registers.

### 4.1.4 What is RoadRantz?

RoadRants is an application that allows angry drivers to express their rage about other drivers in a more civilized manner than vocally on the road. In RoadRantz the user is able to write down his or her comments (rants) about other vehicles and the rant is available for all visitors to see.
4.2 Getting started

The first two things that needs to be specified are operating system and database provider. These two “parameters” are free choices because they will not impact the solution given in this thesis.

Eclipse is a good choice for IDE. To easily start a new web application project, choose to start a new Dynamic Web Project. Eclipse has built in Tomcat servers that can be deployed when running a project. This requires some minor manual configuration.

The JAR files used in this solution [1] are packaged in the RoadRantz.war file accompanying this thesis. Put them in your WEB-INF/lib directory and include them in the Eclipse project by: right click the project name → Properties → Java Build Path → Libraries → Add JARs...

Next: database drivers. You can choose to download the newest drivers from your database provider, or if you use PostgreSQL you can use the one provided in this thesis. The driver JAR is put in your servers lib directory. It is not enough to put it in your WEB-INF/lib directory because it is the server that need the JAR, not your application.

The following xml are examples of database connection configuration in the server for a PostgreSQL database.

context.xml:

```xml
<Resource
   name="jdbc/postgres"
   auth="Container"
   type="javax.sql.DataSource"
   driverClassName="org.postgresql.Driver"
   url="jdbc:postgresql://127.0.0.1:5432/roadRantzDatabase"
   username="postgres"
   password="bollkontroll"
   maxActive="20"
   maxIdle="10"
   maxWait="-1" />
```

web.xml:

```xml
<resource-ref>
   <description>postgreSQL Datasource</description>
   <res-ref-name>jdbc/postgres</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

[1] The JAR files are all new as of august 2010 and are compatible with each other.

4.2.1 What goes where?

The best way to understand what files goes where in the Eclipse IDE is to import the RoadRantz.war project file and explore the file structure. The “Java Resources: src” folder of the Eclipse project is equal to the WEB-INF/classes folder in the server folder structure and is also the root of the classpath.
4.2.2 About debugging

There are two debug settings available. Either use slf4j-log4j12-1.6.1.jar or use slf4j-simple-1.6.1.jar. To use the one or the other, just mark the one you do not wish to use with “.jar_” at the end. Log4J can give very verbose server output, while the “simple” variant keeps it more easily read. Log4J has a properties file called logging.properties that exists in the WEB-INF/classes folder. The properties in this file can be changed between INFO and DEBUG. INFO giving less verbose information and DEBUG gives practically all available information. You might wish to use both at one time or another depending on the circumstances.

4.3 Dependency Injection

So what is a dependency? A dependency is simply a class B that is used by some other class A. The word “dependency” is derived from the fact that class A is dependent on class B to perform its function.

When applying Dependency Injection objects are given their dependencies at creation time by some container, in our case the Spring container. This is in contrast to the traditional way of programming where objects are themselves responsible for obtaining references to the objects they are collaborating with. This means that all objects has to know how to instantiate their dependencies. This creates hard coupling between objects and can lead to difficulties when you need to swap out the implementation of a dependency. This is because you need to change the actual code that instantiates the dependency.

With DI on the other hand a dependency can be swapped out without the depending object knowing the difference. This is accomplished with the help of interfaces. Suppose the following:

- Class B is injected to class A though DI.
- Class A only knows about class B though an interface.

Then the implementation of class B can be freely swapped without class A ever knowing the difference.

4.4 Front End

In this chapter we will cover the inner workings of the Spring MVC module. However even though Spring MVC is a great framework, it is not the only web framework out there. WebWork, Tapestry, Struts and JSF are all widely used web frameworks that deserve some credit. However since Spring has so much to offer the developer, things like DI, AOP, a rich database management support, JMX support and Web Services support, then we would only add complexity by using a stand-alone web framework. If however you have previous knowledge about any of these frameworks and is hesitant to switch to a new one, Spring also offers support to make these frameworks Spring-aware.

Having said that, lets turn our attention how to build a robust MVC architecture using Spring.
4.4.1 Spring MVC

In Spring, all HTTP requests to an application go through a single point of entry. This single point of entry is called the Dispatcher Servlet. The Dispatcher Servlet dispatches the request to a controller, which in turn returns a model and view back to the Dispatcher Servlet. Then the request is sent to the view determined by the controller accompanied by a model for that view.

![Diagram of the flow of a request in Spring MVC](image)

**Fig.1: The flow of a request in Spring MVC**

### 4.4.1.1 Dispatcher Servlet

**WEB-INF/web.xml:**

```xml
<servlet>
  <servlet-name>roadrantz</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>roadrantz</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

The `<url-pattern>*.htm</url-pattern>` tag tells that all requests ending with `.htm` will be handled by the Dispatcher Servlet. `.htm` is arbitrary and can be any URL pattern.

The name specified in the `<servlet-name>` tag is significant since the Dispatcher Servlet will load the Spring application context from an xml file with that name followed by “-servlet.xml”. In our case that file will be roadrantz-servlet.xml.

### 4.4.1.2 Context files

All Spring application context files will be put in the WEB-INF/ folder. These files are where Spring loads all application context configuration. To be able to split up the configuration into several context files we need to configure a Context
Loader. The following xml defines the context files used in the RoadRantz application.

**WEB-INF/web.xml**

```xml
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/roadrantz-service.xml
    /WEB-INF/roadrantz-data.xml
    /WEB-INF/roadrantz-security.xml
    /WEB-INF/roadrantz-cache.xml
    /WEB-INF/roadrantz-transaction.xml
    /WEB-INF/roadrantz-logging.xml
  </param-value>
</context-param>
```

Here we have defined a quite large list of context xml files. These are all the Spring context xml files that we will use throughout this solution.

All Spring context files needs to be declared with a namespace. In the case of the roadrantz-servlet.xml file, all configuration is put between the following tags:

**WEB-INF/roadrantz-servlet.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
<!--put configuration here-->
</beans>
```

Next we will start to fill our roadrantz-servlet.xml with Handler mappers, View resolvers, controller and other beans in the Spring MVC.

### 4.4.1.3 Handler Mapping

Handler Mapping is the second step for our request. The handler mapping maps the an request URL to a controller. There are several Handler Mappers available in Spring but here we will configure a SimpleUrlHandlerMapping:

**WEB-INF/roadrantz-servlet.xml**

```xml
<bean id="simpleUrlMapping"
      class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/home.htm">homePageController</prop>
      <prop key="/addRant.htm">addRantFormController</prop>
      <prop key="/changeLang.htm">changeLangFormController</prop>
      <prop key="/login.htm">loginPageController</prop>
    </props>
  </property>
</bean>
```
All we have to do is to put this bean declaration in the xml file and Spring will use it as Handler Mapping. You might wonder where all the other controllers come from. Don't fret, they will be added slowly throughout this solution. And if you notice later on that you don't want to use them, you can remove the ones that are not used.

### 4.4.1.4 Domain model

The domain of an application reflects the logical structure of the problem that the application is trying to solve. The design of the Domain model is one of the most important parts of designing a new web application, or an application of any kind that uses a database in combination with an ORM for that matter. The reason for this is that when using a Object Relational Mapping tool like Hibernate, the Domain is mirrored in the database. Hence, a bad domain gives a bad database. In our case the user is supposed to be able to enter a rant about a vehicle. But if we think one step ahead, we realize that there might be several rants about one vehicle. Hence we have a many-to-one relationship. This will become more important when we design our database and prep our domain model for persistence to the database. Right now though it is enough to conclude that we will have to create a class Rant, that can store a rant, a date and that has an object Vehicle:

```java
package com.roadrantz.domain;

public class Rant{
    private String rantText;
    private String rantDate;
    private Vehicle vehicle;
    public Rant() {}

    public String getRantText(){return rantText;}
    public void setRantText(String rantText){this.rantText = rantText;}

    public String getRantDate(){return rantDate;}
    public void setRantDate(String rantDate){this.rantDate = rantDate;}

    public Vehicle getVehicle(){return vehicle;}
    public void setVehicle(Vehicle vehicle){this.vehicle = vehicle;}
}
```

```java
package com.roadrantz.domain;

public class Vehicle{
    private String plateNumber;
```
private String state;

public Vehicle() {}

public String getPlateNumber(){return plateNumber;}

public void setPlateNumber(String plateNumber)
    {this.plateNumber = plateNumber;}

public String getState(){return state;}

public void setState(String state)
    {this.state = state;}

4.4.1.5 Command class

A command class is used between the Controller and View layers of an Spring application as a carrier of information. How this works will be explained below. The RoadRantz application will use Rant and Vehicle as a command classes.

4.4.1.6 Controllers

We have configured our Dispatcher Servlet to dispatch requests to different controllers. Now we have to build those controllers. There are many different controllers in Spring but in the RoadRantz application we will use two types of controllers: SimpleFormController and AbstractController. As you can see SimpleFormController is deprecated in Spring 3.0 in favor of an annotated version. Here is the code for the HomePageController of the RoadRantz application:

```java
package com.roadrantz.mvc;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import com.roadrantz.domain.Rant;
import com.roadrantz.service.RantService;

public class HomePageController extends AbstractController{
    public HomePageController(){
    }

    protected ModelAndView handleRequestInternal(
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception{
        List<Rant> recentRants = rantService.getAllRants();
        return new ModelAndView("home", "rants", recentRants);
    }

    private RantService rantService;
    public void setRantService(RantService rantService){
        this.rantService = rantService;
    }
}
```

This controller is used to populate the /home.htm site with a model. For now, do not worry about where the rantService is coming from. In the AbstractController
there is only one real method at work, namely handleRequestInternal(). This method has to return a ModelAndView. In our case the ModelAndView is populated with a view called “home”, and a model called “rants”. The model is a List of rants received from the RoadRantz service layer.

Next we have the AddRantFormController. It is of the type SimpleFormController:

```java
package com.roadrantz.mvc;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.roadrantz.domain.Rant;
import com.roadrantz.domain.Vehicle;
import com.roadrantz.service.RantService;

@SuppressWarnings("deprecation")
public class AddRantFormController extends SimpleFormController{
    private static final String[] ALL_STATES = {"SV", "SU", "KTH"};

    public AddRantFormController(){
        setCommandClass(Rant.class);
        setCommandName("rant");
    }

    protected Object formBackingObject(
        HttpServletRequest request)
        throws Exception{
        Rant rantForm = (Rant) super.formBackingObject(request);
        rantForm.setVehicle(new Vehicle());
        return rantForm;
    }

    protected Map<String, String[]> referenceData(
        HttpServletRequest request)
        throws Exception{
        Map<String, String[]> referenceData =
            new HashMap<String, String[]>();
        referenceData.put("states", ALL_STATES);
        return referenceData;
    }

    protected ModelAndView onSubmit(
        Object command, BindException bindException)
        throws Exception{
        Rant rant = (Rant) command;
        rantService.addRant(rant);
        return new ModelAndView(getSuccessView());
    }

    private RantService rantService;
    public void setRantService(RantService rantService){
        this.rantService = rantService;
    }
}
```

This controller is used to populate the /addRant.htm part of the RoadRantz application with a list of states and to process the response from that page. Here
we have several methods at work. The constructor tells Spring what command class to use in the JSP.

The formBackingObject() function is not always necessary. But in the RoadRantz application we must instantiate a new Vehicle for every Rant so defining Rant as the command class is not enough in this case.

Next we have the referenceData() function. This function simply populates a Map to be used in the JSP.

When the user has entered a rant, the request will go to the onSubmit() function. Here we forward the complete rant to the service layer and returns a success view, telling the user that all has gone well. The success view will be defined in the controllers bean configuration.

### 4.4.1.7 Declaring beans

If an object needs to be known in the Spring context, then it is declared as a bean. Now we will declare our controllers as beans in the Spring context:

```
roadrantz-servlet.xml
<bean name="homePageController" class="com.roadrantz.mvc.HomePageController">
  <property name="rantService" ref="rantService" />
</bean>

<bean name="addRantFormController" class="com.roadrantz.mvc.AddRantFormController">
  <property name="rantService" ref="rantService" />
  <property name="formView" value="addRant" />
  <property name="successView" value="rantAdded" />
</bean>
```

The formView defines what view to respond with in the first request for the form site. In our case we want to display a site which prompts the user to submit a rant. The successView defines what view the user should be sent to when the form process is complete.

We also have to declare the bean of our RantService. There is nothing fancy about declaring a service. It is just a normal POJO. So we do not need any special modifications to our context file namespace:

```
roadrantz-service.xml
<?xml version="1.0" encoding="UTF-8"?>
  <bean name="rantService" class="com.roadrantz.service.RantServiceImpl" />
  <property name="rantDao" ref="rantDao" />
</bean>
```

4.4.1.8 Service Layer

The service layer exists to separate as much logic as possible from the controllers and to provide reusable methods. Later we will use the service layer to define transactions in our application. Here is the code for the RantService class so far:

```java
package com.roadrantz.service;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import com.roadrantz.domain.Rant;
import com.roadrantz.domain.Vehicle;
import com.roadrantz.dao.RantDao;

public class RantServiceImpl implements RantService {
    public List<Rant> getAllRants() {
        return rantDao.getAllRants();
    }

    public void addRant(Rant rant) {
        Vehicle rantVehicle = rant.getVehicle();
        Vehicle existingVehicle = rantDao.findVehicleByPlate(rantVehicle.getPlateNumber());
        if (existingVehicle != null)
            rant.setVehicle(existingVehicle);
        Set<Rant> rants = new HashSet<Rant>();
        rants.add(rant);
        rant.getVehicle().setRants(rants);
        rant.setRantDate(new SimpleDateFormat("yyyy/MM/dd").format(new Date()));
        rantDao.saveRant(rant);
    }

    public List<Rant> getRantsForDay(String date) {
        return rantDao.getRantsForDay(date);
    }

    private RantDao rantDao;
    public void setRantDao(RantDao rantDao) {
        this.rantDao = rantDao;
    }
}
```

As you can see the Service Layer communicates with a DAO layer. We will get back to DAOs later. For now I recommend that you just create and return a List of a few rants in the getAllRants() function, and maybe put a simple System.out.println() in the addRant() function to be sure that it works. Also you can remove the RantDao Dependency Injection in the bean declaration, or create an empty DAO class to be injected if you wish.

Just to not leave any details behind, here is the code for the RantService interface:

```java
package com.roadrantz.service;
```
import java.util.List;
import com.roadrantz.domain.Rant;

public interface RantService {
    public List<Rant> getAllRants();
    public void addRant(Rant rant);
}

### 4.4.1.9 View Resolver

Now that the Dispatcher Servlet has its Model and View, it needs to know what actual view to use to render the response. In our HomePageController the view was named “home”, but this is just a logical name. We need to map that name to a specific JSP. This is where the View Resolver comes in to the picture. There are many View Resolvers in Spring and we will look at one more later, but for now we will use InternalResourceViewResolver:

```
roadrantz-servlet.xml

<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/" />
  <property name="suffix" value=".jsp" />
</bean>
```

The InternalResourceViewResolver takes the logical name of the view and adds a prefix and suffix to it. In our case the view “home” will be renamed to /WEB-INF/jsp/home.jsp. This might seem unnecessary. Why not give it the right name from the start? The reason for this is that the logical name “home” might be something that can be used in more ways than one. Later we will see how to change the View Resolver to something that makes use of the Apache Tiles2 framework for the view tier of the application.

### 4.4.1.10 JSP and JSTL

Now we need to create our home.jsp page. Put all your .jsp files in a WEB-INF/jsp folder. Here is home.jsp at this stage of the project:

```
home.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Rantz</title>
</head>
<body>
<h2>Welcome to rantz</h2>
<h3>All rantz:</h3>
<ul>
    <c:forEach items="${rants}" var="rant">
        <li>${rant.vehicle.state}</li>
    </c:forEach>
</ul>
```

16
The JSTL tag is defined with the prefix “c”. JSTL makes the code less verbose and easier to read in combination with HTML than pure Java code. Notice that the variable name $\{rants\}$ is the same as the logical name “rants” returned in the ModelAndView from the HomePageController.

Next we have addRant.jsp:

```
addRant.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add Rant</title>
</head>
<body>
    <h2>Enter a rant...</h2>
    <form:form method="POST" action="addRant.htm" commandName="rant">
        <br>
        <c:forEach var="state" items="${states}">
            <option value="${state}">
            <c:out value="${state}" />
        </option>
        </c:forEach>
        <br>
        <label for="vehicle.plateNumber">Plate #:</label>
        <br>
        <form:input path="vehicle.plateNumber" />
        <br>
        <label for="rantText">Rant text:</label>
        <br>
        <form:textarea path="rantText" rows="5" cols="50" />
        <br>
        <input type="submit" />
    </form:form>
</body>
</html>
```

Notice the Spring specific form tag. This is used to bind the form elements to the command class. We will use a few more Spring specific JSP tags later on for internationalization, form validation and security.

Lastly we defined the AddRantFormController to have a success view named rantAdded. Nothing new is introduced in this jsp.

```
rantAdded.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" %>
```
NOTE! From now on I will leave out the standard header in the beginning of each jsp file to save space.

### 4.4.2 Adding new functionality

There... now we have written an application that works. Granted, it does not look very good and it doesn't actually do anything, but at least it generates output to the user, and that is the least common denominator for web pages. Now we will start adding new functionality to our page. The idea here is that you can pick and choose the functionality and chapters you like to your application and it will work. If, however, you decide to skip ahead a few chapters to add that hip functionality you always wanted and for some reason it doesn't work, it might be a good idea to back up a few chapters to see if there is something you missed. I'm saying this because I have not tried to add the functionality in this thesis in a random order and hence I cannot promise that anything works that I have not tried. But it should work =).

#### 4.4.2.1 Externalized messages/Internationalization

So what is internationalization really? It is the ability for the user to change language with a click and the ability for the programmer to implement this in the easiest fashion possible. To kick start our internationalization efforts we need to add a few beans:

```xml
roadrantz-servlet.xml

<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames">
    <list>
      <value>messages</value>
      <value>errors</value>
    </list>
  </property>
</bean>

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver"/>

<bean id="localeChangeInterceptor"
      class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
  <property name="paramName" value="siteLang"/>
</bean>
```
The messageSource bean defines that we will have two types of message sources, namely messages and errors. Messages are for standard text on the page and errors are text messages that will appear if we encounter an error whilst entering data to a form. What this means in practice is that we will have two sets of files that will be placed in the WEB-INF/classes folder. These files will be named messages.properties and errors.properties respectively.

The localeResolver is a bean that enables the application to specify several different message files and several different errors files depending on what language or locale the user specifies. For example, if the user specifies Spanish as the preferred language/locale of choice, then the application will look for messages and errors in files ending with _es. i.e. messages_es.properties and errors_es.properties. Soon we will see how to configure Spring to make this work.

Last but not least we have the localeChangeInterceptor. This bean defines what POST variables the application should look for to change the locale. In the RoadRantz application we have defined the locale changer POST variable to be called “siteLang”.

Now we need to do four things:

1. We need to change our jsp files to make use of these externalized messages.
2. We need to create our properties files.
3. We need to create a page that actually enables the user to change the language.
4. Last we need to start thinking about input validation so that we may know when to display errors in our form pages.

Part three and four will be presented in the next two chapters since part three is really a recap of what we have already been doing, namely adding jsp:s controllers and beans. And part four deserves to be a chapter of its own.

I apologize that the bold text does not stand out more in the jsp code below but everything marked in bold is new to the jsp files.

home.jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Rantz</title>
</head>
<body>
<h2><spring:message code="home.welcome" /></h2>
<h3><spring:message code="home.recent" /></h3>
<ul>
     <c:forEach items="${rants}" var="rant">
     <li>
        <c:out value="${rant.vehicle.state}"/>
        <c:out value="${rant.vehicle.plateNumber}"/>
     </li>
     </c:forEach>
</ul>
<c:out value="${rant.rantText}"/>
<c:out value="${rant.rantDate}"/>
</li>
</c:forEach>
</ul>
<a href="/RoadRantz/addRant.htm">
<spring:message code="home.addRant" />
</a>
<br />
<a href="/RoadRantz/changeLang.htm">
<spring:message code="home.changeLang" />
</a>
<br />
</body>
</html>

addRant.jsp

<!-- Taglibs -->
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<style>
.error {
  color: red;
}
</style>
<title>Add Rant</title>
</head>
<body>

<h2>
<spring:message code="field.addRant" />
</h2>
<form:form method="POST" action="/addRant.htm" commandName="rant">
<b>
<spring:message code="field.state" />
</b>
<select name="vehicle.state">
<c:forEach var="state" items="${states}">
<option value="${state}">
<c:out value="${state}" />
</option>
</c:forEach>
</select>
<form:errors path="vehicle.state" cssClass="error"/>
<br />

<b>
<spring:message code="field.plateNumber" />
</b>
<form:input path="vehicle.plateNumber" />
<form:errors path="vehicle.plateNumber" cssClass="error"/>
<br />

<b>
<spring:message code="field.rantText" />
</b>
<form:textarea path="rantText" rows="5" cols="50" />
<form:errors path="rantText" cssClass="error"/>
<br />
<input type="submit" />
</form:form>
</body>
</html>

rantAdded.jsp

<!-- Taglibs -->
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<h2>
<spring:message code="rantAdded.rantAdded" />
</h2>
<a href="/RoadRantz/home.htm"/>

20
WEB-INF/classes/messages.properties

home.welcome=Welcome to rantz
home.recent=All rantz:
home.addRant=Add a rant
home.changeLang=Change language
home.logout=Log out

field.addRant=Enter a rant...
field.state=State:
field.plateNumber=Plate #:
field.rantText=Rant text:
rantAdded.rantAdded=Rant was added...
rantAdded.home=Home

changeLang.choose=Choose language:
langChanged.langChanged=Language was changed...

WEB-INF/classes/messages_es.properties

home.welcome=Rantz el vocabulario
home.recent=El completado del rantz
home.addRant=El ranto del addo
home.changeLang=El languero del changeo
home.logout=Cierre de sesión

field.addRant=Por que?
field.state=Maria!? 
field.plateNumber=Es le corazón
field.rantText=Mirar le septiembre
field.rantText=Addodar del rantular
rantAdded.home=Zú casa

changeLang.choose=El languero del choosluerdo
langChanged.langChanged=Changelurdo el languero...

WEB-INF/classes/errors.properties

required.state=State is required.
required.plateNumber=License plate number is required.
required.rantText=Rant text is required.
invalid.plateNumber={0} is an invalid license plate number.

WEB-INF/classes/errors_es.properties

required.state=El estado se requiere.
required.plateNumber=El número de la matrícula se requiere.
required.rantText=El texto de lenguaje declaratorio se requiere.
invalid.plateNumber={0} en un número inválido de matrícula.

The {0} variable in the errors.properties files specifies the value of the invalid form field input by the user. More on this will be explained in the Input Validation chapter.

4.4.2.1 Recap: add jsp:s, controller and bean declarations

Now we need to add the functionality that enables the user to actually change the language. Try to start by applying what you have learned so far by adding everything that is needed for the internationalization functionality yourself. If you
think that that is a stupid idea or if you get stuck adding the functionality, this chapter will walk you though the process once more. Let's start with the controller:

