STUDY OF OPEN SOURCE FRAMEWORKS IN JEE – SPRING AND HIBERNATE-ORM FRAMEWORKS

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Srinivas Kola
Fall 2012
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Srinivas Kola:

Study of Open Source Frameworks in JEE – Spring and

Hibernate-ORM Frameworks

Joseph Lewis, Chair
Department of Computer Science

Carl Eckberg
Department of Computer Science

Peter Blomgren
Department of Mathematics and Statistics

8/16/2012
Approval Date
ABSTRACT OF THE THESIS

Study of Open Source Frameworks in JEE – Spring and Hibernate-ORM Frameworks
by
Srinivas Kola
Master of Science in Computer Science
San Diego State University, 2012

The demand for web applications increased exponentially in the mid-1990s. With the emergence of object oriented programming languages, distributed web applications were built with the former client-server model that simultaneously solved the purpose of many online applications. Since then, Java has become dominant in the world of enterprise application development. This dominance of Java language is due to open source frameworks and many third party communities. In recent years, Spring and Hibernate-ORM (Object Relational Mapping) are the two most influential and famous open source frameworks, which stood out and are contributing an enormous success in the development of web applications. This research will study the advantage of combining both Spring and Hibernate and, how it affects the performance, programmability, cross cutting concerns and portability of an enterprise web application. Along with that, this research will particularly focus on Spring MVC (model, view, control) – an open-source web application framework, Spring AOP (Aspect Oriented Programming) for handling cross cutting concerns and analyze the advantages of using Hibernate over traditional JDBC (Java Database Connectivity) in Spring projects. To really understand how it is possible, a web application with layered architecture is built for managing computer science past thesis defense documents called TMS (Thesis Management System) using Hibernate and Spring frameworks.
TABLE OF CONTENTS

