Developing two web application modules in a Tomcat Enviroment

John Frizzell

Andrew Sweeney

B.Sc.(Hons) in Software Development

April 23, 2016

Final Year Project

Advised by: Dr John Healy

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract This project was contracted to us through Enterprise Ireland1[1] from a company called RoamPA. We have previously worked on a project for RoamPA in the past. RoamPA was building a web application aimed at business users and required two modules to be developed for them externally. One of the modules consists of generating reports for a user based on streamlined, friendly queries for user in a management position. This system would let the user generate queries about users or groups of users relating to their business activities. The other module consists of a reviews system which allows users to generate reviews about businesses,faculties, points of interest etc. The review system requires the ability to create, view, like and dislike reviews, all of which is presented to the user through a map. Both of these modules had to be created from scratch and had to integrate into their existing server environment whilst using their database technologies. There was a lot of requirements, frameworks and dependencies. This project was a great experience and because it required knowledge about technologies we had never used before so we were delighted to take this challenge.

Authors The authors of this project are Andrew Sweeney and John Frizzell. We are both graduates of the BSc in Computing in Software Development which was the preceding course to the current honours degree. We both have a great interest in programming and technology. We are constantly seeking ways to improve our knowledge and skills.

Chapter 1

Introduction

RoamPA Ltd is a start-up company founded by Una McNeill. They are developing a web application which is used to streamline the process of moving employees globally for work and business purposes. They wish to provide a much improved employee experience, significant cost savings to clients and greater return on investments with assignments and business travel.

We had previously worked for her on another project. The previous project had brought us outside the scope of our college work and proved to be a great learning experience. A year later We were asked again if they would develop two separate modules for this web application.

This time the focus of the project was completely different. This new project involved technologies and methodologies new to us. The focus was to create a reviews system where a user can view, create and rate reviews. The reviews would be then displayed an some form of map.

JOHNS BIT ABOUT REPORT GENERATOR

All of these technologies were outside of the scope of our college work and required a lot of research, experimentation and tutorials to complete the solution. Aside from programming in Java through our coursework and having some experience with MySQL databases we hadn't any previous experience with these technologies. Our course work covered some basic HTML and CSS in first year, so all of the skills to develop this application were learned as we went and through other experiences with web technologies.

Chapter 2

Context

2.1 Why we chose this project

Throughout our college course we have developed in many different languages. Java, C#, C and Python to name a few. We have not had the chance to really to dig deeply into web development. Web languages like PHP and Javascript were not part of our syllabus. Our sister course "Computing in Digital Media" does some of the design elements of web development but not much of the back-end stuff that we've both liked. Throughout our course we had a few semesters that had Database modules involving MYSQL. We heard the current project involved full blown MVC, Servlets, ORM and JSP pages. We also learned that the main language required for the project was Java which is one of the core languages for our college course. This project gave us a whole new area to experiment with and learn. Due to the challenge and interest we accepted the contract for this project.

2.2 Challenges we had to face

Straight from the beginning, the challenge was to understand all of the frameworks and how all of the different parts of the environment work together. This meant completely understanding concepts like ORM and how they are applied through the framework Hibernate. Other concepts included learning how MVC is applied through SpringMVC with routes etc. There was a lot to research, a lot of frameworks to read up on. There was also numerous dependencies to figure out and all of this had to be. The amount of research, tutorials and understanding to be put into the project was the challenge.

2.3 Objectives

- Follow the specifications provided to make integration easier
- Research the frameworks and environment
- Create a reports module
- Create a reviews module

2.4 Specifications

We were creating a two modules for RoamPA. Both modules had to be integrated onto RoamPA's existing Apache Tomcat server. The Frameworks required of us were to use Hibernate ORM and SpringMVC. The existing database RoamPA uses is MySQL. All of the code handed over to RoamPA must be integrable on their server.

The first module involved creating a reviews system. The reviews system should allow the customer to look up an area using a search box and using filters then find locations for buildings like hospitals,restaurants and numerous other points of interest. The customer could then click on the POI and create a review for a business. They can also rate a current review that exists on there and like or dislike it.

The second module

JOHNS BIT ABOUT REPORT GENERATOR

Chapter 3

Methodology

Consisting of a team of two we communicated weekly through Skype calls,reviewing over each others code. The process was incremental as we first primarily spending a lot of time researching and understanding the environment. We went through cycles of researching and going through tutorials on Hibernate and then later on tutorials for SpringMVC. This helped reinforce our understanding of how everything worked. At times we would meet up to do some paired programming, this proved as a great experience as we were able to help each other understanding the project a lot quicker. The was numerous cycles of experimenting with Hibernate, trying pass information through it to the database and gradually building up knowledge from there.

We used GitHub as a repository for our code. This is how the code you later be passed over to the developers at RoamPA. Using GitHub enabled us to download each others work and test locally. GitHub greatly helped our work-flow enabling us to share code between us in minutes remotely from home.

