Hibernate 'til Spring
Benefits of Spring MVC, Hibernate and Struts for the Development of a Web Application

Chris O'Brien

March 7, 2014

Abstract

Web development is one of the fastest growing areas in software development, with new tools being developed yearly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Introduction

This project concerns the development of a web application using a web framework in conjunction with a number of other tools. Throughout development, there is a particular cognisance towards the support of Non-Functional Requirements [NFRs] by both the web framework and the supporting tools throughout the development process.

1.1.1 General Introduction

The main goal of this project is to reflectively analyse a WAF [Web Application Framework], and architecture stack, in the creation of a website. This will be analysed in respect to both functional and non-functional requirements. Two key requirements are extensibility and maintenance. Extensibility refers to the ability of the framework to allow added functionality to the web application without having to modify the core workings of the application. Maintenance refers to the upkeep of the code, and facilitates the modification of the source code after the product is deployed. This may be to correct faults, improve attributes such as performance and security. The creative driver of the project is the development of a website to meet the requirements and needs of Monaleen Tennis Club, for both members of the club and of the committee. These needs will overlap as all committee representatives are all club members, but not all members are on the committee. From this, it was important to identify the precise requirements for each type of user. The main focus of this project was for the club to be able to perform their core functions through the website. This extended to the registration of members, a timetable for the courts, the creation and distribution of tournament schedules, the organisation and timetabling of training sessions, a method to contact all members and a news section to update and advise members of changes and upcoming events .

- Member Management
- Timetable Management
- Tournament Management

1.2 Objectives

1.3 Scope

1.4 Methodoloy

The methodology chosen as the foundation for this project is the Russo and Graham (1998) design methodology. It focuses on 9 iterative steps, each with feedback loops. The steps are outlined below

- Identification of the problem
- Analysis
- Design of the Application
- Resource Gathering
- Coding
- Testing
- Implementation
- Post Implementation Review and Maintainance

Other methodologies that were examined such as Balasubramanin and Bashian (1997), Siegel (1997), Iskawitz et al (1995) and Cranford-Teague (1998). The pros and cons of these methodologies were examined by Howcroft and Carroll (Howcroft and Carroll 2000), and after an examination of their findings, the Russo and Graham methodology best suited the nature and scale of this project. While the other methodologies are strong, they are geared towards large scale web development projects, or towards document-centred websites, and would not suit this project.(Howcroft and Carroll 2000) Using these as a guide, the following methodology was established.

- Identification of the problem
- Structured Literature Review

- Statement of the FYP Objectives
- Design of the Test Suite
- Development of the Prototype

  Analysis

  Design of the Application

  Resource Gathering

  Design Review

  Coding

  Testing

  Implementation

  Post Implemetation Review and Maintainance
- Emperical Study
- Critical Evaluation of the Results

1.5 Overview of Report

1.6 Motivation

The motivation behind this project for me was to examine, understand and work with software frameworks and methodologies that would be commonly used in industry, and to develop a software application from them. The module, Distributed Systems, touched on some of the tools and technologies, Netbeans and EJB respectively, used in relation to Java Enterprise development, and this formed the foundation of my interest in the area. I felt the FYP was a perfect vehicle to supplement my knowledge of this subject, with particular attention being paid to popular and in demand technologies.

Chapter 2

Background

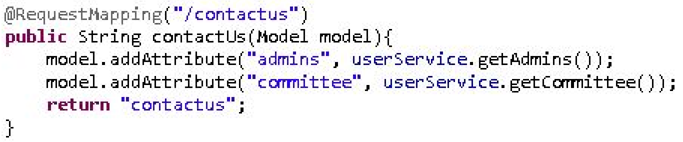
2.1 Architectures of Web Applications

2.2 Technologies

There are a number of components needs to build the architecture of a web application. The nature of these components is explored below, and their contribution to the creation of a web application is analysed.

2.2.1 Web Application Framework

The WAF chosen for this project is Spring MVC [Model View Controller]. Shan and Hua defined a WAF as a defined support structure in which other software applications can be organized and developed. (Shan and Hua 2006). Model-View-Controller is a software pattern that facilitates the use of a user interface. The Model manages the behaviour and data of the application. The View will manage the information obtained from the model and display it to the user. The Controller takes user input, such as key strokes, mouse movements or a touch display, and can interact and invoke functionality within the Model and/or View.



```
@RequestMapping("/contactus")
public String contactUs(Model model){
    model.addAttribute("admins", userService.getAdmins());
    model.addAttribute("committee", userService.getCommittee());
    return "contactus";
}
```

Figure 2.1: Contoller adding Model to View

2.2.2 Application Server

2.2.3 Project Management Tool

2.2.4 Database Model

2.2.5 Source Control

2.2.6 Integrated Development Environment

2.2.7 Logging

2.2.8 Web Page Creation

2.3 Software Engineering

2.3.1 Requirements

2.3.2 Design

2.3.3 Testing

2.3.4 Software Quality

Chapter 3

Requirements

Chapter 4

Design

Chapter 5

Implementation and Testing

5.1 Implemention

5.1.1 Spring

In order to begin implementation with the Spring MVC framework, there are a number of configuration files that are necessary. The core file is the *web.xml* file. This file is reponsible for the configuration for the framework. One of the key responsibilities is the definition of the context xml files, whose purpose will be elaborated on later. Different development profiles can be configured within this file in order to produce different development environments, such as production and testing environments.