ABSTRACT ............................................................................................................................. iv
LIST OF TABLES ................................................................................................................... ix
LIST OF FIGURES ................................................................................................................ x
CHAPTER
1 INTRODUCTION .........................................................................................................1
   1.1 Java Enterprise Applications ..............................................................................1
   1.2 Spring and Hibernate Frameworks ....................................................................1
2 OBJECT RELATIONAL MAPPING ............................................................................4
   2.1 The Relational Problem .....................................................................................4
   2.2 Pure JDBC Mapping ..........................................................................................4
      2.2.1 Boilerplate Coding ................................................................................... 4
      2.2.2 General Interaction with RDBMS ........................................................... 5
      2.2.3 Data Specific SQL Queries ...................................................................... 5
   2.3 What is ORM? ...................................................................................................8
      2.3.1 The Cause – Model Mismatch ................................................................ 8
      2.3.2 Need of ORM ........................................................................................... 9
      2.3.3 ORM Design – Domain Model ................................................................. 9
      2.3.4 How ORM Managed It? ........................................................................... 9
         2.3.4.1 Development Productivity ............................................................ 10
         2.3.4.2 Code Maintainability ................................................................. 10
         2.3.4.3 Vendor Independence ............................................................... 10
      2.3.5 Criteria for Choosing an ORM Tool ...................................................... 10
         2.3.5.1 Technical Criteria ........................................................................ 10
         2.3.5.2 Non-Technical Criteria ................................................................. 12
3 HIBERNATE-ORM FRAMEWORK ............................................................................13
   3.1 Hibernate Revolutions .....................................................................................13
      3.1.1 Top Level Hibernate Workflow .................................................................. 13
3.1.2 How Hibernate Works Internally .............................................................. 14
3.1.3 The Hibernate Core ............................................................................... 14
3.1.4 Hibernate EntityManager ................................................................. 15
3.2 Hibernate Basic Architecture .................................................................... 15
3.3 Connection Pooling .................................................................................. 18
3.4 Hibernate Object States ........................................................................... 18
3.5 Mapping File Structure by Annotations in Hibernate .............................. 18
3.6 Hibernate Configuration ........................................................................... 20
3.7 Hibernate Query Languages .................................................................... 20
3.8 Criteria API in Hibernate .................................................................... 21
  3.8.1 Advantages of Hibernate ................................................................ 23
  3.8.2 Disadvantages of Hibernate .......................................................... 24
4 SPRING FRAMEWORK ................................................................................. 25
  4.1 Before Spring Revolution .................................................................... 25
  4.2 History of Spring Framework ............................................................... 26
  4.3 Spring Framework ............................................................................... 26
  4.4 Spring Modules .................................................................................. 27
  4.5 Spring 3.0 Features and Enhancements ................................................. 27
  4.6 The Spring MVC 3.0 Architecture ....................................................... 28
    4.6.1 Basic Spring MVC Workflow ....................................................... 28
    4.6.2 Client Request Processing Workflow in Spring MVC ..................... 30
      4.6.2.1 The DispatcherServlet ....................................................... 30
      4.6.2.2 Handler Mapping ............................................................... 32
      4.6.2.3 Controllers ......................................................................... 32
      4.6.2.4 ModelAndView ................................................................. 33
      4.6.2.5 The ViewResolver ........................................................... 34
      4.6.2.6 Views .............................................................................. 34
  4.7 Spring AOP ....................................................................................... 34
  4.8 Spring Mail ....................................................................................... 36
  4.9 Spring IoC Container ........................................................................... 37
5 DESIGN AND IMPLEMENTATION OF TMS SYSTEM ................................ 39
  5.1 Overview ......................................................................................... 39
5.2 TMS Application Architecture ................................................................. 39
5.3 TMS Functional Specifications ................................................................. 40
5.4 Use Case for TMS Application ................................................................. 42
5.5 TMS Data Model Design ........................................................................ 42
5.6 Sequence Diagrams for TMS Application Operations ......................... 42
5.7 TMS Application Flow and Configurations ........................................... 42
  5.7.1 Step 1 ................................................................................................. 43
    5.7.1.1 Spring MVC Configuration ......................................................... 43
    5.7.1.2 Spring Application Context ......................................................... 44
    5.7.1.3 Spring MVC Annotation-Based Configuration .......................... 48
  5.7.2 Step 2 ................................................................................................. 48
    5.7.2.1 Controller Layer ................................................................. 49
    5.7.2.2 Class Diagram ............................................................................. 49
    5.7.2.3 Annotations Used in the Controller Class .................................... 51
  5.7.3 Step 3 ................................................................................................. 52
    5.7.3.1 Service Layer ............................................................................. 52
    5.7.3.2 Class Diagram ............................................................................. 52
    5.7.3.3 Annotations Used in the Service Layer ....................................... 53
  5.7.4 Step 4 ................................................................................................. 55
    5.7.4.1 DAO Layer ................................................................................. 55
    5.7.4.2 Class Diagram ............................................................................. 55
    5.7.4.3 Annotations Used in Hibernate DAO ('studentDetailsHibernateDaoImpl') ........................................................ 56
    5.7.4.4 Hibernate with Spring Configuration ....................................... 58
    5.7.4.5 JDBC with Spring Configuration ................................................ 62
  5.7.5 Step 5 ................................................................................................. 62
    5.7.5.1 Package Contents ..................................................................... 63
    5.7.5.2 Class Diagram ............................................................................. 63
    5.7.5.3 How Spring AOP Dealt the Cross Cutting Concern in TMS ...... 65
  5.8 Step 6: The Presentation Layer (The View) .......................................... 67
    5.8.1 JSP as the User Interface ................................................................. 68
    5.8.2 ExtJs as the User Interface .............................................................. 73
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1. Properties Used in Criterion Restrictions</td>
<td>23</td>
</tr>
<tr>
<td>Table 5.1. Technology Stack Used in TMS Application</td>
<td>41</td>
</tr>
<tr>
<td>Table 5.2. Difference between Hibernate and JDBC</td>
<td>76</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1. North America job trends in front-end frameworks ................................................2
Figure 1.2. North America job trends in back-end frameworks ................................................3
Figure 2.1. Code snippet showing JDBC hard coded SQL statements to insert, delete, update, and search the records in the database, implemented in TMS application .................................................................6
Figure 2.2. Code snippet showing boilerplate coding in JDBC, showing how to open a connection-to send SQL statements to get the result set, implemented in TMS application ..............................................................................6
Figure 2.3. Code snippet showing JDBC code implemented in TMS application to map the Java model objects with relational data ..........................................................7
Figure 2.4. Code snippet for showing result set handling in TMS application .................................7
Figure 2.5. Mapping between object models and relational database ........................................9
Figure 3.1. High level view of Hibernate architecture ..................................................................16
Figure 3.2. View of runtime architecture .....................................................................................16
Figure 3.3. The full architecture with connections and transaction management .........................17
Figure 3.4. Code snippet showing Java model (entity class) mapping using Hibernate annotations, used in TMS application ...........................................................................19
Figure 3.5. Code snippet showing Hibernate configuration for connection management, connection pooling and transaction management, used in TMS application ...................................................................21
Figure 3.6. Some of the examples for HQL queries .......................................................................22
Figure 3.7. Criteria used in TMS application for searching the record by ‘red-id’ or by ‘advisor name.’ .................................................................................................................................22
Figure 4.1. Overview of the Spring framework ..............................................................................27
Figure 4.2. Spring MVC high level architecture diagram ............................................................29
Figure 4.3. Request processing workflow in Spring MVC ............................................................29
Figure 4.4. web.xml configuration for invoking DispatcherServlet ............................................30
Figure 4.5. Context hierarchy in Spring web MVC .......................................................................31
Figure 4.6. Configuration to declare multiple application context files with <context-param> .................................................................31
Figure 4.7. Configuration to map a particular URL with `@RequestMapping` Spring MVC annotation

Figure 4.8. Controller configuration with `@Controller` Spring annotation, it tells the container that 'StudentThesisController' is a controller class.

Figure 4.9. Code snippet showing ModelAndView given to DispatcherServlet, it then resolves the logical view ('showContacts') with a particular view technology ('showContacts.jsp').