```java
package com.roadrantz.mvc;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.roadrantz.domain.Language;
@SuppressWarnings("deprecation")
public class ChangeLangFormController extends SimpleFormController{
    private static final String[] ALL_LANGUAGE_NAMES = {
        "English", "Spanish (mocked)";
    }
    private static final String[] ALL_LANGUAGE_ABBRS = {
        "", "es"};
    public ChangeLangFormController(){
        setCommandClass(Language.class);
        setCommandName("lang");
    }
    protected Map<String, String[]> referenceData(HttpServletRequest request)
        throws Exception{
        Map<String, String[]> referenceData = new HashMap<String, String[]>(){
            referenceData.put("language_names", ALL_LANGUAGE_NAMES);
            referenceData.put("language_abbrs", ALL_LANGUAGE_ABBRS);
            return referenceData;
        }
    }
    protected ModelAndView onSubmit(
        Object command, BindException bindException) throws Exception{
        return new ModelAndView(getSuccessView());
    }
}
```

The command class Language is not needed. Spring will look for the locale changer POST variable “siteLang” automatically. However when using a SimpleFormController, Spring requires you to specify a command class. This problem is solved by using an empty class Language:

```java
package com.roadrantz.domain;
public class Language {};
```

NOTE: A lot of work has been spent to make sure that workarounds are kept to the absolute bare minimum in this thesis. Never the less I consider this Language class to be a workaround. However one possible use of this class might be if you need to use the selected language inside the application (in a controller or otherwise). This is the only workaround that you will find in this thesis.
The JSTL code in changeLang.jsp will use the String arrays to create a drop-down list displaying the two different languages and also applying the language abbreviations that correspond to the file endings on the message and error properties files to the value elements of the option list:

changeLang.jsp

```jsp
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  </head>
  <body>
    <form:form method="POST" action="changeLang.htm" commandName="lang">
      <b><spring:message code="changeLang.choose" /></b>
      <select name="siteLang">
        <c:forEach var="language_abbr" items="${language_abbrs}" varStatus="rowCounter">
          <option value="<c:out value="${language_abbr}" />">
            <c:out value="${language_names[rowCounter.count-1]}" />
          </option>
        </c:forEach>
      </select>
      <input type="submit" />
    </form:form>
  </body>
</html>
```

langChanged.jsp

```jsp
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  </head>
  <body>
    <h2><spring:message code="langChanged.langChanged" /></h2>
    <a href="/RoadRantz/home.htm"><spring:message code="rantAdded.home" /></a>
  </body>
</html>
```

Now all that is left is to define the controller bean:

roadrantz-servlet.xml

```xml
<bean name="changeLangFormController" class="com.roadrantz.mvc.ChangeLangFormController">
  <property name="formView" value="changeLang" />
  <property name="successView" value="langChanged" />
</bean>
```

There you have it. Fully functioning internationalization of your application. If you wish to add more languages, just write more .properties files and add the language to the String arrays in the ChangeLangFormController. Easy as Pie.
4.4.2.2 Input Validation

So what happens if the user types in wrong or no information in the addRant form? Our rants will not be complete. What we need is a validator that checks the submitted information. This chapter explains how to use Input Validation in combination with externalized messages. However you do not have to implement externalized messages for this to work. Just ignore everything that has to do with externalized messages and it will work fine. We start by updating the bean declaration of our AddRantFormController:

```xml
<bean name="addRantFormController"
    class="com.roadrantz.mvc.AddRantFormController">
    <property name="rantService" ref="rantService" />
    <property name="formView" value="addRant" />
    <property name="successView" value="rantAdded" />
    <property name="validator">
        <bean class="com.roadrantz.mvc.RantValidator" />
    </property>
</bean>
```

Notice the added property validator that refers to the class RantValidator. Now we will define that class:

```java
package com.roadrantz.mvc;
import org.apache.orotext.perl.Perl5Util;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import com.roadrantz.domain.Rant;

public class RantValidator implements Validator{
    public boolean supports(Class clazz){
        return clazz.equals(Rant.class);
    }

    public void validate(Object command, Errors errors){
        Rant rant = (Rant) command;
        ValidationUtils.rejectIfEmpty(errors, "vehicle.state", "required.state", "State is required");
        ValidationUtils.rejectIfEmpty(errors, "vehicle.plateNumber", "required.plateNumber", "The license plate number is required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "rantText", "required.rantText", "You must enter some rant text.");
        if(!errors.hasFieldErrors("vehicle.plateNumber")){
            validatePlateNumber(rant.getVehicle().getPlateNumber(), errors);
        }
    }
}
```
private static final String PLATE_REGEXP = "/[a-z0-9]{2,6}/i";
private void validatePlateNumber(String plateNumber, Errors errors) {
    Perl5Util perl5Util = new Perl5Util();
    if (!perl5Util.match(PLATE_REGEXP, plateNumber)) {
        errors.rejectValue("vehicle.plateNumber",
                         "invalid.plateNumber",
                         new Object[] {plateNumber},
                         "Invalid license plate number."
        );
    }
}

This class might look big and scary but all it does is checking the three form input
variables so that they are in the right format. The rejectIfEmpty() function takes
four arguments. The first being an errors object that is used to store what errors
occurs. The second is a String specifying the specific field the is being checked.
The third is what, if any, externalized message to use as a message to the user. The
fourth is the default message to the user if no externalized messages could be
found. If the field in question is empty the field will be rejected by the function.

The hasFieldErrors() function checks if a specific field has been given any errors.
The rejectValue() function plainly reject the value given to it. The Object array is
used by the externalized message invalid.plateNumber that is defined in an
externalized message file.

4.4.2.3 Exception handling

Now what would happen if something went wrong in the code of our beautiful
application? Well the user would get a big stack trace straight in the face. And no
user would like that unless he or she is a big web-application geek like us. So
what we need to do is to make sure that a nice friendly message is displayed
telling the user that something has gone wrong and maybe what has gone wrong.
This can easily be done with Spring's exception handling. We start by declaring a
bean that tells Spring what exceptions should be mapped to what jsp:s:

roadrantz-servlet.xml

<bean id="exceptionResolver" class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.Exception">friendlyError</prop>
        </props>
    </property>
</bean>

Since java.lang.Exception is the mother of all exceptions, all exceptions in our
application will be forwarded to the view with the logical name of
“friendlyError”. Our view resolver will pick up this logical name and rename it
to /WEB-INF/jsp/friendlyError.jsp. Now we need to write the jsp:
fiendlyError.jsp

```jsp
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Fiendly error</title>
</head>
<body>
<h4><spring:message code="friendlyError.friendlyError" /></h4>
<a href="/RoadRantz/home.htm">
  <spring:message code="rantAdded.home" />
</a>
<br />
<h6><c:out value="${exception}" /></h6>
</body>
</html>
```

Add `friendlyError.friendlyError=There has been an application error`. To your externalized messages properties files. You will have to make up some mock Spanish on your own now.

The `{exception}` variable is actually the exception being thrown by the application. Printing it in this way will display a short description of the exception along with some useful additional information. In my opinion not too much for a common user to take offense and very useful for the developer.

### 4.4.2.4 Using Apache Tiles2

So what is this Tiles framework? If you haven't read the Market Study part of this thesis then let me recap:

"Tiles is a framework that until recently was a part of Struts, and is now known as Apache Tiles. Tiles is a framework that helps constructing a JSP page that is divided into tiles that can include other JSP pages."

- [Spring in action, Second Edition - Book]

Another way of putting it is that Tiles provides a neat way of dividing up your jsp:s into sections where each section is a self sustaining jsp. Tiles also provides special Tiles Controllers for each section but we will get in to that later.

Let's start by defining the outline of our web page:

rantzTemplate.jsp

```jsp
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>
  <tiles:getAsString name="title" />
</title>
</head>
<body>
```
By some unfathomable reason Eclipse decided not to color the code for rantzTemplate.jsp so will have to make due in back and white.

The outline of our web page is simply divided into three horizontal parts. A header, content and a footer. These three areas are defined using the tiles insertAttribute tag. The title of the page is defined with the help of tiles getAsString tag.

Now we need to tell Spring that we have started dealing with a Tiles based view. Lets declare some beans:

roadrantz-servlet.xml

```xml
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.tiles2.TilesViewResolver" />

<bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
    <property name="definitions">
        <list>
            <value>/WEB-INF/roadrantz-tiles.xml</value>
        </list>
    </property>
    <property name="preparerFactoryClass" value="org.springframework.web.servlet.view.tiles2.SpringBeanPreparerFactory"/>
</bean>
```

In this bean declaration we are dealing with a brand new viewResolver, namely TilesViewResolver. This means that you will have to erase or comment out the InternalResourceViewResolver we defined earlier.

As you might have gathered from the tilesConfigurer bean definition, we are going to create a separate roadrantz-tiles.xml file for dealing with tiles. Also we need a SpringBeanPreparerFactory to be able to declare Tile controllers accessible in the Spring context.

Lets carry on with the tiles xml:

roadrantz-tiles.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC 
"-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN" 
"http://tiles.apache.org/dtds/tiles-config_2_1.dtd">
```
<definition name="template" template="/WEB-INF/jsp/rantzTemplate.jsp">
  <put-attribute name="title" value="RoadRantz" />
  <put-attribute name="header" value=".header" />
  <put-attribute name="content" value="/WEB-INF/jsp/defaultContent.jsp" />
  <put-attribute name="footer" value="/WEB-INF/jsp/footer.jsp" />
</definition>

<definition name=".header" template="/WEB-INF/jsp/header.jsp" preparer="headerTileController"/>

<definition name="home" extends="template">
  <put-attribute name="title" value="Welcome to RoadRantz" />
  <put-attribute name="content" value="/WEB-INF/jsp/home.jsp" />
</definition>

<definition name="addRant" extends="template">
  <put-attribute name="content" value="/WEB-INF/jsp/addRant.jsp" />
</definition>

<definition name="changeLang" extends="template">
  <put-attribute name="content" value="/WEB-INF/jsp/changeLang.jsp" />
</definition>

<definition name="rantAdded" extends="template">
  <put-attribute name="content" value="/WEB-INF/jsp/rantAdded.jsp" />
</definition>

<definition name="langChanged" extends="template">
  <put-attribute name="content" value="/WEB-INF/jsp/langChanged.jsp" />
</definition>

<definition name="friendlyError" extends="template">
  <put-attribute name="content" value="/WEB-INF/jsp/friendlyError.jsp" />
</definition>

</tiles-definitions>

Granted, it seems to be a bit more work to configure the view resolver in a Tiles application. But we will rarely change anything in this file once we have entered it.

To start with we have a special namespace in this xml file, namely http://tiles.apache.org/dtds/tiles-config_2_1.dtd (21/10-2010). This namespace enables us to use the Tiles specific tags that are used in this xml.

The most important definition in this xml is the one named “template”. It defines that rantzTemplate.jsp is the jsp we are going to use as the template of the application. Actually the name “template” is not important, we could have named the definition anything. It also defines that we have a title, header, content and footer and what values those should have. All these values can be overridden by other definitions that extend “template”. As you can see with the “home” definition it overrides the values for both title and content.

The default setting for the content tile is a file called defaultContent.jsp. However we will always navigate the page by specifying a new content tile so we will never need to create the defaultContent.jsp file.
IMPORTANT: The name of the definitions corresponds exactly to the logical view names defined in the ModelAndView returned from the controllers.

Now the last weird looking thing is the definition `.header`. It defines that the header part of the template should be populated with header.jsp and that header.jsp has a special preparer named HeaderTilesController. Lets declare this controller in the Spring context:

```
roadrantz-servlet.xml

<bean name="headerTileController"
     class="com.roadrantz.tiles.HeaderTileController">
    <property name="rantService" ref="rantService"/>
</bean>
```

So why go through the trouble of creating a special Tiles controller for the header. Well wouldn't it be be nice to be able to have some dynamic content showing in a tile that is not called directly by a ModelAndView? To be able to do that one could put the dynamic content in the ModelAndView itself or in the referenceData() function of a controller. But then you would have to put that code in all controllers in the application. You could hide most of the code by extending a helper class or creating a utility class, but this adds unnecessary complexity to the code for all controllers.

A Tiles controller is handy when a tile needs dynamic information and it is not called directly by a Spring controller. Lets see what this special controller looks like:

```
HeaderTileController

package com.roadrantz.tiles;
import com.roadrantz.service.RantService;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.tiles.AttributeContext;
import org.apache.tiles.Attribute;
import org.apache.tiles.preparer.ViewPreparer;
import org.apache.tiles.preparer.PreparerException;
import org.apache.tiles.context.TilesRequestContext;
public class HeaderTileController implements ViewPreparer {
    public void execute(TilesRequestContext tilesContext, AttributeContext attributeContext) throws PreparerException {
        String date = new SimpleDateFormat("yyyy/MM/dd").format(new Date());
        int rantsToday = rantService.getRantsForDay(date).size();
        attributeContext.putAttribute("rantsToday", new Attribute(rantsToday));
    }

    private RantService rantService;
    public void setRantService(RantService rantService) {
        this.rantService = rantService;
    }
}
```
This controller uses rantService to calculate how many rants has been made to the RoadRantz application today. There is one unanswered question here that has not been resolved in this thesis due to time deficiency, namely the precise use of the tilesContext variable. But as the name suggest Tiles has its own context which might be useful in some occasions.

Now we need to add the getRantsForDay() function to our rantService implementation:

```
RantServiceImpl

public List<Rant> getRantsForDay(String date) {
    return rantDao.getRantsForDay(date);
}
```

RantService interface

```
public List<Rant> getRantsForDay(String date);
```

A tiles controller returns an object of type Attribute. This Attribute object is made available to the jsp that the controller is preparing. Now all that is left is to create the missing jsp:s:

```
header.jsp
<%@ taglib prefix="spring" uri="http://www.springsource.org/tags%>
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<style>
   .text { color: gray; }
</style>
<title>Insert title here</title>
</head>
<body>
   <h6 class="text">
      <spring:message code="header.before" />
      <tiles:getAsString name="rantsToday" />
      <spring:message code="header.after" />
   </h6>
</body>
</html>
```

```
footer.jsp
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<style>
   .text { color: gray; }
</style>
```
Add the following lines to the .properties files in the classes folder:

messages.properties

header.before=Helping
header.after=Motorists deal with road rage today!

footer.text=This is the footer.<br />The header and the "all rants" list is cached.<br />Caching will flush when adding a rant.<br />Add rant is an all-or-nothing transaction.

The text in the footer is of course not true at the moment, but it gives a sneak peek of what goodies are to come...

The name “rantsToday” in the tiles specific getAsString tag corresponds exactly to the “rantsToday” in the Tiles controller:

attributeContext.putAttribute("rantsToday", new Attribute(rantsToday));

We will no longer have to define the titles in the jsp:s as the titles are defined in the roadrantz-tiles.xml file.

The headerTilesController bean declaration were one of the last beans that we will declare in roadrantz-servlet.xml except for some more simple controllers. Placing controllers in a separate xml file might seem like a good idea, and it would be, if it weren't for the fact that they must be placed in the Dispatcher Servlets own xml (for some reason I've not been able to figure out). From now on we will start filling up those other ominous xml:s.

### 4.5 Back End

There you have it. All the technology to implement a fully functioning Front End (Except perhaps for some cosmetic work) In this chapter we will cover everything that goes on in the data handling part (database and DAOS) of the application. Sadly the RoadRantz application is an island of its own. What I mean by that is that we will not build any Web Services or take advantage of other Web Services. One way of using Web Services could be to enable RoadRantz to connect to a third party service supplier of, lets say, weather information. This would enable the application to display the weather at the location of the event that the user is ranting about. Or maybe connect to the road department for some tasty data that could enhance the experience for the user when using RoadRantz. It would certainly be fun to implement such functionality but in order to not feel to bad about leaving out such an important aspect of web page programming, I have made a list of Spring Web Services support functionality. That list is displayed in the Excluded functionality section of chapter 1.
Having said that, what we are now going to build is a robust data retrieval and storage layer for the RoadRantz application. This means that we are going to design our database, build a DAO layer, enable caching, and assure that everything goes smoothly with secure Transactions.

### 4.5.1 Database management

In this chapter we will build and handle everything that has to do with the database and database handling. To honor the occasion I have created a picture of the flow architecture as designed in the RoadRantz application.

![Flow Architecture](image)

**Fig2: The flow architecture as designed in the RoadRantz application.**

This design mirrors the design of Craig Walls application in the Spring in Action, second edition book.

As we know, a domain object can carry information from the View layer through the Controller layer to the Service layer and back again. Now we will build the architecture for the domain object to be carried all the way to the database and back to the Service layer. For the database handling part of this architecture we are going to use the Object Relational Mapping (ORM) framework Hibernate.

#### 4.5.1.1 Designing the database

Now we are going to continue the discussion we had in the Domain Model chapter. As we discussed, the database will mirror the Domain Model. So how is our Domain Model designed? We have A Rant that among other information contains a Vehicle. But we also know that a Vehicle can have many rants. Hence we have a Many-To-One relationship:
After you have created these tables and set up their relationships in the database you will also need to create a sequence in the database with the name of “hibernate_sequence”. Hibernate will search for this sequence in the database for RoadRantz’s automatic generation of primary keys.

### 4.5.1.2 Data Access Layer

In the introduction I explained that: “I will not discuss the reason why the design patterns (program flows) are as they are.” I will start this chapter by breaking this promise about design patterns with some of my own reasoning about the DAL and the Service layer architecture. This part does not include any practical information so if you are not interested these things you can skip ahead.

One question you might ask yourself is: why bother dividing the Service layer with a Data Access Layer? Why not combine service objects with DAOs? The answer to this question is not directly researched in this theses but practical experience suggests that even in a simple application such as RoadRantz, this division has its uses. I have three examples to motivate this:

1. First and foremost, situations where several service functions might need to use the same database functionality is bound to arise in projects larger than the RoadRanz application. In these cases DAOs provide functionality that the service functions can share.

2. Second and less obvious is that implementing caching is by far most practical in a DAO environment since each DAO function is more atomic in its handling of the database. This atomicity helps ensure that nothing that is not supposed to be cached is cached. It also helps determining what functions that are most suitable to be assigned to task of clearing the cache.

3. Contrary to caching, transaction implementation is more practical in a service environment since each service function might include several DAO calls to perform some logic. A transaction is not complete unless all parts of the function is successfully completed. If some call to the database fails or is compromised in some manner, the transaction is rolled back.

### 4.5.1.3 Building DAOs

Now we will make use of the configurations that we did in the Getting started chapter where we enabled Tomcat to supply a connection to a database. If you are not running Tomcat then you still have to configure a database connection.

The database connection is called a data source. This is because it is of the type `javax.sql.DataSource` that we defined in the web.xml in the Getting started chapter. There are several ways to configure a data source in Spring but we are going to
use JNDI look-up from the server. Luckily Spring makes this real easy for us. First we are going to extend our roadrantz-data.xml file with a new namespace:

```
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.0.xsd">

<!--put configuration here-->
</beans>
```

The code needed for the new namespace is marked in bold. With the jee namespace configured, all we need to do to retrieve our data source from JNDI is adding one single line:

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/postgres" resource-ref="true" />
```

Next we need a session factory bean. A session factory is a bean that produces a Hibernate session factory. This may sound confusing so I will put it in other terms. To use Hibernate we need a Hibernate session factory. Spring handles all configuration and instantiation of that factory with its own session factory. Furthermore we will use an annotated version of Spring's session factory. This means that we will soon get our first glimpse of how to use Java 5 annotations (except for @SuppressWarnings()).

```
<bean id="sessionFactory" class= "org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="annotatedClasses">
    <list>
      <value>com.roadrantz.domain.Rant</value>
      <value>com.roadrantz.domain.Vehicle</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">${hibernate.dialect}</prop>
    </props>
  </property>
</bean>
```

To start with we have wired our dataSource bean to the dataSource property in the AnnotationSessionFactoryBean. Then we have specified which classes that the
sessionFactory bean is going to look in for annotations. In our case those classes are our Domain model Rant and Vehicle.

The last thing that our session factory bean needs is an hibernate dialect. For this I have opted to use a PropertyPlaceholderConfigurer bean to extract the actual value to an .properties file that we will put among our other properties files in the WEB-INF/classes folder. This has nothing to do with configuring our DAOs so if you think this is a waste of time then you can replace the \${hibernate.dialect} variable with org.hibernate.dialect.PostgreSQLDialect directly. Here is the declaration of the PropertyPlaceholderConfigurer bean:

```
roadrantz-data.xml

<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="locations">
    <list>
      <value>classpath:hibernate.properties</value>
    </list>
  </property>
</bean>
```

WEB-INF/classes/hibernate.properties

hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

OK, so we have our session factory nice and ready. So how do we use it? It is time to declare our first DAO in the Spring context:

```
roadrantz-data.xml

<bean id="rantDao" class="com.roadrantz.dao.hibernate.HibernateRantDao">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Since we are using Hibernate as our persistence framework we will name our RantDao implementation HibernateRantDao. We will also put it in a subpackage of com.roadrantz.dao called hibernate. This is because if someday we decide to change our DAO layer to use another persistence framework then we have a clear division of our classes.