We were restrained to using technologies that would only work on an Apache Tomcat server. This means programming using the following

- Programming all of the server backend in Java
- Designing all of the webpages using HTML/CSS Javascript and JQuery, these then were later fitted into JSP pages
- Keeping all of the database queries to conform with MySQL
- Using Hibernate to facilitate the passage of data too and from the database
- Using SpringMVC to manage the structure of the project

This meant the only real choice in technologies to use we had was deciding on what map API to use.

Google Maps was always the first consideration for what mapping API would use. The most well known and popular choice may not always be the best solution to the problem. This lead us to experimenting with several other map API's. Alternatives involved looking at Bing Maps, OpenStreetMap and Leaflet. Even with all of the choice Google Maps still proved to be the best selection. There were varying reasons for not using tha alternatives primarily because of terms of contract, the cost and the map API's being relatively new left them open to change in future. Google Maps has been around 11 years and is constantly updated. Google Maps API allowed querying exactly what we wanted. Google Places is part of the Google Maps API and allows for querying using different type filters. An example of some of the type filters are airports,libraries, department stores, restaurants etc. You can pass in a location, a filter and a radius to the API and have all businesses of that type returned back in JSON format.

JOHNS BIT ABOUT Methodology

Chapter 4

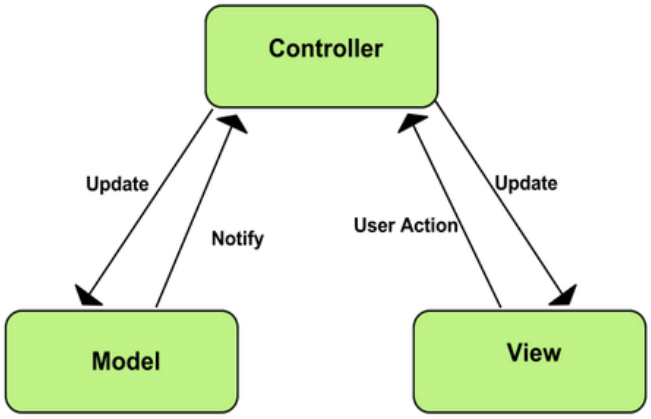
Technology Review Breakdown

- Spring MVC and MVC
- Hibernate ORM and MySQL
- HTML and CSS, Javascript/jQuery
- JSP Pages

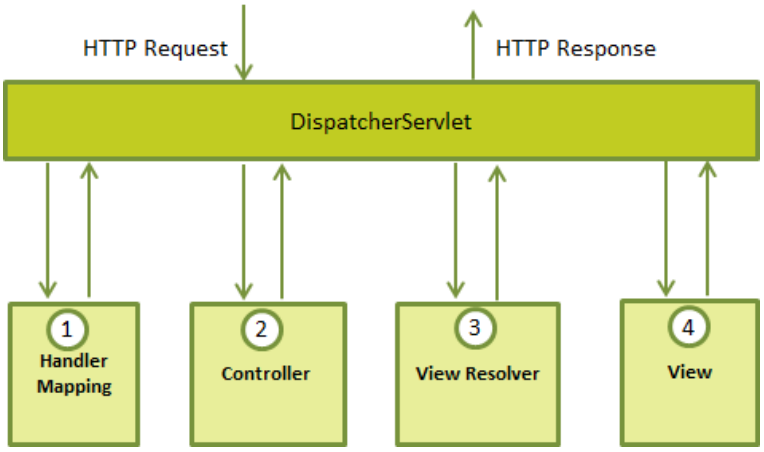
4.1 Spring MVC and MVC

SpringMVC means Spring Model-View-Controller. SpringMVC is the framework we had to conform to. Quite a lot of time was spent learning how to use Spring along with understanding Hibernate ORM.

Model View Controller is a design pattern which consists of a Model, a View and a Controller. MVC is a widely used design pattern by web application frameworks. A user makes a request to the server through a controller and a view is served back. Traditionally the controller will take the request and then manage gathering all of the information from the database and build up a View. Once the information is gathered the controller finally serves the completed view to the user. This view is typically a mix of static and dynamic information based of what's currently in the database. This is a common design pattern for web applications. It scales well and deals with pushing the right information to the user based on the parameters passed to the controller. A request comes in from the user and a response is passed back to the user in the form of a response or a view. Shown in the image below.



SpringMVC has it's own slightly altered version of this. It works off the same principals just with a more indept controller.



After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller.

The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.

The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.

Once the view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

It was very important to the project to understand this MVC concept.

4.2 Hibernate ORM

Hibernate ORM is referred usually too as Hibernate for short. Hibernate is an Object Relational Mapping framework which is what ORM stands for. ORM is a technique for mapping objects, in our case, mapping instances of Java classes and relating them to tables and columns in our chosen database. There instances of Java classes are known as POJO's or plain ordinary Java objects. Mapping of the objects to columns and tables in Hibernate is done through XML files or more recently through annotations. Hibernate XML files can be generated through Eclipse but if column names differ from field names you will find yourself spending a lot of time manually editing these files. This leads a lot of margin for error and leads to a lot of overhead. E.g Manually assigning primary keys, fields and making sure relations are properly defined.