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>
       classpath:beans/dao-context.xml
       classpath:beans/service-context.xml
       classpath:beans/security-context.xml
</param-value>
 </context-param>
```

Of particular importance are the definition of the context parameters. In this project, there were three main context files.

- Data Access Object Context
- Service Context

- Security Context

The DAO Context file specifies the packages that contain the various DAO classes within the application. It also contains configurations for both the database connection details, and Hibernate configurations. Packages containing entity classes for Hibernate are specified within this context also.

```
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
</props>
</property>
```

The Service Context file is responsible for specifying the base package containing the Service classes necessary to facilitate the collaboration between the Controller classes and the DAO classes. This file specifies that annotations will be used to configure the Service classes.

```
<context:annotation-config></context:annotation-config>
<context:component-scan base-package="service"></context:component-scan>
```

The Security Context file is the larger of the three files, and is responsible for the security configuration of the web application. There are four main areas within the file that were used to configure the web application created in this project.
The User Service aspect of the configuration file is responsible for retrieving users and their authority within the scope of the web application.
The URL access configuration ensures that only users who are authorised to access certain portions of the site are allowed access.
The Security Annotations allow the creation of an extra level of security into an application. At class level, annotations can be placed on methods to further ensure that proper access is enforced throughout the application.
Lastly, the Security Context is responsible for creating the password encoder bean in which passwords are encoded, and decoded, upon account creation and login. This ensures that no passwords in plain text form are ever stored on either the server or the database within the web application

- User Service

```
<security:authentication-manager>
       <security:authentication-provider>
       <security:jdbc-user-service data-source-ref="dataSource"
              id="jdbcUserService" authorities-by-username-query="select
                  username, authority from users where binary username = ?" />
       <security:password-encoder
           ref="passwordEncoder"></security:password-encoder>
       </security:authentication-provider>
</security:authentication-manager>
```

- URL Access

```
<security:intercept-url pattern="/timetable" access="permitAll"/>
<security:intercept-url pattern="/reportNoShow" access="permitAll"/>
<security:intercept-url pattern="/admin" access="hasRole(’ROLE_ADMIN’)"/>
<security:intercept-url pattern="/approveMembers"
    access="hasRole(’ROLE_ADMIN’)"/>
```

- Security Annotation for Service Class

```
<security:global-method-security
    secured-annotations="enabled"></security:global-method-security>
//Java Code from TimetableService class.
//This code is invoked when booking a slot on the timetable and is only
    accessible by registered members.
@Secured({"ROLE_ADMIN", "ROLE_MEMBER", "ROLE_COMMITTEE", "ROLE_WARNING",
    "ROLE_SUSPEND"})
       public void update(Timetable t){
              timetableDAO.updateTimetable(t);
       }
```

- Password Encoding

```
<bean id="passwordEncoder"
class="org.springframework.security.crypto.password.StandardPasswordEncoder">
</bean>
```

5.1.2 Security

A core part of the Spring platform is the Security support. Security is an important aspect for any application, but more-so for one that stores user data, particularly sensitive data such as names, addresses, phone numbers, and payment details. While there is no payment infrastructure within this application, there is scope for a system to be implemented, and it could certainly be a future requirement.

Spring handles security a number of ways. Firstly, it uses an *authority* hierarchy to separate different levels of users. For this web application, there were three main levels of authority, with one level containing three different branches.

- ROLE ADMIN
  - This refers to the main administration group. The group retains full rights across the web application
- ROLE COMMITTEE
  - This refers to the committee, as defined by the club themselves. This group with have the ability to perform some administrator privileges, but only those directly related to club activities, not site activities.
- ROLE MEMBER
  - The default user state. This group can perform actions such as booking slots in a timetable, registering for a tournament, and will have access to parts of the site unavailable to non-registered users.
- ROLE WARNING
  - A restriction placed upon a member. For example, a member who books time slots, but does not attend.
- ROLE SUSPEND
  - A further restriction placed upon a member.

To ensure that the correct user is logged in, the framework provides a SecurityContextHolder class which can be used by the Controllers in order to ensure that that any actions performed by the system are attributed to the correct user. In this regard, it was important that the system ensures that the duplication of a user-name is restricted. In this project, the user table had two keys: an integer id, which was the primary key, and the user-name was an email address. The validity of the email address was enforced at class level with the use on annotations on the user-name attribute within the User class.