We will not use the sessionFactory directly in our code. Rather we will extend a Spring class named HibernateDaoSupport that needs the session factory to take care of the magic in the background for us:

```
package com.roadrantz.dao.hibernate;

import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean;
import com.roadrantz.dao.RantDao;
import com.roadrantz.domain.Rant;
import com.roadrantz.domain.Vehicle;
```
public class HibernateRantDao extends HibernateDaoSupport implements RantDao{
    public HibernateRantDao(){}
    public List<Rant> getAllRants() {
        return getHibernateTemplate().loadAll(Rant.class);
    }

    @SuppressWarnings("unchecked")
    public Vehicle findVehicleByPlate(String plateNumber) {
        List<Vehicle> results = getHibernateTemplate().find("from Vehicle vehicle where vehicle.plateNumber = ?", new Object[]{plateNumber});
        return results.size() > 0 ? (Vehicle) results.get(0) : null;
    }

    public void saveRant(Rant rant) {
        getHibernateTemplate().saveOrUpdate(rant);
    }

    @SuppressWarnings("unchecked")
    public List<Rant> getRantsForDay(String date){
        List<Rant> results = getHibernateTemplate().find("from Rant rant where rant.rantDate = ?", new Object[]{date});
        return results;
    }

    @SuppressWarnings("unused")
    private AnnotationSessionFactoryBean sessionFactory;
    public void setSessionFactory(AnnotationSessionFactoryBean sessionFactory){
        this.sessionFactory = sessionFactory;
    }
}

Hmm... Now lets see here. Have you ever written JDBC before? Then you would know that querying the database takes up a lot of boiler plate code. In fact in JDBC about 80% of the code is plumbing code and about 20% does any real work (if you are not working with really huge SQL queries). Here all we have to do to save a rant to the database is typing:

getHibernateTemplate().saveOrUpdate(rant);

We do not have to worry about creating connections and cleaning up after ourselves. Spring does that for us.

So where did the Vehicle object in the Rant object go? Where does that get saved to the database? Well, Hibernate makes sure that everything in the rant object gets saved to the database, even if it involves saving different parts of the object in different tables.

One last thing you may have noticed is that the SQL in the findVehicleByPlate() and getRantsForDay() methods does not look like real SQL. That is because its not. It is actually Hibernate Query Language (HQL). An object oriented database querying language used by Hibernate.

Now the only thing left to do is to create our RantDao interface:

RantDao interface
When dealing with Hibernate I strongly recommend buying a book. My recommendation goes to “Hibernate Made Easy: Simplified Data Persistence with Hibernate and JPA”. What I found is that using examples off the internet is far from an optimal solution when working with Hibernate. If you are serious about your application, don't do like I did. Buy a book.

In this part of the chapter we will annotate the Domain for persistence with the database. For this we will use JPA annotations that will be read and used by Hibernate. Hold on to your hat because the Domain model is about to get a whole lot more interesting:

```
package com.roadrantz.domain;

import java.io.Serializable;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "rant")
public class Rant implements Serializable{

    private static final long serialVersionUID = -6334115270005501771L;

    private Integer rantId;
    private String rantText;
    private String rantDate;
    private Vehicle vehicle;

    public Rant() {} 

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "rant_id", unique = true, nullable = false)
    public Integer getRantId(){return rantId;}
    public void setRantId(Integer rantId){this.rantId = rantId;}

    @Column(name = "ranttext")
    public String getRantText(){return rantText;}
    public void setRantText(String rantText){this.rantText = rantText;}
```
@Column(name = "date")
public String getRantDate(){return rantDate;}
public void setRantDate(String rantDate){this.rantDate = rantDate;}

@ManyToOne(cascade = {CascadeType.ALL}, fetch = FetchType.EAGER)
@JoinColumn(name = "vehicle_id", nullable = false)
public Vehicle getVehicle(){return vehicle;}
public void setVehicle(Vehicle vehicle){this.vehicle = vehicle;}

Vehicle
package com.roadrantz.domain;
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
@Entity
@Table(name = "vehicle", uniqueConstraints = {
    @UniqueConstraint(columnNames = "platenumber")})
public class Vehicle implements Serializable{
    private static final long serialVersionUID = -9139622073433542787L;
    private Integer id;
    private String plateNumber;
    private String state;
    private Set<Rant> rants = new HashSet<Rant>(0);
    public Vehicle() {};
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "vehicle_id", unique = true, nullable = false)
    public Integer getId(){return id;}
    public void setId(Integer id){this.id = id;}
    @Column(name = "platenumber", unique = true)
    public String getPlateNumber(){return plateNumber;}
    public void setPlateNumber(String plateNumber){
        this.plateNumber = plateNumber;
    }
    @Column(name = "state")
    public String getState(){return state;}
    public void setState(String state){this.state = state;}
    @OneToMany(cascade = {CascadeType.ALL},
            fetch = FetchType.LAZY, mappedBy = "vehicle")
    public Set<Rant> getRants(){return rants;}
    public void setRants(Set<Rant> rants){this.rants = rants;}
}

This code was probably the hardest code to write in this thesis. It might be that I am not as clever as many other people out there but in my experience, the majority of examples off the internet did not work for squat. Neither did the book
I bought online make it in time to my apartment. But to make my blood sweat and tears all worth it I will now explain most aspects of the code so that you will not have to go though the same experience.

I want to start off by saying that extending Serializable is not a requirement for the application to work. I found this text online[1]:

"The class should implement Serializable. Strictly speaking, this is not a requirement. However, in practice you will normally want your Hibernate objects to be serializable so that they can be (potentially) migrated around a multiprocessor cluster or saved and restored across a web server reboot etc."

In hindsight most of the annotations seem very natural and easy:

- @Entity specifies that the class is an Hibernate entity.
- @Table specifies the name of the table for this entity and also uniqueConstraints if any.
- @GeneratedValue() generates a unique value out of a sequence. This sequence is the one and the same as the “hibernate_sequence” that you created in the database.
- @Id annotation tells Hibernate that the variable is a primary key.
- @Column() specifies the name of the column in the database and can also specify constraints on the column/field.
- @OneToMany() specifies a one-to-many relationship with the field specified by the “mappedBy” property. FetchType.LAZY means that when the object is loaded from the database, it will not load the rants that might be associated with it. CascadeType.ALL tells Hibernate to cascade any operation (delete, update etc.) to all referenced objects.
- @JoinColumn specifies which column any @ManyToOne() relationship relates to.

The FetchType.LAZY or FetchType.EAGER properties are the most important things to think about in the Domain. They decide whether or not all referenced objects will be loaded along with the fetched object. The reason why I chose to specify FetchType.LAZY for Vehicle and FetchType.EAGER for Rant is that when we use findVehicleByPlate() function, we will not load all the rants referenced by the vehicle, because we don't need them. On the other hand when we want a Rant, we will always want a Vehicle to accompany it. However if we later on decide that we need all rants for one or more vehicles, then its possible to override the FetchType using HQL (or change the FetchType property).

For your convenience I will now supply a piece of code that is not present in the RoadRanz.war code base:

```java
print all rants about a vehicle

Vehicle existingVehicle = rantService.getVehicleByPlateNumber("a plate number");
System.out.println("HashSet contains : ");
```
```java
for (Rant rant : existingVehicle.getRants())
    System.out.println(rant.getRantText());
```

Set the `FetchType` to `FetchType.EAGER` in Vehicle and try out this code in a controller that uses rantService and implement a function in rantService that gets a single vehicle with attached rants from the DAO layer and see that it works. If the `FetchType` is set to `FetchType.LAZY` then printing this will give you an exception.

Now you might wonder why you shouldn't put the print code directly in a DAO or the rantService. The answer to this question is Transactions. Later on our rantService and consequently our DAOs will be wrapped with Transactions.

**IMPORTANT INFORMATION!:** If we use an object, lets say a Vehicle with `FetchType.LAZY` settings, then it is still possible to fetch its referenced objects, in this case Rants, **while still inside the transaction.** This means that you are never safe to assume that your `FetchType` settings will be enforced until you are outside the transaction.


### 4.5.2 Caching

The theory of caching is very simple: if some value is fetched from the database and its not likely it will change in the database, then its better to save it and reuse it from memory until there is reason to believe that it needs to be updated from the database again. This is what we are going to do in our RoadRantz application. For this we are going to use a third party framework called Ehcache. The first thing we will do is define a namespace for our roadrantz-cache.xml file:

```
roadrantz-cache.xml
```<?xml version="1.0" encoding="UTF-8"?><beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ehcache="http://ehcache-spring-annotations.googlecode.com/svn/schema/ehcache-spring"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                      http://ehcache-spring-annotations.googlecode.com/svn/schema/ehcache-spring
                      http://ehcache-spring-annotations.googlecode.com/svn/schema/ehcache-spring/ehcache-spring-1.1.xsd">
    <!--put configuration here-->
</beans>
```

As you can see the namespace is provided by a fourth party named googlecode. Lucky for us googlecode is dedicated to joining the wonderful worlds of Ehcache and Spring. The following bean declarations enables us to use annotated caching in RoadRantz:

```
roadrantz-cache.xml
```<ehcache:annotation-driven />

40
With the `<ehcache:annotation-driven />` tag in place, Spring will automatically search and identify any functions annotated for caching.

As you can see there is a reference to a new xml file in the `<EhCacheManagerFactoryBean />` declaration. This is the configuration file that will configure the behaviour of Ehcache. This file is not managed by Spring but naming it with the `roadrantz-` prefix and putting it in the `/WEB-INF/` folder felt appropriate:

```
roadrantz-ehcache.xml
```

This configuration defines two caches. The first one is the default cache and is mandatory. The default cache is the cache that will be used if no specific cache is specified. The other cache `rantzCache` is exactly the same but I put it there to show you how to use several different caches.

Now its time to start thinking about how we are going to cache the application. So what do we know? We know that each time a user visits `/home.htm` all rants in the database is displayed (I am well aware that that might not be a very logical functionality). We also know that the only way to change any rant or the number of rants in the database is to add a new rant. This presents us with the perfect opportunity for caching because we know that a value is often shown, and we know exactly when that value will change. And lets not forget that we have a header that is querying the database at every single request the user makes whether the user is logged in or not. Lets cache our DAO:
HibernateRantDao

```java
package com.roadrantz.dao.hibernate;

import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean;
import com.googlecode.ehcache.annotations.Cacheable;
import com.googlecode.ehcache.annotations.TriggersRemove;
import com.roadrantz.dao.RantDao;
import com.roadrantz.domain.Rant;
import com.roadrantz.domain.Vehicle;

public class HibernateRantDao extends HibernateDaoSupport implements RantDao{
    public HibernateRantDao(){
    }

    @Cacheable(cacheName="rantzCache")
    public List<Rant> getAllRants() {
        return getHibernateTemplate().loadAll(Rant.class);
    }

    @SuppressWarnings("unchecked")
    @Cacheable(cacheName="rantzCache")
    @TriggersRemove(cacheName="rantzCache", removeAll=true)
    public void saveRant(Rant rant) {
        getHibernateTemplate().saveOrUpdate(rant);
    }

    @SuppressWarnings("unchecked")
    @Cacheable(cacheName="rantzCache")
    public List<Rant> getRantsForDay(String date){
        List<Rant> results = getHibernateTemplate().find("from Rant rant where rant.rantDate = ?", new Object[] {date});
        return results.size() > 0 ? (Rant) results.get(0) : null;
    }

    @SuppressWarnings("unused")
    private AnnotationSessionFactoryBean sessionFactory;
    public void setSessionFactory(AnnotationSessionFactoryBean sessionFactory){
        this.sessionFactory = sessionFactory;
    }
}
```

Easy as pie. The @TriggersRemove annotations will clear all data stored in the rantzCache, and @Cacheable will tell Spring to cache the returned value(s) from the function. Technically caching works in this way: all parameters to a cached function are mapped to the return value. If the same parameters come in again to the cached function, the function will not be executed at all. Rather the mapped return values of the input parameters will be returned. Its worth mentioning that only external calls to a cached method is cached. I.e. if an internal call within the class calls a cached method, the cache is not activated.
Of course this is not all of the functionality that Ehcache offers. For example there are more types of configuration properties possible in the configuration xml file and in the @Cacheable and @TriggersRemove annotations. One such functionality for the annotations is the ability to only remove one single value from the cache with the @TriggersRemove annotation. For this we would need to specify a key generator. The following example from http://code.google.com is not supplied in the RoadRantz.war file. It defines a cache that removes a single “weather” value from the cache. It is defined in an interface:

```java
@Cacheable(cacheName="weatherCache",
    keyGenerator = @KeyGenerator {
        name = "HashCodeCacheKeyGenerator",
        properties = @Property(name="includeMethod", value="false")
    })
)

public Weather getWeather(String zipCode);

@TriggersRemove(cacheName="weatherCache",
    keyGenerator = @KeyGenerator {
        name = "HashCodeCacheKeyGenerator",
        properties = @Property(name="includeMethod", value="false")
    })
)

public Weather deleteWeather(String zipCode);
```

4.5.3 Transactions

The final part of the back end chapter is here. And maybe the most important. If you have read this document from start to end then you might know that I have discussed Transactions a few times in previous chapters. Otherwise the end of “Annotating the Domain for Hibernate” and part “3” of the “Data Access Layer” chapter contains some useful information about Transactions.

This chapter will make sure that you can sleep well at night, not thinking about if a user manages to fetch a vehicle from the database while adding a rant just milliseconds before another user adds the first rant about the very same vehicle. Nightmare oh nightmare. The user will get a nasty

```java
org.springframework.dao.DataIntegrityViolationException with a nested
org.hibernate.exception.ConstraintViolationException and will be very sad indeed.
```

Luckily for us we can make sure that that will never happen our innocent unsuspecting users. Lets declare some beans:

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation=""
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">
```

43
The transactionManager takes the sessionFactory that we configured for our database connection in the roadrantz-data.xml as a property. This chapter will turn out be be short:

RantServiceImpl

import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;

@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public class RantServiceImpl implements RantService{
    ...
    @Transactional(propagation=Propagation.REQUIRED, readOnly=false)
    public void addRant(Rant rant) {
        Vehicle rantVehicle = rant.getVehicle();
        Vehicle existingVehicle = rantDao.findVehicleByPlate(rantVehicle.getPlateNumber());
        if(existingVehicle != null)
            rant.setVehicle(existingVehicle);
        Set<Rant> rants = new HashSet<Rant>();
        rants.add(rant);
        rant.getVehicle().setRants(rants);
        rant.setRantDate(new SimpleDateFormat("yyyy/MM/dd").format(new Date()));
        rantDao.saveRant(rant);
    }
    ...
}

That's it! In truth there are a lot of factors to take into consideration when declaring transactions but I will strip those away in benefit of convention over configuration.

Four lines were added to RantServiceImpl. Two imports and two annotations. The entire class is annotated with the Propagation.SUPPORTS attribute. This tells Spring (who in turn tells Hibernate) that all the methods in the class don't have to start a new transaction to operate, and if one transaction is already operating then they can safely operate inside that one. This is because all functions except the addRant() function are atomic. Nothing can really go wrong in them. However addRant() reads from the database and uses that information to make a decision about an object that is then inserted into the database. This requires addRant() to
be in a transaction, hence the function is overridden with \texttt{Propagation.REQUIRED}. The \texttt{readOnly=true/false} is an optimization that allows Hibernate to apply optimizations as it sees fit. Since \texttt{readOnly=true} only optimizes at the start of new transactions the setting doesn't really do anything with propagation settings that aren't able to start new transactions (for example \texttt{Propagation.SUPPORTS}), but it can't hurt to put the setting there in case of future changes.

### 4.6 Security

We are finally here. The big elephant in the room. How are we supposed to tackle this problem and feel confident that we have done a good job? Experience shows that often we can't. But to make you feel a little easier about the subject and the framework we are going to use I can mention that Spring security is used by government agencies, military applications and central banks [1]. There is also an entire book dedicated to only Spring security. It is a huge topic. This security solution is based on much research but there were two websites that got me over the threshold [2][3]. The other main source of information than these two sites were the Spring security API docs [4]. One good tip when searching in the API is to Google the class you are interested in and just add a 3.0 at the end. The 3.0.3 version will always be one of the top search results.

NOTE: in this chapter I will try out something a bit different. I will depict the route of a request that gets handled by Spring security in a story like fashion. Because this might be a bad idea I will write the story in \textit{italic} format so that if you do not like it, you can skip over it.

ABOUT REFERENCES: All of the information in this chapter is taken from the Spring security API docs unless otherwise stated.

Before we start, here is a little tip. If you enable slf4j-log4j12-1.6.1.jar instead of slf4j-simple-1.6.1.jar in the WEB-INF/lib (by removing and adding an underscore to the suffix of the files respectively), you will be able to see in much greater detail how Spring security handles HTTP requests.


### 4.6.1 Evil requests

We will start by thinking about how we want a request enter our application. All possible URL combinations from any possible angle should always get intercepted by our security layer. To achieve this we will return to our long lost web.xml to define a filter:

```xml
<filter>
  <filter-name>roadRantzFilterChainProxy</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
  <filter-name>roadRantzFilterChainProxy</filter-name>
</filter-mapping>
```
This regular JEE configured filter is implemented by Spring's DelegatingFilterProxy. This filter proxy will stop any request heading for our application and reroute it into the Spring context.

Now imagine that you are a request with malicious intents flying freely out in the ether towards an URL (our application). Suddenly something grabs a hold of you and drags you backwards into a dark cave of scrutiny and interrogation. Spring’s got you and now there is nowhere out.

More technically speaking the characters "/*" in the <url-pattern> tag of the <filter-mapping> tag makes sure that all requests directed at our server will be intercepted by our DelegatingFilterProxy. We could narrow down the amount of requests handled by our security layer by changing the pattern to lets say "/RoadRantz/*" or something even more specialized, but for now we want our security layer to cover all requests directed at the server.

Now we will start cutting out our dark cave of interrogation.

### 4.6.2 Spring security

We start by declaring the namespace of the xml:

roadrantz-security.xml

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sec="http://www.springframework.org/schema/security"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                          http://www.springframework.org/schema/security
                          http://www.springframework.org/schema/security/spring-security-3.0.xsd"/>
```

The first thing we will put in place is something that can take over from the DelegatingFilterProxy, something that is just as strong and reliable. Say hello to the FilterChainProxy:

roadrantz-security.xml

```xml
<bean id="roadRantzFilterChainProxy" class="org.springframework.security.web.FilterChainProxy">
<sec:filter-chain-map path-type="ant">
   <sec:filter-chain pattern="/**" filters="channelProcessingFilter,
                      securityContextFilter,
                      logoutFilter,
                      formLoginFilter,
                      anonymousAuthenticationFilter,
                      ..."/>
</sec:filter-chain-map>
</bean>
```
The DelegatingFilterProxy filter and the FilterChainProxy need to have the same name and id respectively. In our case the name and id connecting these two beans is “roadRantzFilterChainProxy”. Through the remainder of our visit in the security layer we will be put though a series of interrogation filters (rooms) by the FilterChainProxy. The filters= property of the <sec:filter-chain> tag defines this series of filters.

IMPORTANT: All the filters in Spring security have a certain order. If you decide to use a certain filter then it must be in the right order in relation with other filters to work.

The first thing that the FilterChainProxy need to perform its duties is a language in which to read all its instructions. In our case that language is ”ant” (read about ant in the Market study document). The FilterChainProxy gets a chance to decide for itself which URLs that it should lead through the cave in the sec:filter-chain pattern property. Having it set to “/**” ensures that it will not have any discriminating behaviors about whom to let into the security layer. Everyone is equally welcome. Lucky us.

*As we are pushed inside the first room by the FilterChainProxy we discover a small rusty sign saying “channelProcessingFilter”. Where are they taking us?*

**4.6.3 ChannelProcessingFilter**

*It turns out that our captors are a bit paranoid. Putting us through a cave of horrible experiments doesn't seem to be enough. They need to know that no-one is listening in on the conversations that we are about to have in the cave. And also if they eventually trust us and lets us out of the cave into the application, they need to know that we can not be spied on by any third party. In the ChannelProcessingFilter room they will make sure that we always encrypt what we are saying and that we always have the means of understanding the encrypted answers given to us by enforcing on us the HTTPS protocol. Horror of horrors! Oh wait, that does not hurt at all. We quietly go along with their request:*

```
roadrantz-security.xml
<bean id="channelProcessingFilter" class="org.springframework.security.web.access.channel.ChannelProcessingFilter">
    <!-- Property "channelDecisionManager" is defined in the ChannelDecisionManager.xml file.
    It is the same bean as the "channelDecisionManager" property defined in the SecurityMetaDataSourceBean.
    The ChannelDecisionManagerImpl is used to implement the channelDecisionManager property.
    -->
    <property name="channelDecisionManager" ref="channelDecisionManager"/>
    <property name="securityMetadataSource">
        <sec:filter-invocation-definition-source>
            <sec:intercept-url pattern="/**" access="REQUIRES_SECURE_CHANNEL"/>
        </sec:filter-invocation-definition-source>
    </property>
</bean>
```