Below is an example of this

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

Annotations, which were brought in by newer versions of Hibernate scraps the ideas of these XML files and makes things alot easier. Instead of mapping out every member variable for a class you can set up mapping now with a few simple annotations

```
import javax.persistence.*;

@Entity
@Table(name = "EMPLOYEE")
public class Employee {
   @Id @GeneratedValue
   @Column(name = "id")
   private int id;

   @Column(name = "first_name")
   private String firstName;

   @Column(name = "last_name")
   private String lastName;

   @Column(name = "salary")
   private int salary;

   public Employee() {}
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

Hibernate also has a configuration file which must be set up. This holds connection information for the database and other settings.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
   </property>

   <!-- Assume test is the database name -->
   <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
   </property>
   <property name="hibernate.connection.username">
      root
   </property>
   <property name="hibernate.connection.password">
      root123
   </property>

   <!-- List of XML mapping files -->
   <mapping resource="Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

Originally experimenting and learning was done using version 3 of Hibernate and used Hibernates Object Relational Mapping using XML files. These files are in typical XML format. To use this method of mapping must define the mapped files in the hibernate configuration file, if not when you try to commit the object to the database an exception is thrown. Furthermore any errors in the XML or mistypes in the mapping will stop any operations using this object.

We were successful with implementing some basic CRUD methods (Create,Read,Update and Delete). This required alot of manual configuration

with the XML files. The Eclipse IDE which we used allows you to generate these hbm.xml files but they were not always correct. You must take into account that when generating these xml mapping files Eclipse has no knowledge of the database. If for whatever reason you have configured your column names in the database differently you will have configure them manually for every POJO with mapping in your project. Due to this it was best to keep field names in the POJO classes the same in both the project and the database. Hibernate will also try to generate keys and guess which field is key in the POJO or not even find the key at all. Usually the autogeneration just looked for a field called "id". Due to this you'll have to manually configure the id and it's relationship. Another major overhead of autogenerating these files is having to generate a mapping file for each class dependant on the current class you are generating the mapping for. The more relations an object has, the more files you have to configure and this was very tedious. The margin for error was high, autogeneration lead to mistakes and overall it was very rigid. This lead to very slow workflow because the IDE wouldn't return the location of errors in these files and would leave you guessing alot of the time. This experience stood to us as we now knew how to commit the objects to the database correctly. Once we had information going into the database and coming back we went back into another research phase.

Upon researching more we found alot of information about using annotations with Hibernate and that in recent version they had removed away from the traditional method of mapping using the hbm.xml files. In the case where future changes might be made to the work we have worked on it was best to research annotations more as it seemed it could speed up development and remove the need for those mapping files. We moved up a version of Hibernate and discovered Hibernate JPA Annotations. While we succeeded in getting this XML mapping working Hibernate Annotations drastically changed the workflow. Annotations removed the need for hbm.xml files and you could simply add simple annotations to the POJO class.

Creating the actual database from these classes had to be done manually. Hibernate does have the capacity to do this automatically based off the current classes but this lead to issues. The particular component was "hibernate.hbm2ddl.auto" this is called in the spring.xml file which serves as configuration file for server connection strings and to name the annotated or mapped classes. The reason this wasn't used was because it would force the generation of tables for classes we didn't use and forced the generation of tables for interfaces strangely. Some of these tables weren't needed so the tables that Hibernate generated for us were used as a guide more than anything. Using Hibernate we also found that Hibernate would look for those interface tables, this really complicated things, we only required a minimal

amount of tables in the database. Due to this the decision was taken to ignore some of the composition with the POJO's and manage them manually. So in our case the class DefaultOrientationService held a set of Reviews in a ReviewSet. It was much cleaner to assign a review set id to a review and store the reviews individually that way. This solution removed the need for all these interval tables in the database. The idea was to keep our footprint on the database as light as possible. In the end we reduced the amount of tables needed in the database down too four relatively simple tables.

4.3 HTML and CSS, Javascript/jQuery

We didn't use much HTML and CSS because the sidebars and navigation bars around the content would later be updated to the style created by RoamPA's team. A simple Bootstrap was used as a template and was later removed. All of the functionality of the map was achieved using Google Maps Places API. This was all done through Javascript. Anything that needed to be done by the backend was sent through to the proper MVC routes on the server controller by using Javascript.

Chapter 5

System Design

Chapter 6

System Evaluation

As many pages as needed.

- Prove that your software is robust. How? Testing etc.
- Use performance benchmarks (space and time) if algorithmic.
- Measure the outcomes / outputs of your system / software against the objectives from the Introduction.
- Highlight any limitations or opportuni-ties in your approach or technologies used.

Chapter 7

Conclusion

About three pages.

- Briefly summarise your context and ob-jectives (a few lines).
- Highlight your findings from the evalua-tion section / chapter and any opportuni-ties identified.

Bibliography

[1] https://www.enterprise-ireland.com/en/

[2] https://www.tripadvisor.ie/

[3] X

[4] X

[5] X

Bibliography

[1] A. Einstein, "Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]," *Annalen der Physik*, vol. 322, no. 10, pp. 891–921, 1905.