```
//Excerpt from the User class
@NotNull(groups={PersistenceValidationGroup.class, FormValidationGroup.class})
@Pattern(regexp=".+\\@.+\\..+", message="This does not appear to be a valid email
    address", groups={PersistenceValidationGroup.class, FormValidationGroup.class})
@Column(name="username")
String username;
```

Another aspect of the Spring Security platform was Form Validation. When registering a user, there are a number a validation constraints that are placed upon the User class. Spring provides a facility to ensure these constraints are enforced, and to also provide a positive user experience. It does this through the use of a BindingResult object. This object holds a record of any errors from the form that the user populates. The controller that deals with the form will check the BindResult object for errors, and can respond appropriately. In order for this to work, both the Controller and the form need to be defined clearly. The form needs to be creating using the Spring Framework form tag library, and errors needs to be specified for each input within the form. These inputs, when used with Hibernate, all need to match with the attribute names given in the class they represent. This example references the User class.

```
<!-- Excerpt from the User registration form. Formatting removed for clarity --!>
<sf:form id="details" method="post"
    action="${pageContext.request.contextPath}/register" commandName="member">
Name<<sf:input name = "name" path="name" type="text"/>
<sf:errors path="name" cssClass="error"></sf:errors>
Password<sf:input id="password" name = "password" path="password" type="password"/>
<sf:errors path="password" cssClass="error"></sf:errors>
</sf:form>

//Method from the MembersController class
//This method is responsible for validating the form that users complete to
    register.
@RequestMapping(value = "/register", method = RequestMethod.POST)
public String doRegister(Model model,
@Validated(FormValidationGroup.class) @ModelAttribute("member") User member,
    BindingResult result) {
if (result.hasErrors()) {
        return "createmembers"; // if the result has errors, go back to create page
}
if (userService.exists(member.getUsername())) {
        result.rejectValue("username", "Duplicate Key",
        "This email address has already been used");
        return "createmembers";
        //if the email address already exists, return with this message.
}
        else {
                try {
                        member.setAuthority("ROLE_MEMBER");
                        userService.create(member);
                        return "registerSuccess";
                        //successful creation of member
                        } catch (Exception e) {
                                return "error";
                        }
                }
}
```

The UserDAO class then encodes the password using the already configured Password Encoder bean prior to saving it to the database. It was this process that introduced an issue with the validation process. When defining the constraints placed upon the password attribute within the User class, the encoding of the password at a later stage was not taken into account. When encoded, the length of the password increased well beyond the scope of the initial constraint. When Hibernate attempted to persist the User object, it found that the password was now in violation of the constraints placed upon this attribute within the User class. This issue did not occur in the application when using a JDBC database, and is a result of the close links that Hibernate has with its entity classes. In order to resolve this issue, Validation Groups were introduced. A Validation Group is a mechanism in which a class is used to define the validity of attributes within a class. Using these groups, the web application can enforce validation on attributes depending on the validation group specified in the Controller. As respects the application developed for this project, the constraint for the password only had to be enforced within the scope of the form. Once the form validation had passed, enforcement of the constraint was not a concern. By using a separate group for Hibernate persistence, the issue was resolved. (This needs revising! Bit wobbly and lacks structure, also previous code excerpt shows this in action also. Move up to there?)

The Security Context XML file is key to maintaining the integrity of the application security. This file defines access rights to the URL mappings within the application, and also enforces security for the service class methods. By default, the configuration denies access to all parts of the application, and access rights must be explicitly stated. This system relies on the developers to ensure that proper testing is completed to ensure that all access works as designed. This is a better solution than allowing access to all the site and restricting certain parts, such as the administrator panel, however.

```
<security:http use-expressions="true">
        <security:intercept-url pattern="/static/**" access="permitAll" />
        <security:intercept-url pattern="/images/**" access="permitAll" />
        <security:intercept-url pattern="/createmembers" access="permitAll" />
        <security:intercept-url pattern="/approveMembers"
            access="hasRole('ROLE_ADMIN')" />
        <security:intercept-url pattern="/tournamentRegister"
            access="isAuthenticated()" />
</security:http>
```

This code fragment from the *Security-Context.xml* file displays how access for the web application is configured. While there are five roles within the system, it is not necessary to explicitly define what actions a role can perform. For example, all registered users should be able to register for an existing tournament. In this case, once the framework detects that a user is authenticated, that is sufficient to allow access to that part of the application. Other examples are the display of static images, such as banners and advertising. These should be visible to all visitors to the site, regardless of their authentication status. Accordingly, this area of the application is set to permitAll allowing such access.

Chapter 6

Software Quality

Chapter 7

Evaluation

Chapter 8

Conclusions

Bibliography