```
roadrantz-security.xml
<bean id="channelDecisionManager" class="org.springframework.security.web.access.channel.ChannelDecisionManagerImpl">
    <property name="channelProcessors">
        <list>
            <!-- The list of channelProcessors is defined in the ChannelProcessorsBean file. -->
            <bean/>
        </list>
    </property>
</bean>
```
The \textit{channelProcessingFilter} uses a \textit{securityMetadataSource} to help determine which URLs that requires a secure channel and which ones does not. In our case the \textit{securityMetadataSource} is configured to enforce \texttt{REQUIRES_SECURE_CHANNEL} on all URLs, \\”/\*\*”\“. Thus to enter the application, it is mandatory to use HTTPS.

\textbf{IMPORTANT:} If you by some reason which to build an application that mixes secure and insecure channels, i.e. HTTP and HTTPS then you must remember that any session started in an insecure environment will be invalidated and discarded when entering a secure environment by Tomcat. You can read more on this topic here \cite{1}.

Next the \textit{channelProcessingFilter} uses a \textit{channelDecisionManager} to help enforce which ever channel that the \textit{securityMetadataSource} property specifies.

Technically speaking what has happened here is that if we tried to connect to the server though HTTP, then Spring will refuse the connection and instead insist that the connection is remade using HTTPS. This change is done automatically for the user.

\textit{Now that we have learned to whisper everything we say so that none other than the intended recipient can hear, we are dragged though a long hall by our leash holder, the FilterChainProxy to the next room which happens to be called SecurityContextPersistenceFilter.} \cite{2} \cite{3}  

\begin{itemize}
\item \cite{1} http://forum.springsource.org/archive/index.php/t-65651.html 21/10-2010
\item \cite{2} http://static.springsource.org/spring-security/site/docs/3.0.x/apidocs/org/springframework/security/web/access/channel/ChannelProcessingFilter.html 21/10-2010
\item \cite{3} http://static.springsource.org/spring-security/site/docs/3.0.x/apidocs/org/springframework/security/web/access/channel/ChannelDecisionManagerImpl.html 21/10-2010
\end{itemize}

\subsection*{4.6.3.1 Configuring Tomcat HTTPS with a keystore}

If you use Tomcat and need to use HTTPS, then Put the following inside the \texttt{<Service name=\”Catalina\”>} tag of your server.xml file:

\begin{verbatim}
<Connector port=\"8443\" protocol=\"HTTP/1.1\" SSLEnabled=\"true\"
maxThreads=\"150\"
scheme=\"https\"
secure=\"true\"
clientAuth=\"false\"
sslProtocol=\"TLS\"
keyStoreFile=\"conf/keystore\" />
\end{verbatim}

48
You also need to generate a keystore and put it in the /conf directory in the Tomcat configuration of your Eclipse project. Follow this guide to create a keystore:


The following is how you find where to put the keystore (depending on your placement of the Eclipse workspace):

My Documents\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\conf

The number in the “tmp0” part of the path reflects the number of your project. The first project you create in Eclipse gets the number 0, the next get number 1 and so on.

4.6.4 SecurityContextPersistanceFilter

The bean definition of the SecurityContextPersistenceFilter is simple:

roadrantz-security.xml

```xml
<bean id="securityContextFilter" class="org.springframework.security.web.context.SecurityContextPersistenceFilter" >
    <property name="securityContextRepository" ref="securityContextRepository" />
</bean>

<bean id="securityContextRepository" class="org.springframework.security.web.context.HttpSessionSecurityContextRepository" />
```

The securityContextRepository is a security bean that will be used by more beans than one in the Spring security context. Therefore it is recommended that you put it separate from all the “filter” beans in the roadrantz-security.xml.

The SecurityContextPersistenceFilter is responsible for storing the security context between requests and making that context available during requests. This means that once a user is authenticated by Spring security, the authentication information will be fetched in this filter and the user will not have to re-authenticate with each request. Behind the scenes a security context is a SecurityContext bean that is stored in a SecurityContextHolder while performing a request. Between requests the SecurityContext is stored in a SecurityContextRepository. In turn a SecurityContext stores an Authentication. The Authentication bean stores all the authentication information and credentials for a user. We will get back to the Authentication bean in later filters.

IMPORTANT: You should never use the HttpSession in the application. If you want to use a HttpSession you should use the SecurityContextHolder instead which you can use DI to inject into which ever class that needs it.

[1][2]
We have now been assigned a “backpack” containing all the credentials we need to authenticate ourselves. But we have not actually authenticated ourselves yet so our backpack is empty. However if this was not the first time that we visited the site, then this backpack of credentials would make sure that Spring knows exactly who we are and what authorizations we have (as we will see, credentials are not given out easily by Spring). All this ruckus simply means that once logged in, the credentials of a user is persisted by Spring in memory/RAM.

Since nothing actually happened to us in this room we are swiftly taken to the next room called LogoutFilter.

2. [http://docs.huihoo.com/spring/spring-security/3.0.x/technical-overview.html#tech-intro-sec-context-persistence](http://docs.huihoo.com/spring/spring-security/3.0.x/technical-overview.html#tech-intro-sec-context-persistence)

4.6.5 LogoutFilter

If we were a request for logging out then this filter would be where Spring security would handle the logout procedure. Lets see how this filter is decorated:

roadrantz-security.xml

```xml
<bean id="logoutFilter" class="org.springframework.security.web.authentication.logout.LogoutFilter">
    <constructor-arg value="/login.htm" />
    <constructor-arg>
        <list>
            <bean class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler" />
        </list>
    </constructor-arg>
</bean>
```

The first thing that might catch your eye is the comment on the top. It says something about a `/j_spring_security_logout`. Now what would that be? Well Spring security's `LogoutFilter` listens for requests with this POST variable in the URL. If it finds this variable then this filter steps in and logs the user out. This is done by the `SecurityContextLogoutHandler` bean that modifies the `SecurityContextHolder` that we spoke about in the last chapter so that the user I logged out. It also invalidates any `HttpSession` that might be active.

Through the `<constructor-arg value="/login.htm" />` tag we can specify where to send the user after the logout is carried out. In our case that would be a page called `login.htm`, one that we haven't created yet.

[1][2]

Since we are not in the businesses of logging ourselves out, (remember that we are a malicious request trying to wreak havoc on the application) we will continue further
down the cave (with the FilterChainProxy still thoroughly holding our leash). Next room seems to be called formLoginFilter.


### 4.6.6 FormLoginFilter

Before we enter the room we can't help but notice strange sounds of machinery and cogs turning in the background, and we see bits and pieces of ones and zeroes scattered all about the floor of the room. We instantly get an eerie feeling in our HTTPS request HEADER. Could this be where requests goes to disassemble? Is this the end of our computation? Lets take a deep Thread.sleep() and take a closer look at the room:

roadrantz-security.xml

```xml
<!--formLoginFilter-->  
<bean id="formLoginFilter" class="org.springframework.security.web.authentication. UsernamePasswordAuthenticationFilter">
  <property name="authenticationManager" ref="authenticationManager" />
  <property name="authenticationSuccessHandler">
    <bean class="org.springframework.security.web.authentication .SavedRequestAwareAuthenticationSuccessHandler">
      <property name="defaultTargetUrl" value="/home.htm" />
    </bean>
  </property>
  <property name="authenticationFailureHandler">
    <bean class="org.springframework.security.web.authentication. SimpleUrlAuthenticationFailureHandler">
      <property name="defaultFailureUrl" value="/authorizationError.htm" />
    </bean>
  </property>
</bean>
</property>
</bean>
</bean>
```

That was that, we are definitely toast. Our days as a malignant request seems to be over. This looks like way to much security for us to be able to talk ourselves out of it. But we'll be damned if we don't try...

To start with, this is a UsernamePasswordAuthenticationFilter. The function of the filter is exactly as the name implies. Similar to the LogoutFilter, this filter also listens for special POST variables. Those variables are j_username and j_password and this filter will read the values assigned to these POST variables and send them to an authenticationManager. How this manager works is a small chapter of its own and I will treat it as such.

Now imagine that the authenticationManager has provided us with either a successful authentication, or a failed authentication. The authenticationSuccessHandler and authenticationFailureHandler will treat each outcome respectively. If the authentication was successful the authenticationManager will return an Authentication bean that will be placed in the SecurityContextHolder described in the SecurityContextPersistanceFilter
chapter. Next the `authenticationSuccessHandler` will be called to redirect the user to the appropriate destination. This `authenticationSuccessHandler` is implemented by the `SavedRequestAwareAuthenticationSuccessHandler` bean. The default destination is set to be “/home.htm” but that might not always happen. If the user has previously requested to enter a site, but was not logged in at the moment, then the user will be automatically redirected to the login page. If the user then submits correct authentication information, he or she will be redirected to the original site they requested.

On the other hand if the user failed to supply valid authentication information at any time, the `authenticationFailureHandler` that is implemented by the `SimpleUrlAuthenticationFailureHandler` bean will be put into action. This bean always redirect the user to the value of the `defaultFailureUrl` property, here specified to be “/authorizationError.htm”.

[1][2][3]


### 4.6.7 AuthenticationManager

The `authenticationManager` is responsible for authorizing requests. It does this by iterating through a list of `AuthenticationProvider`s, asking each to authenticate the request. Lets see what this bean looks like:

```xml
<bean id="authenticationManager" class="org.springframework.security.authentication.ProviderManager">
  <property name="providers">
    <list>
      <ref bean="daoAuthenticationProvider" />
    </list>
  </property>
</bean>
```

When calling an `AuthenticationProvider` one of three things will happen;

- If the authentication is successful the `AuthenticationProvider` will return an `Authentication` object including credentials. This will stop the iteration and the `Authentication` object is returned.

- If the `AuthenticationProvider` is unable to handle the request it will return null and the `authenticationManager` will continue the iteration.
If the AuthenticationProvider can handle the request but the authentication is fails, then it will throw an AuthenticationException and the authenticationManager will continue the iteration.

If all of the AuthenticationProviders is unable to even handle the request, (i.e scenario two above for all AuthenticationProvider) then an ProviderNotFoundException will be thrown by the authenticationManager. If one or more of the AuthenticationProviders where able to handle the request but all authorizations failed, then the latest AuthenticationException thrown by a AuthenticationProvider is used. Otherwise if the authentication was successful the authenticationManager will return the Authentication object to the caller. Any exceptions thrown by the authenticationManager will be caught and handled by the ExceptionTranslationFilter further down in the FilterChainProxy chain.

4.6.8 Authentication Providers

An authentication provider is basically an object that compares user input to stored data to validate a request and return an Authentication object. An Authentication object can have several different implementations depending on which Authentication the user has. In the RoadRantz application the user can have one of two AuthenticationS: anonymous or authenticated. These two Authentication types are represented in Spring security by AnonymousAuthenticationToken and UsernamePasswordAuthenticationToken. However we will not need to implement an anonymousAuthenticationProvider for a user with a AnonymousAuthenticationToken. This is because (and now it will become a bit technical) the filter AnonymousAuthenticationFilter (that we will implement later) that supplies a user with an AnonymousAuthenticationToken and saves it to the SecurityContextHolder will also delete it from the holder after the request is completed so that it can't be saved to the SecurityContextRepository by the SecurityContextPersistenceFilter. This also means that it can never be retrieved by the SecurityContextPersistenceFilter in the beginning of the filter chain and thus a anonymousAuthenticationProvider is not necessary, simply because the user will never have a AnonymousAuthenticationToken at this stage of the authentication process [1].

This piece of information was extremely hard to dig out and puzzle together from the Spring APIs, forums and Jira threads and it might not even interest you. But it felt necessary to have this information to be able to provide a security solution without such logical faults as to define an authenticationProvider that is not really necessary.

Anyhow, Here is the bean definition of the authenticationProvider we do need:

roadrantz-security.xml

<!--authenticationProviders-->  
<bean id="daoAuthenticationProvider" class="org.springframework.security. authentication.dao.DaoAuthenticationProvider">
The “commented out” xml is an alternative implementation of an `userDetailsService`. A `userDetailsService` is a bean that retrieves user details from different sources. The reason why there are two different versions is because it can be useful to be able to easily switch between a `userDetailsService` that fetches information directly from the roadrantz-security.xml file, and one that operates on a database.

The `passwordEncoder` and `saltSource` properties specifies that when we retrieve our password from which ever source we use, we will revert an encryption hash done on it. In our case the encryption is an implementation of the MD5 hash done with a salt. The salt is the username belonging to the password whose hash is being reversed:

```
roadrantz-security.xml
<bean id="passwordEncoder" class="org.springframework.security.authentication.encoding.Md5PasswordEncoder" />
<bean id="saltSource" class="org.springframework.security.authentication.dao.ReflectionSaltSource">
    <property name="userPropertyToUse" value="username" />
</bean>
```

Now we will define our two different `userDetailsService`s.

```
roadrantz-security.xml
<bean id="inMemoryDaoImpl" class="org.springframework.security.core.userdetails.memory.InMemoryDaoImpl">
    <property name="userMap">
        <value>
bob=boll,ROLE_USER,ROLE_AUTHENTICATED
sam=fisk,ROLE_USER
        </value>
    </property>
</bean>
<bean id="jdbcDaoImpl" class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">
    <property name="dataSource" ref="dataSource" />
    <property name="usersByUsernameQuery">
        <value>
            SELECT email as username, password, enabled 
            FROM Users 
            WHERE email=?
        </value>
    </property>
    <property name="authoritiesByUsernameQuery">
        <value>
            SELECT a.email as username, authority 
            FROM Authorities a, Users u 
            WHERE a.email = u.email AND u.email=?
        </value>
    </property>
</bean>
```
Finally we are getting our hands dirty with something recognizable. Our good friend the `dataSource` is even making a reappearance. As you might remember from the “Building DAOs” chapter the `dataSource` provides us with a connection to the database. In this case it will help us fetch user information and any authorities associated with that user. A perfect example of what an authority might look like are the capital letters of the two user accounts specified in our `inMemoryDaoImpl` namely “bob=boll,ROLE_USER,ROLE_AUTHENTICATED” and “sam=fisk,ROLE_USER”. The word before the “=” is the username of the account and the word after is the password. Observe that the passwords are written in plain text. If we would try to use the `inMemoryDaoImpl` without commenting out the `passwordEncoder` and `saltSource` properties of the `daoAuthenticationProvider`, then Spring would assume that these passwords are hashed with an hopeless pile of jibberish as the result.

Now let’s direct our focus to the SQL queries. They seem to operate two tables “users” and “authorities”. How fun! It seems to be time to design some more database tables:

![Fig4: Database relationship diagram](image)

Having these tables defined in the database allows us, or rather Spring, to store and retrieve user information from the database. Now let’s analyze those SQL queries. The `usersByUsernameQuery` query obviously returns users by username. Unless something has gone horribly wrong, there should never be more than one user as a result to this query. The `authoritiesByUsernameQuery` fetches all the authorities of a single user. You might wonder why “username” is substituted for “email”. The thought where to incorporate email into the RoadRantz application but those plans never came to fruition.

That was that. Having configured the `authenticationManager` to use the `daoAuthenticationProvider` to use a `userDetailsService` to encrypt, and compare the submitted information with the retrieved user information from the database, we are now ready to continue our story taking place in the `formLoginFilter`.

[1] https://jira.springsource.org/browse/SEC-1313 21/10-2010

4.6.9 FormLoginFilter continuation...

OK, so where were we? Right, we were about to get terribly scared by all the horrible machinery noise and suspicious remains of HTTPS requests scattered on the floor. We
where just about to make a Thread.sleep() to collect our thoughts when we suddenly remember that we are only a HTTPS request, and we have absolutely no power to execute a Thread.sleep(). We must simply wait and see what happens. Our guard the FilterChainProxy quickly disperses us in to a gruesome looking machine with an old rusty sign saying authenticationManager. As soon as we tumble in we are as quickly spat out again, this time with a stamp on our forehead saying ProviderNotFoundException. What events that just transpired has barely begun to write itself in to our HTTPS HEADER when the FilterChainProxy quickly picks us up and starts to carry us to the next room. Well at least we are still alive and might also still have a fighting chance of doing some damage to the application. This FilterChainProxy doesn’t really look that tough anyway. The next room seems to be called AnonymousAuthenticationFilter. We are carelessly thrown inside...

4.6.10 AnonymousAuthenticationFilter

We pick ourselves up from the floor and brush ourselves off. We realize that we are inside a nice and clean looking room. No HTTPS disassembling seem to have transpired here since there are no traces of any ones and zeroes that are not in their proper eight and eight order. There is a machine in the middle of the room. Lets see what will happen...

If a request ends up at this filter that still has no Authentication in its SecurityContextHolder then this filter will supply it with an AnonymousAuthenticationToken (that is an implementation of an Authentication). Here is the bean definition:

roadrantz-security.xml

<bean id="anonymousAuthenticationFilter"
class="org.springframework.security.web.authentication.AnonymousAuthenticationFilter" >
    <property name="key" value="9753ThisIsMyX-JobbsKey3579" />
    <property name="userAttribute" value="anonomousUser, ROLE_USER" />
</bean>

The <property name="key" value="9753ThisIsMyX-JobbsKey3579" /> tag specifies a global key for all AnonymousAuthenticationTokens. Since we will never use an authenticationProvider to verify an AnonymousAuthenticationToken we will never have any use of this key. However it is required by Spring that it is specified.

The userAttribute property however is a little bit more interesting. This property specifies what credentials an anonymous user should have. The “anonomousUser” credential is mandatory to specify but is not used in RoadRantz. However the “ROLE_USER” credential is. Later we will specify that all pages will at least have a security level that requires that uses have the “ROLE_USER” credential.

The machine starts to purr like a kitten and all of a sudden a small package plops out on a small conveyor belt in the front of the machine. An automatic arm picks up the package and puts it in our empty backpack. The package has a note on it saying AnonymousAuthenticationToken. How nice of the machine to give us presents. The FilterChainProxy, who doesn’t particularly seem to care about our sudden joy, starts leading us to the next room.
Let's start with the bean definition straight off the bat:

```xml
<bean id="sessionManagementFilter" class="org.springframework.security.web.session.SessionManagementFilter">
  <constructor-arg ref="securityContextRepository" />
  <property name="sessionAuthenticationStrategy">
    <bean class="org.springframework.security.web.authentication.session.SessionFixationProtectionStrategy" />
  </property>
</bean>
```

This is the other filter that uses the `securityContextRepository` bean that we talked about in the `SecurityContextPersistenceFilter` chapter. This filter uses the `securityContextRepository` to get the `SecurityContext` of the user.

So what is the `SessionManagementFilter` about? Well if the user is successfully authenticated, the `SessionFixationProtectionStrategy` bean property is Spring security's response to session-fixation attacks. This bean creates a new session for the newly authenticated user if they already have a session, and copies the information from the old session to the new session. This ensures that an authenticated user always has a different session id with each request.

But there is an even more restricting session strategy available, namely `ConcurrentSessionControlStrategy` that extends `SessionFixationProtectionStrategy`. This strategy restricts how many sessions a user can have open at the same time. So when might that be useful? One occasion might be if we want make sure that one user account can only be accessed through a single browser from a single computer at the same time. This is provided that the maximum amount of concurrent sessions are set to one. The implementation of `ConcurrentSessionControlStrategy` is not supported in this solution.

As for our hero, the malicious HTTPS request. It gets a new serial number replacing the old one on its backpack. It is starting to look like the evil request might make it to the end of our dark cave of interrogation with its URL intact (whichever URL that might be). But we still have two rooms left...

---

**4.6.12 ExceptionTranslationFilter**

Do you remember that we received a `ProviderNotFoundException` stamped to our forehead in the `formLoginFilter`? This is where that exception is handled:

```xml
<bean id="exceptionTranslator" class="org.springframework.security.web.access.ExceptionTranslationFilter">
  <property name="authenticationEntryPoint">
    <bean class="org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint">
      <property name="loginFormUrl" value="/login.htm"/>
    </bean>
  </property>
</bean>
```

---
So far in the filter chain there are two types of exceptions possible, `AuthenticationException` and `ProviderNotFoundException`, both of which thrown in the `formLoginFilter`.

The first exception, `AuthenticationException` will trigger the `accessDeniedHandler` property. If the request has an `AuthenticationException` that means that the user has tried to authenticate him or her self but has supplied the wrong username and/or password. In RoadRantz that means that they will be redirected to a `errorPage` with the URL `/authorizationError.htm`.

The second exception, `ProviderNotFoundException` will trigger the `authenticationEntryPoint` property and redirect the URL of the request to `/login.htm`. This means that the user will be redirected to a login page where he or she will be prompted to login with a username and password. Remember that with the `SavedRequestAwareAuthenticationSuccessHandler` configured in the `formLoginFilter`, once the user has authenticated him or her self, then he or she will be redirected to his or hers original URL (as we will see later, even with successful authentication, they might not get to where they want without proper credentials).

Now since our malicious request has an `ProviderNotFoundException`, This means that it's URL will be overwritten with `/login.htm`. But its original URL will also be saved. If the user behind the request manages to provide valid username and password, the old URL will be restored. We now arrive to final room, the `FilterSecurityInterceptor`.

### 4.6.13 FilterSecurityInterceptor

This filter is what (almost) all our efforts so far boils down to. This is where it is decided whether or not you have the right credentials to enter the site that you are requesting. Lets have a look at our last filter:

```xml
roadrantz-security.xml
<!--filterSecurityInterceptor-->  
<bean id="filterSecurityInterceptor" class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
    <property name="securityMetadataSource">
        <sec:filter-security-metadata-source>
            <sec:intercept-url pattern="/login.htm" access="ROLE_USER" />  
            <sec:intercept-url pattern="/register.htm" access="ROLE_USER" />  
            <sec:intercept-url pattern="/authorizationError.htm" access="ROLE_USER" />  
            <sec:intercept-url pattern="/**" access="ROLE_USER,ROLE_AUTHENTICATED" />
        </sec:filter-security-metadata-source>
    </property>
</bean>
```
The most interesting part here is of course the `securityMetadataSource` property. It specifies what credentials are necessary to enter a certain page. This configuration ensures that the top three pages can be accessed only with a “ROLE_USER” credential. All the rest of the pages require both “ROLE_USER” and “ROLE_AUTHENTICATED”. However this can be confusing since the last row in the list essentially says that all pages need the two credentials and someone with only the “ROLE_USER” can't get in anywhere. This is why the order of the `sec:intercept-url` properties is important. It is only the first match to the requested page that is used. The rest of the lines in the `securityMetadataSource` list are discarded once a match is found.

The `FilterSecurityInterceptor` also has an `authenticationManager`. However in our implementation of Spring security this `authenticationManager` is not necessary since all the authentication needed has already been performed in the `formLoginFilter` and the `anonymousAuthenticationFilter`. However Spring demands that an implementation of this property is supplied so we need to supply a special `authenticationManager` that doesn't do anything:

```
roadrantz-security.xml

<bean id="nullAuthenticationManager" class="org.springframework.security.
authentication.ProviderManager"
    >
    <property name="providers">
        <list>
            <bean class="org.springframework.security.config.authentication.
            AuthenticationManagerBeanDefinitionParser.
            NullAuthenticationProvider" />
        </list>
    </property>
</bean>
```

Now it would seem that the rest is simple, if the request has the credentials necessary for a certain URL then access is granted. It can be that easy. However if the needs of the application are a little bit more complex, then we are not confined to such a simple model. This is where the `accessDecisionManager` property comes into the picture.

### 4.6.14 AccessDecisionManagers

Consider if we would have a more complex system of credentials than the one we use in the RoadRantz application. We could have several different prefix types (in RoadRantz we only use “ROLE_”), we could have a lot more authorization levels (in RoadRantz we have “USER” and “AUTHENTICATED”). We might also want to set up different rules of authentication. We might want a super user that trumps all other constellations of credentials. In that case we don't want to supply a long list of other lesser credentials to the super user. We might want to be able to say that a certain amount of credentials is enough, that you do not have to have all the credentials for a specific URL to be allowed.
access. These examples are just a few of the possible configurations possible in Spring security. Let’s emerge ourselves in the wonderful world of decisionManagers and decisionVoters:

roadrantz-security.xml

```xml
<bean id="accessDecisionManager" class="org.springframework.security.access.vote.UnanimousBased">
    <property name="decisionVoters">
        <list>
            <ref bean="roleVoter" />
        </list>
    </property>
</bean>

<bean id="roleVoter" class="org.springframework.security.access.vote.RoleVoter">
    <property name="rolePrefix" value="ROLE_" />
</bean>
```

This looks very simple, and it is. We have a UnanimousBased voting system where all the roleVoters (RoadRantz only use one) must agree on whether or not to grant access to a certain resource (URL). This implementation is the easiest possible. One voter that only handles credentials with the “ROLE_” prefix compares the needed credentials for a certain resource, to the credentials of the request. A roleVoter can vote in three ways:

- If all the needed credentials for the resource has an exact match in both prefix and suffix in the list of credentials for the request, then the RoleVoter will vote "ACCESS_GRANTED". Mathematically: the needed credentials for the recourse is equal to or a subset of the list of credentials for the request.

- If none of the credential prefixes for the resource exists in the list of credentials of the request, then the RoleVoter will vote "ACCESS_ABSTAIN".

- If there exists credentials in the list of credentials for the request that have the same prefix as the prefix that the RoleVoter is set to vote on, but still does not meet the "ACCESS_GRANTED" criteria, then the RoleVoter will vote "ACCESS_DENIED".

In practice the difference between "ACCESS_ABSTAIN" and "ACCESS_DENIED" is whether or not there exists one or more credentials in the list of credentials for the request that shares the prefix of the RoleVoter.

There are many more types of voters but one of the more interesting is the AuthenticatedVoter. This voter votes based on the type of authentication that the request has. This has not yet been discussed but all authentications is Spring security is of a certain type. There are three types, namely "IS_AUTHENTICATED_ANONYMOUSLY”, ”IS_AUTHENTICATED_FULLY” and ”IS_AUTHENTICATED_REMEMBERED”. These three types reflect if a user is anonymous, successfully authenticated through a login form or has been automatically authenticated using a cookie (remembered). Depending on configuration this voter will vote if any of the three login types are present, and abstain if they are not.
Now what happens if the voters can't agree? This is where the accessDecisionManager comes into the picture. There are three types:

- **UnanimousBased.** This implementation requires that all voters abtain or grant access.
- **ConsensusBased.** This implementation means majority-rule (ignoring abstains).
- **AffirmativeBased.** This implementation grants access if any voter returns an affirmative response.

The rest is up to you to experiment and mix to find a suitable voting solution for your application. Now we return to the FilterSecurityInterceptor.

### 4.6.15 FilterSecurityInterceptor Continuation...

Now that our accessDecisionManager has come to a decision, let's say that access was granted, then the request is handled as normal by the application. On the other hand if access was not granted the user will be redirected to the login page.

*This means that all in all our little malicious request can not get anywhere except for the `/login.htm` page, the `/register.htm` page or the `/authorizationError.htm` page without logging in.*

Now we are finished! Well not quite. We still need to implement the actual login and registration functionality in RoadRantz. This is what we will do next.

### 4.6.16 Creating login and registration functionality in RoadRantz

That was that! Spring security working full time to please your needs. Now all that is left is to implement the login and registration parts of the RoadRantz application. The implementation of this functionality will introduce some more Spring functionality. However most of the classes we need does not. So in this chapter I will look closer on some JSP functionality that has become available with Spring security, and also how to implement the hash functions that we use to encrypt the passwords in the database. But first off I will show exactly what is left to implement in the following table:
As usual, all these classes can be found in the RoadRantz.war file. So if you get stuck, feel free to take a look. And don't forget to register these files in the Spring context.

Now lets take a closer look at those hash functions:

```
package com.roadrantz.service;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
import org.springframework.security.authentication.encoding.PasswordEncoder;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import com.roadrantz.dao.AccountDao;
import com.roadrantz.domain.Authority;
import com.roadrantz.domain.User;

@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public class AccountServiceImpl implements AccountService{

    @Transactional(propagation=Propagation.REQUIRED, readOnly=false)
    public void addUser(User user) {
        Set<Authority> authorities = new HashSet<Authority>();
        authorities.add(new Authority(user.getUserName(), "ROLE_USER", user));
        authorities.add(new Authority(user.getUserName(), "ROLE_AUTHENTICATED", user));
        user.setAuthorities(authorities);
        user.setPassword(passwordEncoder.encodePassword(user.getPassword(), user.getUserName()));
        user.setEnabled(true);
        accountDao.addUser(user);
    }

```
public List<User> getUserByUsername(String userName) {
    return accountDao.getUserByUserName(userName);
}

private PasswordEncoder passwordEncoder;
public void setPasswordEncoder(PasswordEncoder passwordEncoder)
    this.passwordEncoder = passwordEncoder;

private AccountDao accountDao;
public void setAccountDao(AccountDao accountDao)
    this.accountDao = accountDao;
}

This service uses the same passwordEncoder that we defined in the Spring security context. Remember that we implemented a “ReflectionSaltSource” that used the username as salt? Well here we will do exactly that. The second parameter of the passwordEncoder.encodePassword function takes the salt with which to perform the hash.

Here is the bean definition. Remember the referenced bean passwordEncoder is already defined in roadrantz-security.xml:

roadrantz-service.xml

<bean name="accountService" class="com.roadrantz.service.AccountServiceImpl">
    <property name="accountDao" ref="accountDao" />
    <property name="passwordEncoder" ref="passwordEncoder" />
</bean>

Next up is some neat new functionality for our jsp pages. Let's take a quick look at:

home.jsp

...<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags"%>
...
<h2>
    <spring:message code="home.welcome" />
    <sec:authorize ifAllGranted="ROLE_USER,ROLE_AUTHENTICATED">
        <sec:authentication property="principal.username" />
    </sec:authorize>
</h2>
...

This little peace of bolded code shows how to get the username of the person that is logged in in the jsp context. The “principal.username” might seem strange. But don't worry. This code will work straight off the bat in your application. The ”principal“ part is actually an instance of the object UserDetails and this code will print the username of the person that is logged in.
4.7 Cross Cutting Concerns with AOP

Aspect Oriented Programming is a huge topic and this solution will only scrape the surface of it. But it might give you a hint about when or where you might want to use AOP in your project.

In this chapter we will use AOP for our Logging needs. However AOP is a tool that should not be confined into predefined boxes. A programmer should, when devising a new system, have AOP in the back of his or hers mind as a tool that can be used to solve many different problems. The “soul” of AOP is the ability to activate code at certain points in the program that the program in question doesn't necessarily need to know about. This helps provide highly modularized and decoupled code.

4.7.1 Logging

DISCLAIMER: The purpose of this chapter is solely to display AOP functionality, not to provide extensive logging.

Is there any reason why a class should need to know if it is being logged if it does not have to? I can't think of any reason. That is why we will now start using Spring to help us build some basic AOP logging for our application.

We start by creating a new xml file with a new namespace:

roadrantz-logging.xml

```xml

<!--put configuration here-->
</beans>
```

Here we have defined the new namespace “aop”. With this namespace in place we can start using aop tags. Now we need to start thinking about what to log. I came to the conclusion that our RantService, which in a way is the central hub of our application, might deserve some more attention. Hence I decided to input logging:

```java
public interface RantService {
    ← here
    public List<Rant> getAllRants();
    ← here
    ← here
    public void addRant(Rant rant);
    ← here
    ← here
    public List<Rant> getRantsForDay(String date);
}
In other words this means that we will log all functions in RantService before and after execution. To do this we need some xml:

roadrantz-logging.xml

```xml
<aop:config>
  <aop:aspect ref="rantServiceLogger">
    <aop:pointcut id="addRantPointcut" expression="execution(* *.addRant(..))"/>
    <aop:pointcut id="getAllRantsPointcut" expression="execution(* *.getAllRants(..))"/>
    <aop:pointcut id="getRantsForDayPointcut" expression="execution(* *.getRantsForDay(..))"/>
    
    <aop:before method="logBeforeAddRant" pointcut-ref="addRantPointcut"/>
    <aop:after method="logAfterAddRant" pointcut-ref="addRantPointcut"/>
    <aop:before method="logBeforeGetAllRants" pointcut-ref="getAllRantsPointcut"/>
    <aop:after method="logAfterGetAllRants" pointcut-ref="getAllRantsPointcut"/>
    <aop:before method="logBeforeGetRantsForDay" pointcut-ref="getRantsForDayPointcut"/>
    <aop:after method="logAfterGetRantsForDay" pointcut-ref="getRantsForDayPointcut"/>
  </aop:aspect>
</aop:config>

The aop:pointcut tag tells Spring which bean, and what method in that bean, that we are going to mark for AOP. In our case we will:
log any return value (first *)
from any class (second *)
and all functions named (.addRant())
with any input parameters (..)

Next the aop:before and aop:after tags define which functions to call before and which functions to call after a call to addRant(), getAllRants() or getRantsForDay().

Finally the <aop:aspect ref="rantServiceLogger"> tag that wraps all AOP point-cuts defines what class to use as our logging class. The bean definition is as follows:

roadrantz-logging.xml

```xml
<bean id="rantServiceLogger" class="com.roadrantz.logging.RantServiceLogger"/>
```

Last but not least is the actual implementation of our logger:
package com.roadrantz.logging;
import org.apache.log4j.Logger;

public class RantServiceLogger {
    private static final Logger LOGGER = Logger.getLogger(RantServiceLogger.class);

    public void logBeforeAddRant() {
        LOGGER.info("Before addRant.");
    }

    public void logAfterAddRant() {
        LOGGER.info("After addRant.");
    }

    public void logBeforeGetAllRants() {
        LOGGER.info("Before getAllRants.");
    }

    public void logAfterGetAllRants() {
        LOGGER.info("After getAllRants.");
    }

    public void logBeforeGetRantsForDay() {
        LOGGER.info("Before getRantsForDay.");
    }

    public void logAfterGetRantsForDay() {
        LOGGER.info("After getRantsForDay.");
    }
}

5 Result

As explained in the “Method description” chapter, the goal of this thesis was to write a tutorial. Whether or not this particular tutorial meets the scientific standards of a thesis work has yet to be seen. But what I personally feel is relevant is whether or not the solution meets the standards of a technical tutorial. Since the only way to know for certain whether or not the solution meets the standards is to let a developer with limited knowledge of JEE technologies take a shot in implementing an application with it. Therefore it is hard at this stage of the thesis work to know the quality of the solution since I have not had the honor of such opportunity. However, great care has been taken to ensure that no questions, technical or otherwise, remain unanswered in the regard of each different functionality presented in the solution. Nevertheless, information gaps almost certainly exist. I do not know where in this particular thesis but it is safe to say that almost all technical tutorials have an Achilles heel. And that heel is presumption of knowledge. Presumption of knowledge can render even the greatest of works useless. If the tutorial presents fantastic ways to do one thing or another, but fails to build up all the steps that are required for a common user to be able to take in the information, then the entire work is next to useless. It is a personal belief of mine that presumption of knowledge is the greater of all evils in the computer science community. With this in mind while writing this solution, it is my hope that as few gaps of information as possible has slipped through into the tutorial. Since I never like to leave a work unfinished, I decided to convert this thesis to an HTML website and advertise it in any way I can. I know that this experiment will not be completely futile because I have a thorough, in depth experience of what material is out there on the web. Compared to the work in this
thesis, much of it does not provide much detailed information. The result of that endeavor has yet to come.

6 Recommendations and conclusions

The solution in this thesis is not complete. Major areas of enterprise web development are missing. Amongst these are Web Services and Application modularization. In short I can draw the following conclusion: The solution in this thesis is perfect for stand alone web applications that do not communicate with other applications and are not of such size that they need to be deployed over several machines and or modularized into several distinct parts. However Spring supports all of the mentioned functionality and there are books that explain how to implement it. Especially “Spring Dynamic Modules in Action” is a very interesting book for further research.

7 References

Author: Craig Walls
Publisher: Manning
ISBN-10: 1933988134

Hibernate made easy – Book 2008
Author: Cameron McKenzie
Publisher: Independent
ISBN-10: 0615201954

Test Driven – Book 2008
Subtitle: TDD and Acceptance TDD for Java Developers
Author: Lasse Koskela
Publisher: Manning
ISBN-10: 1932394850

This chapter contains every single website I visited while programming this solution. More than anything else it can be used to see which areas where a problematic, and which where not. The date 21/10-2010 is the last date when I looked though that all the links worked.

Now I would like to mention that links are required by KTH to be printed in their full form, not by a alias like the ones below. But printing them without their alias would create a huge non readable mass of text, so I think this is a better format:
Spring security API

JdbcDaoImpl (Spring Security 3.0.3.RELEASE API)  21/10-2010
User (Spring Security 3.0.3.RELEASE API)  21/10-2010
ReflectionSaltSource (Spring Security 3.0.3.RELEASE API)  21/10-2010
SystemWideSaltSource (Spring Security 3.0.3.RELEASE API)  21/10-2010
SaltSource (Spring Security 3.0.3.RELEASE API)  21/10-2010
TokenBasedRememberMeServices (Spring Security 3.0.3.RELEASE API)  21/10-2010
PersistentTokenBasedRememberMeServices (Spring Security 3.0.3.RELEASE API)  21/10-2010
RememberMeServices (Spring Security 3.0.3.RELEASE API)  21/10-2010
RememberMeAuthenticationToken (Spring Security 3.0.3.RELEASE API)  21/10-2010
Authentication (Spring Security 3.0.3.RELEASE API)  21/10-2010
SecurityContextLogoutHandler (Spring Security 3.0.3.RELEASE API)  21/10-2010
LogoutFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
PortMapper (Spring Security 3.0.3.RELEASE API)  21/10-2010
LoginUrlAuthenticationEntryPoint (Spring Security 3.0.3.RELEASE API)  21/10-2010
AuthenticationProcessingFilterEntryPoint (Spring Security 3.0.3.RELEASE API)  21/10-2010
ConcurrentSessionFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
HttpSessionContextIntegrationFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
SecurityContextPersistenceFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
ChannelProcessingFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
AnonymousAuthenticationProvider (Spring Security 3.0.3.RELEASE API)  21/10-2010
AnonymousAuthenticationToken (Spring Security 3.0.3.RELEASE API)  21/10-2010
AnonymousAuthenticationFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
AbstractSecurityInterceptor (Spring Security 3.0.3.RELEASE API)  21/10-2010
FilterInvocation (Spring Security 3.0.3.RELEASE API)  21/10-2010
FilterInvocationSecurityMetadataSource (Spring Security 3.0.3.RELEASE API)  21/10-2010
FilterSecurityInterceptor (Spring Security 3.0.3.RELEASE API)  21/10-2010
AccessDeniedHandlerImpl (Spring Security 3.0.3.RELEASE API)  21/10-2010
AccessDeniedHandler (Spring Security 3.0.3.RELEASE API)  21/10-2010
SimpleUrlAuthenticationFailureHandler (Spring Security 3.0.3.RELEASE API)  21/10-2010
AuthenticationFailureHandler (Spring Security 3.0.3.RELEASE API)  21/10-2010
SecurityContextHolderAwareRequestFilter (Spring Security 3.0.3.RELEASE API)  21/10-2010
FilterChainProxy (Spring Security 3.0.3.RELEASE API)  21/10-2010
HttpSessionContextIntegrationFilter (Spring Security 3.0.0.RELEASE API)  21/10-2010

Spring Security

Behind the Spring Security Namespace | SpringSource Team Blog  21/10-2010
Is there an example for registration login with email verify - Spring Community Forums 21/10-2010
Preauthentication failing - authentication object not found in securityContextHolder - Spring Community Forums 21/10-2010
Spring Security: Cannot access target page even after successful login - Stack Overflow 21/10-2010
Spring Security – Tutorial: Adding Security to Spring Petclinic  21/10-2010
SpringMVCandSecurity.pdf (application/pdf-objekt)  21/10-2010
Spring security login and logout form JSP Page Example | Loiane Groner 21/10-2010
10. Remember-Me Authentication  21/10-2010
Chapter 7. Channel Security 21/10-2010
Mixing of user credentials in Spring Security 2.0.4 - Spring Community Forums 21/10-2010
How Spring Security hooks to Central Authentication Service (CAS) | mattfleming.com 21/10-2010
graniteds : Message: Re: defining channelProcessingFilter in acegi and GDS 21/10-2010
ChannelProcessingFilter example? - Spring Community Forums 21/10-2010
OpenId authentication and automatic registration with Spring Security 3.0.2 - Stack Overflow 21/10-2010
[#SEC-1400] Support composition of intercept-url lists - Spring Projects Issue Tracker 21/10-2010
i18n with LocaleChangeInterceptor in WebFlow [Archive] - Spring Community Forums 21/10-2010
Configuring Acegi with jdbc - Spring Community Forums 21/10-2010
Hashing and Salting Passwords with Spring Security 2 - Wheeler Software 21/10-2010

68
Ehcache With A Commercial License And Room To Grow—Ehcache, Distributed Cache, Hibernate Cache | Terracotta 21/10-2010
Production & Developer Support – Simplified Operations & Lower Costs | Terracotta 21/10-2010
Java(TM) Boutique - Tutorial - Add Object Caching Using Spring, AOP, and Ehcache 21/10-2010
FrequentlyAskedQuestions - ehcache-spring-annotations - Frequently Asked Questions - Project Hosting on Google Code 21/10-2010

springmodules-ehcache.xsd 21/10-2010
Hosting the Spring Modules Schemas 21/10-2010
Old Nabble - Ehcache - Ehcache with Spring 21/10-2010
ehcache configuration not working? - Spring Community Forums 21/10-2010
Caching the result of methods using Spring and EHCache - Spring Discuss - Confluence 21/10-2010
Build Ehcache from source 21/10-2010

Tomcat configuration
Apache Tomcat 6.0 - SSL Configuration HOW-TO 21/10-2010
Tomcat and SSL 21/10-2010
InformIT: Creating an SSL Keystore Using the Java Keytool > Generating a Private Key and a Keystore 21/10-2010
InformIT: Creating an SSL Keystore Using the Java Keytool > Chaining and Establishing Identities 21/10-2010
generate a java keystore file - ComHit.net 21/10-2010
The Most Common Java Keytool Keystore Commands 21/10-2010
InformIT: Setting Up a Secure Apache 2 Server > SSL Configuration 21/10-2010
article.aspx 21/10-2010
Apache Tomcat 6.0 - JNDIDatasource HOW-TO 21/10-2010
Apache Tomcat / PostgreSQL Configuration 21/10-2010

MD5 hash
http://hash.online-convert.com/md5-salt 21/10-2010
http://md5.my-addr.com/md5_salt_hash-md5_salt_hash_generator_tool.php 21/10-2010
Understanding hash functions - Wheeler Software 21/10-2010
Hashing and Salting Passwords with Spring Security 2: Discussion - Wheeler Software 21/10-2010
Hashing and Salting Passwords with Spring Security 2 - Wheeler Software 21/10-2010
Configuring your app for authentication against secured passwords - Wheeler Software 21/10-2010
Hash Generator 21/10-2010
Adding salt to your passwords - Wheeler Software 21/10-2010
Configuring your app to secure passwords during registration - Wheeler Software 21/10-2010
MD5 salted hash, md5 salt hash generator Tool, md5 salting 21/10-2010
Lösenordshantering - MD5 och salt - Wikibooks 21/10-2010
Calculate a MD5 hash with salt online 21/10-2010
Class: org.springframework.security.providers.dao.salt.Re... - GrepCode.com - Java Source Code Search 2.0 21/10-2010
Free online md5 hash calculator based on php script 21/10-2010
Java md5 example with MessageDigest 21/10-2010
Get MD5 hash in a few lines of Java [java] [md5] 21/10-2010
Check password salt based on MD5 : MD5 Message Digest algorithm « Security « Java Tutorial 21/10-2010
MD5 hashing: Encodes a string : MD5 Message Digest algorithm « Security « Java Tutorial 21/10-2010
Set password salt : MD5 Message Digest algorithm « Security « Java Tutorial 21/10-2010

HTTP to HTTPS
https to http in spring security - Spring Community Forums 21/10-2010
Switching from https to http - Spring Community Forums 21/10-2010
Bug in Spring Security requires-channel="https" mechanism? - Spring Community Forums 21/10-2010
Problem in switching to HTTPS channel 21/10-2010
[#SEC-1019] Login via https didn’apos;t redirect correctly to http page - Spring Projects Issue Tracker 21/10-2010
Old Nabble - Roller - User - Problem in switching to HTTPS channel 21/10-2010
svn via https - can’apos;t login - BitNami Forums 21/10-2010
https in default-target-url spring security - Stack Overflow 21/10-2010
Have to login in twice with SSL enabled - Spring Community Forums 21/10-2010

70
HTTPS to HTTP switch [Archive] - Spring Community Forums 21/10-2010
channelProcessingFilter, https -> http redirection proble - Spring Community Forums 21/10-2010
How to enable HTTPS in Tomcat | ITWorld 21/10-2010
HTTPS - Secure Server: Example Sites 21/10-2010
HTTP to HTTPS in spring MVC - Spring Community Forums 21/10-2010
Taglib for mixing http/https - Spring Community Forums 21/10-2010
Switch between HTTP and HTTPS looses PHP-session - Yireo 21/10-2010

Debugging

Spring Framework DEBUG logging with Log4J - Spring Community Forums 21/10-2010
Debugging and Spring - Spring Community Forums 21/10-2010
log4j 21/10-2010
Apache Logging Services Project - Apache log4j 21/10-2010
Apache Tomcat 6.0 - Logging in Tomcat 21/10-2010
WTP Tomcat FAQ - Eclipsepedia 21/10-2010
[news.eclipse.webtools] Re: Tomcat Logs 21/10-2010
tomcat ignores logging.properties when started from eclipse 21/10-2010
Finding Tomcat 6 logfiles when used as a server in Eclipse 21/10-2010
[news.eclipse.webtools] Re: where are my tomcat logs ? 21/10-2010
NetBeans Forums - Can&apos;t find catalina.out for Netbeans Tomcat install 21/10-2010
Tailing catalina.out on Windows - WG: Programmer&amp;apos;s Cafe - Confluence 21/10-2010
Tomcat Log4j Log Level Configuration 21/10-2010
where is &amp;quot;catalina.out&amp;quot; defined and enabled? 21/10-2010
Log4j tutorial with Tomcat examples 21/10-2010
Apache log4j 1.2 - Frequently Asked Technical Questions 21/10-2010
Log4j Manual 21/10-2010
log4j: where to put properties file? - Java answers 21/10-2010
log4j.properties Translator 21/10-2010
Log4j Bridge 21/10-2010
log4j-over-slf4j.html 21/10-2010
SLF4J 21/10-2010
How to Use SLF4J with log4j « Simples Assim 21/10-2010
SLF4J News 21/10-2010
SLF4J Error Codes 21/10-2010

Core Spring

Core Spring Datasheet.pdf (application/pdf-object) 21/10-2010
Spring – PropertyPlaceholderConfigurer example 21/10-2010
Spring: Setting up a simple PropertyPlaceholderConfigurer example. - Stack Overflow 21/10-2010
Spring PropertyPlaceholderConfigurer: A nice clean way to share on Dion Almaer&amp;apos;s Blog 21/10-2010
Appendix C. XML Schema-based configuration 21/10-2010
Deprecated List 21/10-2010
SimpleMappingExceptionResolver configuration - Spring Community Forums 21/10-2010
Using SimpleMappingExceptionResolver to pass exception to view. - Spring Community Forums 21/10-2010
SimpleMappingExceptionResolver (Spring Framework API 2.0) 21/10-2010
SimpleMappingExceptionResolver configuration [Archive] - Spring Community Forums 21/10-2010

Tiles

Multiple controllers with tiles - Spring Community Forums 21/10-2010
Salmon Run: A Spring/Tiles Example Web Application on Maven2 21/10-2010
Tiles 101/201 21/10-2010
Apache Tiles 2, integration with Spring-MVC « Richard&amp;apos;s Weblog 21/10-2010
Spring 3 Portlet/Tiles 2 configuration problem - Spring Community Forums 21/10-2010
Apache Tiles 2 - Translation of JSP tags 21/10-2010
Apache Tiles 2, ViewPreparer example. « Richard&amp;apos;s Weblog 21/10-2010
Apache Tiles 2, a brief example. « Richard&amp;apos;s Weblog 21/10-2010
2. Tiles XML Configuration 21/10-2010
Spring by Example&amp;apos;s Dynamic Tiles 2 Spring MVC Module 21/10-2010
Spring 2.5 + Struts 2 + Tiles 2 - Spring Community Forums 21/10-2010
16. View technologies 21/10-2010
How do I create a Spring 3 + Tiles 2 webapp using REST-ful URLs? - Stack Overflow 21/10-2010
3. JSP Example 21/10-2010
2. Tiles XML Configuration 21/10-2010
Simple Tiles 2 Spring MVC Webapp 21/10-2010
Web - Spring Community Forums 21/10-2010
ComponentControllerSupport 21/10-2010
ComponentControllerSupport is gone from spring 2.1. - Spring Community Forums 21/10-2010
Tiles & ComponentControllerSupport - Spring Community Forums 21/10-2010
Using the ViewPreparer - question on migration to Tiles2 21/10-2010
Apache Tiles 2 - Java APIs considerations 21/10-2010
Apache Tiles - Framework - Migration from Struts-Tiles 21/10-2010
Apache Tiles - Framework - Home 21/10-2010
Apache Tiles 2 - Home 21/10-2010
ComponentContext (Tiles 2 2.2.2 API) 21/10-2010
ComponentContext (Tiles API Documentation) 21/10-2010
ComponentContext (Struts Struts API Documentation) 21/10-2010
ComponentContext (Apache Struts API Documentation) 21/10-2010
Using CSS in Spring MVC with tiles support - Spring Community Forums 21/10-2010
spring:message and garbled character encoding - Spring Community Forums 21/10-2010
Apache Tiles 2 - Tiles Configuration 21/10-2010
Apache Tiles 2 - Creating Tiles Pages 21/10-2010
Apache Tiles 2 - Nesting and Extending Definitions 21/10-2010
Apache Tiles 2 - Tiles Concepts 21/10-2010
Apache Tiles 2 - Tiles Basic Usage 21/10-2010
Apache Tiles 2 - Tutorial 21/10-2010
Programming Jakarta Struts: Using Tiles, Part 4 - O'Reilly Media 21/10-2010
struts1.2.7 and tiles definitions 21/10-2010
Tiles - Tile Definitions, Create a Tile Definition 21/10-2010
Using tiles-defs.xml in Tiles Application 21/10-2010
Uses of Interface org.apache.tiles.template.AttributeResolver (Tiles 2 2.2.2 API) 21/10-2010
[#SPR-5411] TilesConfigurer class no longer works with Tiles 2.1.1 - Spring Projects Issue Tracker 21/10-2010
I18nFactorySet (Tiles 2 1.3.10 API) 21/10-2010
Apache Tiles - Framework - Tiles Configuration 21/10-2010
Struts 2 Tiles Example 21/10-2010
tiles-config_2_1.dtd 21/10-2010
dtds 21/10-2010
Re: Using tiles without the tiles xml definition - msg#00005 - users-tiles.apache.org 21/10-2010
2. Tiles XML Configuration 21/10-2010
3. Tiles JSP Example 21/10-2010
Integrating Tiles with Spring using Struts - Spring Community Forums 21/10-2010
Struts Tutorial 21/10-2010
first-steps-struts-tiles-tutorial.html 21/10-2010
Java Struts Tiles Tutorial 21/10-2010
Struts Tiles Example 21/10-2010
Developing Simple Struts Tiles Application 21/10-2010
Tiles - Installing Tiles 21/10-2010
Struts Tiles - Examples 21/10-2010
Apache Tiles - Framework - Getting started 21/10-2010

Spring MVC

SimpleFormController in V3.0 - Spring Community Forums 21/10-2010
JavaServer Pages (JSP) and JSTL - must be used with a prefix when a default namespace is not specified 21/10-2010
Outputting stack traces in a JSP using JSTL | Isocra 21/10-2010
Call a bean function from JSTL : jstl, function, call, bean 21/10-2010

72
insert or update violates foreign key constraint. why? 21/10-2010
Hibernate Community • View topic - Hibernate insert statement with missing columns 21/10-2010
INSERT LOW_PRIORITY or INSERT DELAYED in Hibernate - Stack Overflow 21/10-2010
[#HHH-4341] There is no annotation to specify default value for a column - Hibernate JIRA 21/10-2010
Tadayakaposcs Blog: Hibernate: Annotation one-to-one (foreign-key) 21/10-2010
Hibernate Community • View topic - could not get next sequence value 21/10-2010
Getting Started Quickly with Hibernate Annotations | OcpSoft 21/10-2010
Hibernate Community • View topic - Foreign key using hibernate Annotation 21/10-2010
ForeignKey (Hibernate Annotations API Documentation) 21/10-2010
Spring 2.5 + Hibernate 3.2 (Foreign Generator) = HibernateSystemException - Spring Community Forums 21/10-2010
Hibernate annotations mapping id generator foreign: hibernate, generator, foreign, id, annotation 21/10-2010
Hibernate Annotations 21/10-2010
Sequence + Hibernate Annotations 21/10-2010
topicview.php 21/10-2010
Hibernate Community • View topic - GENERATING SEQUENCES 21/10-2010
Blablabla: User Postgresql SERIAL field type in Hibernate 21/10-2010
Old Nabble - dotCMS Users Forum/Mailing List - hibernate_sequence missing in distribution 1.5.1 21/10-2010
GenerationType (Java EE 5) 21/10-2010
hibernate_sequence does exists ERROR (Object Relational Mapping forum at JavaRanch) 21/10-2010
Manning Forums: "hibernate_sequence" ... 21/10-2010
Hibernate Community • View topic - "hibernate_sequence" does not exist 21/10-2010
[Closed] Another Noob: Spring/Hibernate/PostgreSQL 21/10-2010
problem saveOrUpdate object Hibernate (a different object with the same identifier session) - Stack Overflow 21/10-2010
hibernate template saveOrUpdate, hibernate template saveOrUpdate Tutorial, Tutorials hibernate template saveOrUpdate, Example code hibernate template saveOrUpdate, Java hibernate template saveOrUpdate 21/10-2010
Hibernate Query by example and saveOrUpdate [Archive] - Spring Community Forums 21/10-2010
hibernate: could not get next sequence value 21/10-2010
Hibernate Community • View topic - could not get next sequence 21/10-2010
Hibernate + SQLGrammarException : could not get next sequence value (Object Relational Mapping forum at JavaRanch) 21/10-2010
Tadayakaposcs Blog: datasource-proxy framework 21/10-2010
Tadayakaposcs Blog 21/10-2010
SQLGrammarException "hibernate_sequence" [closed] 21/10-2010
Understanding hibernate <generator> element 21/10-2010
Simple Association (one-to-one) 21/10-2010
Hibernate Community • View topic - referencing unique part of a composite key fails 21/10-2010
Hibernate Community • View topic - Hibernate3 and nested classes (or extending hibernate?) 21/10-2010
[#ANN-99] Embedded annotation should support a column prefix - Hibernate JIRA 21/10-2010
Hibernate Annotations for a One-To-Many Mapping Featuring a Many Side, Composite, Primary Key having a Composite, Foreign, Sub-Key « Beaver Creek Consulting 21/10-2010
Hibernate Annotation Join table question - Stack Overflow 21/10-2010
Collection of Elements 21/10-2010
Hibernate Annotations 21/10-2010
Hibernate Tutorial, Hibernate 3 Tutorials, Java Hibernate Example, Hibernate Online Code 21/10-2010
Search results for hibernate nested tables - MarkMail 21/10-2010
Hibernate Community • View topic - Hibernate and Postgresql-8.2 21/10-2010
No identifier specified for entity - Spring Community Forums 21/10-2010
Hibernate Annotations 21/10-2010
Getting Started Quickly with Hibernate Annotations 21/10-2010
Database design for hibernate - DevX.com Forums 21/10-2010
Hibernate: getting a sorted loadAll() result... - Spring Community Forums 21/10-2010
&amp;apos;Hibernate Dialect property&amp;apos; for Different Databases « Lijin’s Localhost 21/10-2010
Writing First Hibernate Code 21/10-2010
New To Java - "Hibernate Dialect must be explicitly set" error occured [Locked] 21/10-2010

75
In Relation To... Hibernate 3.6.0.Beta2 Release 21/10-2010
Hibernate - JBoss Community 21/10-2010

Database

PgAdmin Internals - PostgreSQL Wiki 21/10-2010
EXECUTE SCRIPT 21/10-2010
ERROR: column 'xxx' does not exist (under v. 7.4.1) - PostgreSQL Database answers 21/10-2010
PostgreSQL won't work with Spring and Tomcat - Spring Community Forums 21/10-2010
JDBC With Postgresql 21/10-2010
MyEclipse not picking up Tomcat Library Path 21/10-2010
setting up postgres 8.3.0 with tomcat 6 : postgresql, apache, postgresql, tomcat, 8.3.0, 6, jsp, java 21/10-2010
jndi-datasource-examples-howto.html 21/10-2010
Show databases, Show tables, Describe table - Dev Shed 21/10-2010
#53 (Hierarchical view for inherited tables) – pgAdmin III 21/10-2010
View Table Structure - Dev Shed 21/10-2010
Display or show tables in a PostgreSQL/PgSQL database | PostgreSQL PgSQL | Tech-Recipes 21/10-2010

Java Time

Java Programming [Archive] - java.util.Date - How to get current date time? 21/10-2010
Calendar (Java Platform SE 6) 21/10-2010
Working in Java time - JavaWorld 21/10-2010
Java Database Connectivity (JDBC) - Cannot load JDBC driver class &apos;com.mysql.jdbc.Driver&apos;; [Locked] 21/10-2010
Using the PostgreSQL Driver 21/10-2010

Miscellaneous

Gang of Four - Wikipedia 21/10-2010
Design Patterns (bok) - Wikipedia 21/10-2010
Manning: Spring in Action, Second Edition 21/10-2010
Database normalization - Wikipedia, the free encyclopedia 21/10-2010
Sixth normal form - Wikipedia, the free encyclopedia 21/10-2010
Fifth normal form - Wikipedia, the free encyclopedia 21/10-2010
Fourth normal form - Wikipedia, the free encyclopedia 21/10-2010
Third normal form - Wikipedia, the free encyclopedia 21/10-2010
Second normal form - Wikipedia, the free encyclopedia 21/10-2010
First normal form - Wikipedia, the free encyclopedia 21/10-2010
Backus-Naur Form - Wikipedia, the free encyclopedia 21/10-2010
Konventioner för vetenskapliga beteckningar - Wikipedia 21/10-2010
Hibernate Made Easy: Simplified Data Persistence with Hibernate and Jpa (Java Persistence API) Annotations (0615201954) av Cameron Wallace McKenzie - Adlibris bokhandel 21/10-2010
Amazon.com: Hibernate Made Easy: Simplified Data Persistence with Hibernate and... 21/10-2010

Unsorted

Spring-Loaded 21/10-2010
OTN Discussion Forums : Unable to integrate Spring with ... 21/10-2010
open source framework, web application software development | Flex - Adobe 21/10-2010
How do you make Eclipse recognize JEE jar files so Servlets can compile? - Stack Overflow 21/10-2010
Eclipse: How to add Tomcat Library? (IDEs, Version Control and other tools forum at JavaRanch) 21/10-2010
Print the Stack Trace of the Exception to a String 21/10-2010
[#RF-2320] The function must be used with a prefix when a default namespace is not specified PROBLEM - JBoss Issue Tracker 21/10-2010
StackTraceElement (Java 2 Platform SE v1.4.2) 21/10-2010
Notes on using the Jakarta ORO &lt;code&gt;Perl5Util&lt;/code&gt; package 21/10-2010
Java RegEx Tutorial 21/10-2010
RegEx: online regular expression testing 21/10-2010
Regular Expressions and the Java Programming Language 21/10-2010
8 Attachments

8.1 Attachment 1 RoadRantz.war

This war file is delivered on request.
8.2 Attachment 2 Market study result

This attachment has short descriptions of the companies in the market study and what frameworks they used. The following information is public information gathered from the companies own job adds.

8.2.1 Trade Doubler

Vi är ett internationellt företag inom prestationsbaserad digital marknadsföring. Vår digitala marknadsföringsportfölj omfattar: Affiliatemarknadsföring, Display- och resultatbaserade onlinekampanjer, Sökordshantering och Teknologi som administrar alla dina sökordmarknadsföringsaktiviteter i olika medier.

JavaScript, ExtJS (2.x, 3.x), Freemaker, Velocity
JSP, Sitemesh, Servlets, Webwork, Struts 2
JetbrainsIDEA 7 (9), Eclipse, Subversion, Teamcity
Maven, Ant, Selenium, SQL Developer
Jmeter, Junit, Hamcrest
EJB 2.1, EJB 3.0, Hibernate, Spring, JDBC
Tomcat, Jboss
JMX, RMI, Web service (soap)
TDD, BDD, OOP, DDD

8.2.2 Paxport systems

Our products and services already allow some of the industry’s biggest airlines and tour operators to focus on their core business instead of passenger management. Handling more than 20 million passengers every year – while increasing our customers’ competitiveness and generating additional revenue – we are experts in passenger management solutions.

EJB3 , JPA
Wicket , IText
Swing , Jgoodies
Hessian , Junit , TestNG , Log4J

8.2.3 EDB bank och finans

EDB är en ledande leverantör av IT-tjänster i Norden. Vi tror på ett nära samarbete med våra kunder där vi kombinerar vår tekniska kompetens och djupa branschkunskap med en betydande internationell leveranskapacitet.

Java SE och Java EE
JSF, Spring , JUnit
WebLogic, Tomcat
Web Services, Enterprise Service Bus
HTML och JavaScript
### 8.2.4 ConOne

Vår affärsidé är att tillhandahålla och utveckla kvalificerade IT-konsulter inom systemutveckling, infrastruktur och kommunikation. Vi levererar kunskap och erfarenhet på ett affärsorienterat, modernt och drivande sätt.

- Java EE/SE/J2EE/J2SE,
- Tomcat, CMS, SOA, WebLogic,
- JSP, JSF, JPA, RichFaces, Facelets, Servlets,
- Agile, SCRUM, Glassfish, Eclipse,
- Spring, Databaser,
- JDBC, SQL, MySQL, MSSQL,
- DB2, Oracle, Webservices,
- XML, SOAP, jRuby,
- JBoss, Hibernate,
- Maven, Hudson, JUnit,
- Javascript, AJAX, jQuery.

### 8.2.5 HiQ

HiQ är ett IT- och managementkonsultbolag - specialiserat på kommunikation och mjukvaruutveckling. Med Nordiskt hjärta, och som renodlat konsultbolag, behärskar vi tekniken och hur våra kunder tjänar pengar på den.

- EJB 3, Hibernate, Glassfish,
- JBoss, Webservices, jUnit,
- Hudson, Eclipse

### 8.2.6 InfoTechGroup

Företagets kärnverksamhet består i att tillhandahålla effektiva lösningar inom IT, både genom konsultverksamhet och genom egenutvecklade system inom datalagring och kommunikation.

- Java EE
- JBoss, Struts och Spring
- Oracle-DB, SQL och XML

### 8.2.7 Folksam

Folksam är ett av Sveriges främsta försäkringsbolag med både sak- och livförsäkringar för privatpersoner och företag. Vi ägs av våra fyra miljoner kunder och som anställd hos oss blir du en av 3 800 medarbetare över hela landet.

- Java EE (J2EE) och XML.
SOA, MOM, Web Services, XSLT, 
WSDL, SOAP, Jboss, Eclipse, Maven, Hudson, 
Subversion, UI, MQ, DB2, WebSphere

### 8.2.8 Valtech

Valtech hjälper företag och organisationer att lyckas med sina IT-satsningar, både i Sverige och utomlands. Vår ambition är att arbeta med de roligaste projekten och den senaste tekniken. Ofta går vi in där någon annan har gått bet.

Hibernate, Spring, Selenium, JUnit, Struts 2.0, 
EJB 3, JBoss Seam, DI, CI, TDD, Maven, Subversion

### 8.2.9 Exigen services

Exigen Services är en ledande leverantör av tjänster inom applikations-outsourcing. Vi kombinerar kompetens i världsklass, erkänd expertis inom utvecklingsmetodologi, och branscherfarenhet för att minimera risker, sänka kostnader och leverera resultat.

JSF, Spring, Hibernate, Struts. 
BEA WebLogic, IBM WebSphere, OC4J, JBoss AS eller Sun GlassFish. 
JAAS, Java ACC, Acegi/Sprint
8.3 Attachment 3 Market study questionnaire

1. Använder ni Java EE i er verksamhet?
2. Vilket operativsystem använder ni?
3. Vilken applikationsserver använder ni?
4. Hur stor del av IT-strukturen är implementerad i Java EE?
5. Vilka ramverk använder ni för er implementation och utveckling?
   Exempel: EJB, Hibernate, Struts, Spring, AJAX.
6. För varje ramverk, anser ni att ramverket är:
   1: onödig
   2: användbar
   3: oumbärlig
7. Kan ni tänka er att ge en kort intervju vid en senare tidsperiod där ni djupare förklarar ert bruk av vissa ramverk?
   (Denna punkt utnyttjades aldrig)
## Table of Contents

1 Introduction .................................................................................................................. 86
  1.1 Market study .............................................................................................................. 86
  1.2 Document organization ........................................................................................... 87
2 Java EE ........................................................................................................................... 87
  2.1 JSP ............................................................................................................................. 88
  2.2 JSTL ........................................................................................................................... 88
  2.3 Servlets ...................................................................................................................... 88
  2.4 EJB 4 ......................................................................................................................... 88
3 Web application problems .............................................................................................. 88
  3.1 List .............................................................................................................................. 89
  3.2 Security ..................................................................................................................... 89
4 Problem areas, frameworks and technologies ................................................................. 89
  4.1 Common ..................................................................................................................... 89
    4.1.1 Problem areas ...................................................................................................... 89
    4.1.1.1 POJO ................................................................................................................. 90
    4.1.1.2 Containers ....................................................................................................... 90
    4.1.1.3 MVC pattern .................................................................................................... 90
    4.1.1.4 Dispatcher Pattern ......................................................................................... 90
    4.1.1.5 Command Pattern ......................................................................................... 90
    4.1.1.6 Dependency Injection Pattern 1 ...................................................................... 90
    4.1.1.7 Factories (Pattern) .......................................................................................... 91
    4.1.1.8 RDBMS ............................................................................................................ 91
    4.1.1.9 ORM ............................................................................................................... 92
    4.1.1.10 Persistence .................................................................................................... 92
    4.1.1.11 UI 1 ............................................................................................................... 93
    4.1.1.12 IoC .................................................................................................................. 93
    4.1.1.13 AOP ............................................................................................................... 93
    4.1.1.14 SOA 2 .......................................................................................................... 93
    4.1.1.15 Cloud Computing ......................................................................................... 93
    4.1.1.16 MOM 1 ........................................................................................................ 94
    4.1.1.17 DAO .............................................................................................................. 94
    4.1.1.18 DAL .............................................................................................................. 94
    4.1.2 Technologies ....................................................................................................... 94
      4.1.2.1 SQL 3 ............................................................................................................. 94
      4.1.2.2 MySQL ......................................................................................................... 94
      4.1.2.3 PostgreSQL ................................................................................................. 94
      4.1.2.4 JDBC 3 ......................................................................................................... 95
      4.1.2.5 XML 3 .......................................................................................................... 95
      4.1.2.6 JavaScript 3 ................................................................................................. 95
<table>
<thead>
<tr>
<th>Section</th>
<th>Technologies</th>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1.7</td>
<td>AJAX 1</td>
<td>95</td>
</tr>
<tr>
<td>4.2.1.8</td>
<td>Web Services 5</td>
<td>96</td>
</tr>
<tr>
<td>4.2.1.9</td>
<td>SOAP 2</td>
<td>96</td>
</tr>
<tr>
<td>4.2.1.10</td>
<td>MQ 1</td>
<td>97</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Frameworks</td>
<td>97</td>
</tr>
<tr>
<td>4.1.3.1</td>
<td>Spring 7</td>
<td>97</td>
</tr>
<tr>
<td>4.1.3.2</td>
<td>Hibernate 6</td>
<td>97</td>
</tr>
<tr>
<td>4.1.3.3</td>
<td>JSF 4</td>
<td>98</td>
</tr>
<tr>
<td>4.1.3.4</td>
<td>Struts 5</td>
<td>98</td>
</tr>
<tr>
<td>4.1.3.5</td>
<td>Tiles</td>
<td>98</td>
</tr>
<tr>
<td>4.1.3.6</td>
<td>Ant 1</td>
<td>98</td>
</tr>
<tr>
<td>4.1.3.7</td>
<td>Maven 4</td>
<td>98</td>
</tr>
<tr>
<td>4.1.3.8</td>
<td>Hudson 3</td>
<td>99</td>
</tr>
<tr>
<td>4.1.3.9</td>
<td>JUnit 5</td>
<td>99</td>
</tr>
<tr>
<td>4.1.3.10</td>
<td>Log4J 1</td>
<td>99</td>
</tr>
<tr>
<td>4.2</td>
<td>Rare</td>
<td>99</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Problem areas</td>
<td>99</td>
</tr>
<tr>
<td>4.2.1.1</td>
<td>CMS 1</td>
<td>100</td>
</tr>
<tr>
<td>4.2.1.2</td>
<td>ESB 1</td>
<td>100</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Technologies</td>
<td>100</td>
</tr>
<tr>
<td>4.2.2.1</td>
<td>Swing 1</td>
<td>100</td>
</tr>
<tr>
<td>4.2.2.2</td>
<td>JPA 2</td>
<td>100</td>
</tr>
<tr>
<td>4.2.2.3</td>
<td>RMI 1</td>
<td>100</td>
</tr>
<tr>
<td>4.2.2.4</td>
<td>JMX 1</td>
<td>101</td>
</tr>
<tr>
<td>4.2.2.5</td>
<td>XSLT 1</td>
<td>101</td>
</tr>
<tr>
<td>4.2.2.6</td>
<td>Hessian 1</td>
<td>101</td>
</tr>
<tr>
<td>4.2.2.7</td>
<td>Selenium 1</td>
<td>101</td>
</tr>
<tr>
<td>4.2.2.8</td>
<td>JRuby 1</td>
<td>102</td>
</tr>
<tr>
<td>4.2.2.9</td>
<td>SiteMesh 1</td>
<td>102</td>
</tr>
<tr>
<td>4.2.2.10</td>
<td>IText 1</td>
<td>102</td>
</tr>
<tr>
<td>4.2.2.11</td>
<td>jQuery 1</td>
<td>102</td>
</tr>
<tr>
<td>4.2.2.12</td>
<td>ExtJS 1</td>
<td>102</td>
</tr>
<tr>
<td>4.2.2.13</td>
<td>UDDI</td>
<td>102</td>
</tr>
<tr>
<td>4.2.2.14</td>
<td>WSDL 1</td>
<td>103</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Frameworks</td>
<td>103</td>
</tr>
<tr>
<td>4.2.3.1</td>
<td>WebWork 1</td>
<td>103</td>
</tr>
<tr>
<td>4.2.3.2</td>
<td>Tapestry</td>
<td>103</td>
</tr>
<tr>
<td>4.2.3.3</td>
<td>FreeMarker 1</td>
<td>103</td>
</tr>
<tr>
<td>4.2.3.4</td>
<td>Velocity 1</td>
<td>103</td>
</tr>
<tr>
<td>4.2.3.5</td>
<td>RichFaces 1</td>
<td>104</td>
</tr>
<tr>
<td>4.2.3.6</td>
<td>Facelets 1</td>
<td>104</td>
</tr>
<tr>
<td>4.2.3.7</td>
<td>Wicket 1</td>
<td>104</td>
</tr>
<tr>
<td>4.2.3.8</td>
<td>JGoodies 1</td>
<td>104</td>
</tr>
<tr>
<td>4.2.3.9</td>
<td>TestNG 1</td>
<td>104</td>
</tr>
<tr>
<td>Section</td>
<td>Content</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>4.2.3.10</td>
<td>Jmeter 1</td>
<td></td>
</tr>
<tr>
<td>4.2.3.11</td>
<td>Hamcrest 1</td>
<td></td>
</tr>
<tr>
<td>4.2.3.12</td>
<td>JAAS 1</td>
<td></td>
</tr>
<tr>
<td>4.2.3.13</td>
<td>Java ACC 1</td>
<td></td>
</tr>
<tr>
<td>4.2.3.14</td>
<td>Acegi/Spring Security 1</td>
<td></td>
</tr>
<tr>
<td>4.2.3.15</td>
<td>SQL Developer 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Servers</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Web servers</td>
<td></td>
</tr>
<tr>
<td>5.1.1</td>
<td>Apache HTTP Server 1</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Application servers</td>
<td></td>
</tr>
<tr>
<td>5.2.1</td>
<td>WebLogic 3</td>
<td></td>
</tr>
<tr>
<td>5.2.2</td>
<td>GlassFish 4</td>
<td></td>
</tr>
<tr>
<td>5.2.3</td>
<td>Oracle Application Server 10g 2</td>
<td></td>
</tr>
<tr>
<td>5.2.4</td>
<td>WebSphere 2</td>
<td></td>
</tr>
<tr>
<td>5.2.5</td>
<td>JBoss 8</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Servlet Containers</td>
<td></td>
</tr>
<tr>
<td>5.3.1</td>
<td>Tomcat 3</td>
<td></td>
</tr>
<tr>
<td>5.3.2</td>
<td>OC4J 1</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Database Servers</td>
<td></td>
</tr>
<tr>
<td>5.4.1</td>
<td>DB2 2</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Integration Servers</td>
<td></td>
</tr>
<tr>
<td>5.5.1</td>
<td>Teamcity 1</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>Revision control systems</td>
<td></td>
</tr>
<tr>
<td>5.6.1</td>
<td>Subversion 3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Integrated Development Environments (IDEs)</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Eclipse 4</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>NetBeans 1</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>IntelliJ 1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Programming techniques</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Java Archives</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>JAR</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>WAR</td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>RAR</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>EAR</td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>SAR</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

One of the points of this thesis is to investigate and clarify the jungle of frameworks, concepts and technologies concerning Java Enterprise application development and learn to navigate among their acronyms. It is also aimed at investigating the frameworks that are used in today’s business world in order to not waste any effort learning deprecated technologies. Therefore a market study has been carried out among Java EE web development companies. This market study and the investigations carried out in this document will serve as the foundation for the choices of frameworks and technologies that I will use to build a sample web application at the end of this thesis.

This document attempts to explain the number of frameworks and technologies that were the result of this market study. It also includes a discussion and analysis in the end about what technologies are recommended for implementation. Of course there are many more frameworks out there that did not show up in the market study that might be interesting. Therefore the mindset while writing was to include any framework, concept or technology that seemed to be interesting or popular in the context of Enterprise web development in Java EE.

1.1 Market study

I needed to find companies that performed their web development almost exclusively in Java EE. My way of finding a list of such companies was to look at job advertisements and search for jobs containing words like “Java EE” (or J2EE), “Web development” and “Enterprise development”. Many of these job advertisements include detailed descriptions about what a job applicant is required/desired to know. Also, if a company is using Java well enough to be able to hire more people, then it is reasonable to assume that their technology stack/composition is at least not terrible.

I also performed a phone survey to companies that I thought might be interesting that did not have any advertisements, i.e. large companies with big IT departments. These phone surveys where largely fruitless as it was difficult to get hold of someone that was knowledgeable or interested in sharing information. When such a person was found it was often a matter of giving examples of frameworks and the interviewee gave yes or no answers, resulting in more statistics but no new frameworks.

The companies that in the end gave valuable information where nine in number and gave a surprisingly rich variety of frameworks. These companies where:

- Trade Doubler
- Paxport Systems
- EDB bank och finans
- ConOne
- InfoTechGroup
- Folksam
- Valtech
- Exigen services
• HiQ

The names of these companies will appear in the context of each respective technology as references.

For a summarized list of these companies along with their technologies, see appendix 1.

For the questionnaire used in the market study, see appendix 2.

1.2 Document organization

This document is organized in order of relevance and “need to know” for further reading. The “Common” and “Rare” sections separates the interesting information from the less interesting. What is interesting or not is in many cases a crude line defined by the author and based on experience while researching the information and on the market study. Almost all the different frameworks and technologies in this document have a number applied to them. This number indicates how many of the companies in the market study that used the particular technology or framework in question.

This document is also organized according to the type of technology in question. The three types are:

1. Problem areas
2. Technologies
3. Frameworks

These three types are defined by the author for the purpose of organizing this document and are in no way a standardized way of organizing software. Problem areas describes problems in Enterprise application construction. Technologies describes components in the Java EE language as well as technologies that are outside of the Java language. Frameworks describes third party software designed for Java and Java EE.

This document can be used as a reference if the reader needs to refresh his/her memory during the reading of this thesis.

Note: Sometimes a reference (typically an internet site) may have good phrases and formulations that suits the topic at hand. In such cases, when there is no need to rephrase the information, I have trimmed and copied those phrases to this document and put them inside quotation marks.

2 Java EE

Since this theses is focused on front-end server-side web application development techniques that are used in today's business world, not all of the features in the Java EE API will be included. However it is worth mentioning that the Java EE API has solutions for many of the frameworks and respective problem areas that are presented in this document.
2.1 JSP

JSP stands for Java Server Pages and is a technology for embedding Java code inside HTML documents for creating dynamic web pages. JSP also incorporates different tags for getting and setting data. The applications server interprets the Java code and creates a static HTML document out of it and sends it to the client (web browser). There is nothing prohibiting you from also embedding JavaScript.

You start your Java code with a <\% mark and end it with \%. If you for example want to create a list out of some data, you put your Java inside the <\% \%> tags and the HTML outside. Short example:

```java
<\%
for(int i = 1, i <= 5, i++){

I am row nr <%=i%> of 5 <\br>
\%
\%
```

2.2 JSTL

JSTL is a tag library for JSP pages. JSTL requires less amount of code than plain Java to create dynamic JSP web pages.

2.3 Servlets

Servlets are Java classes that are able to accept HTTP requests. Servlets have two important objects, namely HttpServletResponse and HttpServletRequest. These two classes has many functions but their primary functions are: The request object is used to receive data from the client via HTTP and the response object is used to construct a response HTML document, or to redirect to a JSP file that creates a HTML that is sent to the client.

2.4 EJB

EJB stands for Enterprise Java Bean. Its purpose is to simplify enterprise application development such as transactions and security. Early versions of EJB however complicated the development in other aspects of development by mandating deployment descriptors and plumbing code (code necessary to make the technology work but that did not create any new functionality). EJB 3.0 has solved many of these problems and now supports central concepts such as Dependency Injection (DI) and Aspect Oriented Programming (AOP).

[Spring in action, Second Edition - Book]
(HiQ)(ValTech)(Trade Doubler)(Passport Systems)

3 Web application problems

In this section I have tried to compile a list of as many specific web development problems as possible. These are the kind of problems that front end frameworks often set out to solve. This thesis presents a solution for all of these topics except testing.
3.1 List

- URL construction
- Request dispatch
- User input validation
- Localization/Internationalization
- Unit/Functional/Integration testing
- Exception reporting
- Logging
- Data storage and retrieval
- Caching
- Transactions
- Security
- Web page construction
- MVC architecture
- Loose coupling of objects

3.2 Security

Security in web development is a large topic. I will not dig deep in this topic in this document. However I took the liberty of searching for a up-to-date Internet source dealing with the issue. I found a top-10 list comprised by OWASP (Open Web Application Security Project), delivered in an article by darkreading.com.


4 Problem areas, frameworks and technologies

This chapter describes all the frameworks and technologies that was the result of the market study as well as some fundamental problem areas.

4.1 Common

This section describes the most common frameworks, technologies and problem areas based on the market study and author experience. See chapter 1: Introduction.

4.1.1 Problem areas

This section describes problem areas in programming in connection to web development.
4.1.1 POJO

POJO stands for Plain Old Java Object.

4.1.1 Containers

A container is a system that contains and manages the lifecycle and configuration of application objects. Examples of containers are frameworks such as Spring, web servers such as Apache HTTP server and application servers such as Jboss or Apache Tomcat.

[Spring in action, Second Edition - Book]

4.1.1.2 MVC pattern

MVC stands for Model View Controller. It is the de facto standard design pattern when developing web applications. Its goal is to separate the UI (User Interface) from the business logic. The view contains everything that has to do with the UI. The controller directs the different requests that enters the application and decides what view to use and what data that goes with that view. The model contains all business logic of the application. Further division of the Model can be done to separate the business logic from DAO (Data Access Objects) that operates against a database and even further division can be done with the help of DI (Dependency Injection) and AOP (Aspect Oriented Programming).

4.1.1.3 Dispatcher Pattern

The dispatcher pattern is a design pattern that aims to separate the view from the business and system logic. It is therefore responsible for view management and navigation. If one implements the MVC pattern then Dispatcher Pattern handles the Controller part. [1] is a link to a picture of the flow.


4.1.1.4 Command Pattern

Command Pattern is a design pattern that aims to achieve complete decoupling between objects of different kind, i.e. an object should not need to have any information about another objects interface. It should simply be able to request the execution of an operation.


4.1.1.5 Dependency Injection Pattern

DI stands for Dependency Injection. Instead of objects creating or looking for dependent objects themselves, DI ensures that an object is given its dependencies without it having to ask for them.

A common way to implement DI is with the help of Factories.
4.1.1.6 Factories (Pattern)

Sometimes a object in an application may not have the appropriate information to know what type of sub class in a class hierarchy to instantiate at run time, or there might be complex operations necessary to instantiate an object. Factories are helper classes that helps such an object to instantiate dependency classes. The very point of this is to separate the code that only handles instantiation from the business logic.

An example of this might be an object that uses an Employee class. The Employee class has two subclasses, namely Salesman and Administrator. If someone tries to log in to this system, the supplied information will help determine if the user is an administrator or a salesman. The Factory takes care of which type of class to instantiate and returns the instantiated object to the object that needs it.

Factories are the implementation of the Factory method Pattern [1].

4.1.1.7 Front-end and back-end

The front end and back end of an application can be defined in slightly different ways. In this thesis, front end is anything having to do with View and Controller aspects of the MVC pattern and back end is anything having to do with the Model aspect of the MVC pattern.

Sometimes in some applications, the division between the Model and the Controller/View is very strong, i.e. the model can be very big and complex. In such systems, things like simple database access for the purpose of controller orientation and view population is considered to belong to the front end.

Some applications might also be the other way around i.e. the view is the front end and the controller and model is the back end. It all depends on the complexity and size of the application.

4.1.1.8 RDBMS

RDBMS stands for Relational Database Management System. A relational database is a system that stores information in the form of tables with rows and columns. Typically the information is stored row-wise with each column in the row representing a specific property. The data can be stored as a character, string, integer, float or a more complex data type.
ORM stands for Object-relational Mapping. In this case, “Object” refers to an object in a programming language like Java, and “Relational” refers to tables in relational databases.

Persistence is a way of preserving (or outliving) the state of a process even after the process expires. A program that does not have persistent processes stores its state in the RAM (Random Access Memory) of the computer. If the RAM loses power, i.e. if the computer shuts down, the state of the process is lost. Common computer programs like word processors or games achieve persistence by saving the state of the program in a non-volatile storage such as files on a hard drive (Microsoft word .doc files for example). These files are often specially designed to fit the program. However, using this technique in the development of every single application of any kind would mean that specially designed file types would have to be designed for every application, and searching for data in these files would require the programmers to spend time on search algorithms. This is why “ready to use” database systems have become the standard solution in many fields of programming (and web programming in particular).

Still, even though one has the power of a SQL database at ones disposal, using a technique like JDBC makes it hard to achieve persistence. It is up to the programmer to design the database and the SQL queries so that the program runs efficiently. If persistence is required, the programmer must implement this in a custom way. In this area there are many problems to overcome:

- Object-relational impedance mismatch: The internal structure of an object oriented system and a system of related tables is fundamentally different.
- Writing efficient queries.
- Large amount of database specific code.
- Structural changes in objects or in database.
- Caching data, Is an attribute up to date?
- Version and time stamping

All this comes with high development time and development cost.

Persistence in Java can by example be implemented with Java Persistence API or with the Hibernate framework.
UI stands for User Interface. It may also sometimes be referred to as a GUI (Graphical User Interface). The most common type of UI in web development is HTML web pages. Common technologies that help the development of dynamic web pages include JavaScript, AJAX. These technologies are client-side and is executed in the browser.

(Folksam)

IoC stands for Inversion of Control and is another word for DI (Dependency Injection) [1]

Note: Martin Fowler is a highly respectable author, speaker, and consultant on the design of enterprise software and is often referenced to in computer science articles concerning Java EE.

AOP stands for Aspect Oriented Programming. It is a technique that separates business logic from system services and supporting functions (such as transactions and logging). It does this by applying “aspects” to a program. Aspects are pieces of code that can be called in a program without explicitly stating the call in the code. This is done with point-cuts that are defined outside the code, often in xml files. A point-cut can be defined to intercept a call to a certain function A, and invoke before and after functions to A.

[Spring in action, Second Edition - Book]

SOA stands for Service Oriented Architecture. An example of this is Web services.

[http://www.roseindia.net/webservices/] 11/07 – 2010
(ConOne)(Folksam)

Cloud computing is basically a concept where clients can connect to a “cloud” of services though a single point of entry that does not require the clients to be aware of any of the underlying technology infrastructure. A cloud can be access through a Web service for intercommunication between applications or through a web browser.

### 4.1.1.16 MOM

MOM stands for Message Oriented Middleware. An example of MOM is IBM WebSphere MQ.

(Folksam)

### 4.1.1.17 DAO

DAO stands for Data Access Object and are abstract interfaces designed for storing and retrieving data from a database. DAOs should be implemented as POJOs.

[Spring in action, Second Edition - Book]

### 4.1.1.18 DAL

DAL stands for Data Access Layer. In an Java EE application the DAL can be implemented using DAOs.

### 4.1.2 Technologies

This section describes components in the Java EE language as well as technologies that are disconnected from the Java language.

### 4.1.2.1 SQL

SQL stands for Structured Query Language. It is a computer language designed to perform operations and handle data in a RDBMS. It does this by the use of SELECT, INSERT, UPDATE, DELETE and CREATE statements. As per current trends, SQL is the most common type of database language.

(EDB)(ConOne)(InfoTech)

### 4.1.2.2 MySQL

MySQL is a RDBMS, once owned by a Swedish company and now owned by Sun. MySQL is a very popular database solution for web applications. It is used by Internet sites such as Wikipedia, Facebook and Google. It is also part of the LAMP acronym, standing for Linux, Apache HTTP Server, MySQL and PHP, which is a very common technology composition for developing web applications (the LAMP philosophy is similar to what this thesis is trying to accomplish with Java EE). MySQL is free open source software.


### 4.1.2.3 PostgreSQL

PostgreSQL is a free open source RDBMS that is developed by the “open source community”. The system is very similar in style and functionality to MySQL [1].
4.1.2.4  JDBC  3

JDBC stands for Java Database Connectivity. It allows a Java application to connect, query and update a RDBMS using SQL commands. It is a common and simple technology for a Java application to connect to a database.

While it is easy to use, it is not well suited for big and complex database-heavy applications due to the fact that it requires large amounts of code and planning to achieve desirable functionality, such as Persistence.

4.1.2.5  XML  3

XML stands for Extensible Markup Language. It is a set of rules on how to encode documents in machine-readable form. XML is used in a wide variety of applications with many different purposes. Examples varies from configuration files in web servers or frameworks, transaction protocols such as SOAP or in XML markup languages such as XHTML.

4.1.2.6  JavaScript  3

JavaScript is a scripting language that is written inside HTML documents. Its intended use is client side scripting where the clients web browser interprets and executes the JavaScript code. Thus it can be used in the creation of highly interactive web pages that does not require updating the web page. Examples of use are validation of forms or even simple browser based games. Since JavaScript is written in the HTML document that is sent from the web server to the browser, all JavaScript is openly visible to the end user.

4.1.2.7  AJAX  1

AJAX stands for Asynchronous JavaScript and XML and is more of a technique using several other languages than a language of its own. The point of AJAX is to help create interactive web pages. It does this by retrieving small amounts of data from the web server and displaying it in the web browser without the user having to update the page or press any buttons. Example of the use of AJAX is auto-complete functions in text boxes or in the use of user registration forms where validation of information can be done while the user is typing the information.
AJAX works in the following manner: JavaScript uses the XMLHttpRequest object to make a request to the web server and then handles the return data. This is done asynchronously so the user is wholly unaware of the activity in the background.

(ConOne)

### 4.1.2.8 Web Services

Web services is a SOA (Service Oriented Architecture) technology. In essence Web services enables a unit of code to be accessed remotely via the web without the use of language specific protocols:

“Historically speaking, remote access to binary units required platform-specific and sometimes language-specific protocols. For example ... Enterprise JavaBeans (EJBs) requires a Remote Method Invocation (RMI) Protocol.

One can access Web services using nothing but HTTP. Of all the protocols in existence today, HTTP is the one specific wire protocol that all platforms tend to agree on. Thus, using Web services, a Web service developer can use any language he wish and a Web service consumer can use standard HTTP to invoke methods a Web service provides. The bottom line is that we have true language and platform integration.” – Roseindia.net

In Java, Web services can be implemented using Servlets, JSPs and EJBs. Many IDEs (Integrated Development Environment) incorporate specific functionality for creation of Web Services.

Security-wise there are interesting developments occurring within Web services technology. According to darkreading.com developers are suggesting a framework “Forensic Web Services” that can “recreate the scene of crime in the aftermath of a cyber attack.” [1]

(EDB)(ConOne)(HiQ)(Folksam)(Trade Doubler)

### 4.1.2.9 SOAP

SOAP is a protocol that uses XML and HTML as mechanisms to send information between applications without the dependency of OS or language specific protocols. It is used in Web services to transport data.

(ConOne)(Folksam)
4.1.2.10 MQ

Otherwise known as IBM WebSphere MQ (MQ stands for Message Queuing) is a family of network communication software products. MQ is a MOM (Message Oriented Middleware) and is used for messaging across multiple platforms such as Windows, Linux and Unix. MQ is a vital part of IBM’s SOA strategy for providing Web services.

“With over 50% market share, and the widest product platform coverage, MQSeries is the de facto standard for message oriented middleware.” – capitalware.biz

Note: The quotation is from an article published May 2010.

[Folksam]

4.1.3 Frameworks

Frameworks describes third party software designed for Java and Java EE.

4.1.3.1 Spring

Spring is a comprehensive framework that aims to simplify Java EE design patterns. Using Spring does not require you to know everything about the framework. You can start out with the core functions and then extend your application with Spring functions of your choice.

The most important features of Spring is Dependency Injection (DI) and Aspect Oriented Programming (AOP). Spring also covers integration to database frameworks, AOP transactions, Security, Web Services, EJB integration as well as a comprehensive MVC framework.

[Spring in action, Second Edition - Book]
(EDB)(ConOne)(InfoTech)(ValTech)(Exigen)(HiQ)(Trade Doubler)

4.1.3.2 Hibernate

Hibernate is a data-tier ORM Persistence framework that maps POJOs to database tables. It provides SQL data query and retrieval facilities. It uses XML-files or Java annotations to configure its functionality. Because of this Hibernate requires small amounts of code.

[http://www.hibernate.org/about/orm.html] 28/06 – 2010
(ConOne)(HiQ)(ValTech)(Exigen)(HiQ)(Trade Doubler)
### 4.1.3.3 JSF

JSF stands for Java Server Faces. It is a front-end MVC framework design to simplify the creation of server side user interfaces and MVC flow. It consists of a tag library that can be used instead of Java code inside JSP files. It saves the state of forms and auto-populates them between client requests and its connects events in the user interface with code in the server application.

(EDB)(ConOne)(Exigen)(HiQ)

### 4.1.3.4 Struts

Struts is a front-end MVC framework designed to simplify the development of web applications by enforcing the MVC design pattern. Struts provides a Controller called ActionServlet that forwards requests to appropriate model. The model return an “ActionForward”, a string telling the controller what view to send to the client and what content should be in that view. Struts also have a tag library so that the view easily can read and populate the view with the supplied content.

(InfoTech)(ValTech)(Exigen)(HiQ)(Trade Doubler)

### 4.1.3.5 Tiles

Tiles is a framework that until recently was a part of Struts, and is now known as Apache Tiles. Tiles is a framework that helps constructing a JSP page that is divided into tiles that can include other JSP pages.

[Spring in action, Second Edition - Book]

### 4.1.3.6 Ant

The decision to classify Ant as common is based on experience while researching this thesis. Ant stands for Another Neat Tool and is a automated build tool developed by Apache. It uses XML files to define build orders, i.e. how to clean, compile and package project in JAR files.

Ant has its drawbacks though as it is quite verbose and the build files (XML files) can easily become quite complex especially in larger applications and projects.

(Trade Doubler)

### 4.1.3.7 Maven

Maven is a build tool developed by Apache that sacrifices some generality and flexibility for less complex building scripts (comparing to Ant). Maven uses POM files (Project Object Model) to describe the application being built, the applications dependencies on external components/models and the build order.
Maven dynamically downloads Java libraries that are needed to build the project from different repositories.

(ConOne)(Folksam)(ValTech)(Trade Doubler)

### 4.1.3.8 Hudson

Hudson is a CI tool (Continuous Integration) that runs on a application server. It can run Ant and Maven build projects, execute shell scripts and Windows batch commands and it supports revision control systems like Subversion. Sun started to support Hudson in early 2009 but in 2010 the primary developer of Hudson, Kohsuke Kawaguchi started his own Company for commercial support of Hudson.

(ConOne)(HiQ)(Folksam)

### 4.1.3.9 JUnit

JUnit is a unit testing framework and is an important tool for TDD (Test Driven Development). It consists of JAR files that are imported into test classes. The JUnit test classes supports different assert functions as well as different function prefixes defining if a function is a test, what rules should apply to the test or if a function is a pre-test cleanup function among other things. JUnit can be used in TDD for testing many different aspects of web and software development.

[Test Driven – Book]
(EDB)(ConOne)(HiQ)(ValTech)(Paxport Systems)

### 4.1.3.10 Log4J

Log4j is a logging tool developed by Apache. Log4j enables a developer to turn logging on and off at run time, have a greater choice on what parts to log with inheritance in loggers and also it makes an effort to minimize the time to run the logging. The output can be configured to the local machine or a remote server.

[http://logging.apache.org/log4j/1.2/] 13/07 – 2010
(Paxport Systems)

### 4.2 Rare

This section describes the least common frameworks, technologies and problem areas based on the market study and author experience. See chapter 1: Introduction.

### 4.2.1 Problem areas

This section describes problem areas in programming in connection to web development.
4.2.1.1 CMS

CMS stands for Content Management System and is a system that helps manage the workflow of large numbers of people working on the same software development project.

(ConOne)

4.2.1.2 ESB

ESB stands for Enterprise Service Bus. ESB is a MOM (Message Oriented Middleware) technology that helps implementation of SOA (Service Oriented Architecture). In short, a ESB provides a event driven system that is able to deliver messages between different types of computer languages and technologies.

(EDB)

4.2.2 Technologies

This section describes components in the Java EE language as well as technologies that are disconnected from the Java language.

4.2.2.1 Swing

Swing is rare in the terms of web development (Author experience). Swing is a Java API used for creating visual interfaces.

(Paxport Systems)

4.2.2.2 JPA

JPA stands for Java Persistence API. It is a standard library in Java EE and is similar to the third party framework Hibernate.

(ConOne)(Paxport Systems)

4.2.2.3 RMI

RMI stands for Remote Method Invocation. It is a technology for creating distributed applications by allowing methods on one (Java virtual) machine to be called from another.

(Trade Doubler)
4.2.2.4 JMX 1

JMX stands for Java Management Extensions. JMX uses MBeans (Managed Beans) to manage and monitor applications, system objects, devices (like printers), and services. MBeans are created using DI (Dependency Injection). JMX consist of a 3-tier architecture; the actual MBeans (also called Probes), Core JMX (also called Agent) that acts as an intermediate between the Probes and the applications. And Connectors and Adapters (also called Remote Management level) that enables remote applications to access the MBeans.

(Trade Doubler)

4.2.2.5 XSLT 1

XSLT stands for Extensible Stylesheet Language Transformations. It is an XML-based language that transforms an XML documents and generates output into a different format such as HTML or some type of document that can be read by a browser.

[http://www.roseindia.net/xml/introduction-to-xsltAPI.shtml] 29/06 – 2010
(Folksam)

4.2.2.6 Hessian 1

Hessian is a Web service protocol (API) that enables you to implement a Web service in an easy manner (without the use of WSDL). Hessian has implementations in several languages. Creating a Web service consists of four steps: 1, Create an Java Interface as the Public API. 2, create a client using HessianProxyFactory. 3, create a Web service implementation. 4, configure the service in your servlet engine.

[http://hessian.caucho.com/] 13/07 – 2010
(Paxport Systems)

4.2.2.7 Selenium 1

Selenium is a Firefox plug-in and a desktop application designed for web application testing. It allows you to record what actions you perform in Firefox and the Selenium desktop application generates testing code in Java or other languages for those actions. ([1] is a quick time movie link explaining the concept.)

(ValTech)
4.2.2.8  JRuby  

JRuby is a Java implementation of the Ruby programming language. It also enables two-way access between Java and Ruby code. JRuby was supported by Sun between 2006-2009.

(ConOne)

4.2.2.9  SiteMesh  

SiteMesh is a layout decoration framework for large web sites aimed at creating a consistent look and feel.

At some point, every web application, no matter what language or concussio of frameworks being used, produces a HTML document that it sends to the client web browser. SiteMesh intercepts this document before being sent, parses the document, obtains properties and data, and generates a modified page that adheres to a predefined decorator. It does this for all HTML documents being sent and thus creates a unified look and feel though out the web site.

[http://www.opensymphony.com/sitemesh/] 14/07 – 2010
(Trade Doubler)

4.2.2.10  IText  

“iText is a ... library for developers looking to enhance web- and other applications with dynamic PDF document generation and/or manipulation.” - itextpdf.com/

(Paxport Systems)

4.2.2.11  jQuery  

“jQuery is a ... JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions ...” – jquery.com

(ConOne)

4.2.2.12  ExtJS  

Ext JavaScript is a JavaScript library that simplifies view development with AJAX, DHTML and DOM.

(Trade Doubler)

4.2.2.13  UDDI  

UDDI stands for Universal Description, Discovery and Integration and is a XML based registry for Web services. It was Developed jointly by Microsoft and IBM. UDDI can be seen as a phone book that enables businesses to list their Web services and for service customers to discover services.
4.2.2.14 WSDL

WSDL stands for Web Services Description Language. It is an integral part of UDDI. It uses XML to send and receive information about what methods, parameters, data types and return data that are defined by a Web service.

4.2.3 Frameworks

This section describes third party software designed for Java and Java EE as well as some non Java frameworks and libraries.

4.2.3.1 WebWork

WebWork is a front-end MVC framework that has merged into Struts 2 as of 2008. WebWork is based on the MVC, Command and Dispatcher patterns and its main goals is to achieve IoC (Inversion of Control).

4.2.3.2 Tapestry

Tapestry is a front end web development framework aimed to bring a higher degree of object orientation to web application development.

“In Tapestry, you create your application in terms of objects, and the methods and properties of those objects -- and specifically not in terms of URLs and query parameters.” – tapestry.apache.org/

4.2.3.3 FreeMarker

“FreeMarker is designed to be practical for the generation of HTML Web pages, particularly by servlet-based applications following the MVC ... pattern. ...

Java programs prepare the data to be displayed ... and FreeMarker just generates textual pages that display the prepared data using templates.” – freemarker.sourceforge.net/

(Trade Doubler)

4.2.3.4 Velocity

Velocity is a template language that references objects in Java code. Velocity is a MVC framework and aims to separate business logic from view. It is developed by Apache.

(Trade Doubler)
<table>
<thead>
<tr>
<th>Section</th>
<th>Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.3.5</td>
<td>RichFaces</td>
<td>RichFaces is a framework for easy AJAX integration with JSF applications. [<a href="http://jboss.org/richfaces">http://jboss.org/richfaces</a>] 13/07 - 2010 (ConOne)</td>
</tr>
<tr>
<td>4.2.3.6</td>
<td>Facelets</td>
<td>Facelets is a view handler framework for JSF that focuses on building JSF component trees and simplify JSF development. [<a href="https://facelets.dev.java.net/">https://facelets.dev.java.net/</a>] 13/07 – 2010 [<a href="http://en.wikipedia.org/wiki/Facelets">http://en.wikipedia.org/wiki/Facelets</a>] 13/07 – 2010 (ConOne)</td>
</tr>
<tr>
<td>4.2.3.7</td>
<td>Wicket</td>
<td>Wicket is a front end framework developed by Apache. The goal of wicket is to make it possible to develop web applications in only pure Java and pure HTML. In the process of doing this, it also creates a clear separation of the view from the logic. The only thing that connects the HTML to the Java logic is wicket ids which are used to identify HTML tags. Wicket is not a traditional MVC framework. [<a href="http://wicket.apache.org/">http://wicket.apache.org/</a>] 13/07 – 2010 [<a href="http://en.wikipedia.org/wiki/Apache_Wicket">http://en.wikipedia.org/wiki/Apache_Wicket</a>] 13/07 – 2010 (Paxport Systems)</td>
</tr>
<tr>
<td>4.2.3.8</td>
<td>JGoodies</td>
<td>JGoodies is a framework designed to assist in Java Swing application development. [<a href="http://www.jgoodies.com/">http://www.jgoodies.com/</a>] 13/07 – 2010 (Paxport Systems)</td>
</tr>
<tr>
<td>4.2.3.9</td>
<td>TestNG</td>
<td>TestNG is a testing framework very similar to JUnit. However this framework was developed because of certain frustrations with JUnit. Features includes a way to define tests to belong to different groups, e.g. fast or slow. This enables the developer to choose whether to run a subset of tests or all tests. TestNG is designed to cover all categories of tests, including unit, functional, and integration tests. [<a href="http://testng.org/doc/index.html">http://testng.org/doc/index.html</a>] 14/07 – 2010 (Paxport Systems)</td>
</tr>
</tbody>
</table>
| 4.2.3.10  | Jmeter    | “Apache JMeter is a ... Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web
Applications but has since expanded to other test functions.”

[jakarta.apache.org/jmeter/](http://jakarta.apache.org/jmeter/) (Trade Doubler)

<table>
<thead>
<tr>
<th>4.2.3.11</th>
<th>Hamcrest</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamcrest is a matcher framework. It is most often used in testing for assert functions where two different objects are matched. Hamcrest can be used in test frameworks such as JUnit and TestNG.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4.2.3.12</th>
<th>JAAS</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAAS stands for Java Authentication and Authorization Service. JAAS is a Java security framework embedded in the Java JRE.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4.2.3.13</th>
<th>Java ACC</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC stands for Authorization Contract for Containers. Java ACC defines new java.security.Permission classes to satisfy the Java EE authorization model.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4.2.3.14</th>
<th>Acegi/Spring Security</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acegi is the official security project for Spring and is now called Spring Security. According to springsource.org [1] is Spring Security used in many highly demanding environments such as government agencies, military applications and central banks.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4.2.3.15</th>
<th>SQL Developer</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Developer is developed by Oracle. SQL Developer is a free graphical tool for database development.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5 Servers

A “server” can refer to any hardware providing services used by other computers or by itself. Secondly and in this thesis more importantly, a server refers to a computer program that serves a client.
There is much debate over what classifies as an Application server, a Web server, A Servlet container etc. In this thesis I will try to be as thorough as possible in these classifications even though I encountered many discrepancies when researching the information.

### 5.1 Web servers

The primary function of a web server is to listen for HTTP requests from clients and deliver web pages in the form of HTML documents as well as any additional information such as images, style sheets and JavaScripts. Clients can also submit web forms and upload files with HTTP.

Many web servers also support server-side scripting with PHP. Usually, this functionality is used in combination with a database to create dynamic HTML.


#### 5.1.1 Apache HTTP Server

Apache HTTP Server is the world's most popular web server and as of 2010 it served 54% of all the world's websites and 66% of the million busiest. This is in large parts thanks to server-side programming language support with languages such as Perl, Python, Tcl and PHP.


### 5.2 Application servers

As opposed to a web server which exclusively deals with HTTP, HTML and scripting languages, an application server is used to expose business logic to clients. The difference is that the business logic is most commonly written in Java or .NET. This is done through various protocols including but not exclusively limited to HTTP. A common way for Java applications to expose the business logic is through the use of EJBs. This is not the general truth however as many frameworks has emerged to simplify and replace the use of EJBs.

Moreover, an application server handles security, transaction processing, recourse pooling and messaging.

[1] is a list of Java EE application servers.


#### 5.2.1 WebLogic

WebLogic is a application server developed by Oracle, also known as 11g. It supports Java EE and is also a web server. WebLogic is not free.

5.2.2 GlassFish

GlassFish is a application server developed by Sun for the Java EE platform. GlassFish is free.

[https://glassfish.dev.java.net/] 20/07 – 2010
(ConOne)(HiQ)(Exigen)(HiQ)

5.2.3 Oracle Application Server 10g

Oracle Application Server 10g is a combination of Oracle HTTP web server (based on Apache HTTP Server) and OC4J which is an Java EE container developed by Oracle. 10g supports SOA and is not free.

Note: The two companies using this technology only wrote “Oracle”. That they really meant 10g is an assumption made by the writer.

(ConOne)(InfoTech)

5.2.4 WebSphere

WebSphere is developed by IBM and is a Java EE and Web services application server. WebSphere is not free.

[http://www-01.ibm.com/software/webservers/appserv/was/] 20/07 – 2010
(Folksam)(Exigen)

5.2.5 JBoss

Jboss is a Java EE and Web services application server developed by Red Hat. JBoss is free.

[http://www.jboss.com/] 20/07 – 2010
(ConOne)(HiQ)(InfoTech)(Folksam)(ValTech)(Exigen)(HiQ)(Trade Doubler)

5.3 Servlet Containers

Servlet containers are simply applications which enables the use of Servlets and JSPs with or without a built in HTTP web server.

5.3.1 Tomcat

Tomcat, or Apache Tomcat, is a servlet and JSP container but it does not support EJB. It does not qualify as a full Java EE server. Tomcat is free.

(EDB)(ConOne)(Trade Doubler)
### 5.3.2 OC4J

OC4J is an abbreviation of “Oracle Containers for J2EE”. OC4J is an Java EE container. OC4J is a part of Oracle Application Server.

(Exigen)

### 5.4 Database Servers

A database server may refer to a computer program hosting a database or a computer dedicated to running such a program.

#### 5.4.1 DB2

DB2 is a relational database server developed by IBM. There are free and charged versions of DB2. The free one is called DB2 Express-C.

(ConOne)(Folksam)

### 5.5 Integration Servers

Integration servers are used to enable Continuous Integration (CI).

#### 5.5.1 Teamcity

TeamCity enables a team of developers to construct a test suite and continuously test their program updates for failures or flaws before actually committing the code. This programming methodology ensures that a commit does not produce an error or bug in the code base.

[http://www.jetbrains.com/teamcity/] 21/07 – 2010
(Trade Doubler)

### 5.6 Revision control systems

Revision control systems is used to enable several programmers to work on the same project and on the same files at the same time, thus optimizing the development time.


#### 5.6.1 Subversion

As of 2010 Subversion is a part of Apache software foundation.

(Folksam)(ValTech)(Trade Doubler)
6 Integrated Development Environments (IDEs)

IDEs are used for writing code and keeping track of files in a project. Features include auto complete and the ability to include plug-ins from different frameworks or tools.

6.1 Eclipse

Eclipse is a very popular IDE and there are many plug-ins available. Examples of Eclipse plug-ins include: Spring IDE, Tomcat for Eclipse, Hibernate tools and Apache Maven among hundreds of others. Eclipse is primarily meant for development in Java, but though various plug-ins is also supports languages like C++, Python, PHP and Ruby.

[http://www.eclipseplugincentral.com/] 21/07 – 2010
(ConOne)(HiQ)(Folksam)(Trade Doubler)

6.2 NetBeans

NetBeans is developed by Sun. Other than Java, NetBeans supports JavaScript, PHP, Python, Ruby and C++ among other languages.

(Paxport systems)

6.3 IntelliJ

IntelliJ is developed and sold for profit by JetBrains. There are two versions. The free one enables plain Java development with support for many popular testing frameworks. The other one has a fee and supports Java EE development with Spring and Hibernate among other frameworks.

[http://www.jetbrains.com/idea/] 22/07 – 2010
(Paxport systems)

7 Programming techniques

Here follows a list of programming techniques that has appeared in the research of this document that are all applicable (or essential) to web development.

• Agile Programming
• Extreme Programming
• SCRUM
• Continuous Integration
• Refactoring
• TDD – Test Driven Development
• BDD – Behavior Driven Development
The different Java archives files are generally used to distribute Java libraries or applications. JAR files build on the ZIP format.

### 8.1 JAR

JAR stands for Java Archive.


### 8.2 WAR

WAR stands for Web Application aRchive. Stores XML files, Java classes, JavaServer Pages and other objects for Web Applications.


### 8.3 RAR

RAR stands for Resource Adapter aRchive. Stores XML files, Java classes and other objects for J2EE Connector Architecture (JCA) applications.


### 8.4 EAR

EAR stands for Enterprise Archive. EAR files provide composite Java archives which combine XML files, Java classes and other objects including JAR, WAR and RAR Java archive files for Enterprise Applications.


### 8.5 SAR

SAR stands for Service Archive. SAR is similar to EAR. It provides a service.xml file and accompanying JAR files.