Figure 4.10. Code snippet showing 'ViewResolver' which resolves logical view to a particular view technology.

Figure 4.11. Code snippet showing `@Aspect` annotation used in TMS application, the container will know that 'ApplicationMetricsAspect' is the aspect oriented class.

Figure 4.12. Code snippet showing `@Pointcut` expression declaration on a method.

Figure 4.13. Code snippet showing Spring's AOP `@Around` advice annotation declaration on a method used in TMS application.

Figure 4.14. Code snippet showing Spring email service configuration in TMS application.

Figure 4.15. Code snippet showing Spring IoC `@Autowired` annotation used in TMS application.

Figure 4.16. Code snippet showing `@Qualifier` annotation used in TMS application.

Figure 5.1. Complete architectural design of TMS application.

Figure 5.2. Use case diagram for TMS application.

Figure 5.3. Data model used in TMS application.

Figure 5.4. Sequence diagram for adding a new record into the database.

Figure 5.5. Sequence diagram for updating the record into the database.

Figure 5.6. Sequence diagram for deleting a record from the database.

Figure 5.7. Sequence diagram for viewing all the records from the database.

Figure 5.8. Sequence diagram for viewing all the records from the database in ExtJs view.

Figure 5.9. Code snippet showing web.xml configuration file, 'DispatcherServlet' definition used in TMS application.

Figure 5.10. Code snippet of application context (app-config.xml) used in TMS application.

Figure 5.11. Class diagram for the package ‘com.sdsu.tms.controller’ in TMS application.

Figure 5.12. Code snippet of ‘StudentThesisController’ for in TMS application.
Figure 5.13. Code snippet of ‘ExtJsStudentThesisController’ used in TMS application...................................................................................................................50

Figure 5.14. Class diagram of ‘com.sdsu.tms.service’ package used in TMS application...................................................................................................................53

Figure 5.15. Code snippet of ‘StudentThesisService’ service object used in TMS application...................................................................................................................54

Figure 5.16. Class diagram for ‘com.sdsu.tms.dao’ package used in TMS application........55

Figure 5.17. Code snippet of ‘StudentDetailsDao’ used in TMS application. .................56

Figure 5.18. Code snippet of ‘studentDetailsHibernateDaoImpl’ implementing ‘StudentDetailsDao.’ ....................................................................................................57

Figure 5.19. Code snippet of ‘studentDetailsJDBCDAOImpl’ implementing ‘StudentDetailsDao.’ ....................................................................................................57

Figure 5.20. Code snippet showing db-config.xml for database configuration shows the configuration in app-config.xml. ...............................58

Figure 5.21. Properties declared in db-config.xml for Hibernate configuration used in TMS application........................................................................59

Figure 5.22. Code snippet showing ‘StudentDetails’ as the ‘Entity’ used in TMS application...............................................................................................................61

Figure 5.23. JDBC connection properties in a utility file for MySql database used in TMS. ...........................................................................................................63

Figure 5.24. TMS JDBC code for opening and closing the connection, the developer should handle the connection management. ............................................63

Figure 5.25. Typical way of mapping column name from the result set to Java model class...........................................................................................................64

Figure 5.26. Class diagram of ‘com.sdsu.tms.aop.aspect’ package used in TMS application...........................................................................................................64

Figure 5.27. Aspect class addressing cross cutting concern in TMS application.............66

Figure 5.28. Configuring ViewResolver in application context (app-config.xml)...........67

Figure 5.29. Enter the URL: localhost:8180/thesis-manager to access TMS application.....................................................................................................................68

Figure 5.30. TMS application’s homepage displayed after entering the URL showing the entire previous student records. .........................................................69

Figure 5.31. Searching record with ‘red-id’ in TMS application. ......................................69

Figure 5.32. Search results after entering the key ‘red-id.’ ............................................69

Figure 5.33. Searching record with ‘advisors name’ in TMS application.......................69

Figure 5.34. Search results after choosing the ‘advisor’s name.’ ..................................69
Figure 5.35. Screens to add new document along with validations.................................70
Figure 5.36. Newly inserted record in the home page information table. .........................71
Figure 5.37. Gmail notification in TMS application, a new email notification will be sent to the student’s Gmail account after successfully saving their records..............71
Figure 5.38. Editable screen in TMS application, popping out after clicking edit link in the home screen-information table. .................................................................71
Figure 5.39. New record after editing the information, reflected in the home screen table.................................................................................................................................72
Figure 5.40. Deleting a record in TMS application, caution window popup before deleting the record.................................................................................................................72
Figure 5.41. Option to upload the document from the system............................................72
Figure 5.42. User will be prompted whether to open or save the document in to the system. ........................................................................................................................................73
Figure 5.43. Full view of student details table in ajaxified view implemented in ExtJs........74
Figure 5.44. ExtJs screens to add new student details and edit screen for editing student information with all the necessary validations..................................................74
Figure 5.45. ‘Save’ operations performed by taking different file sizes using Hibernate DAO. ............................................................................................................................75
Figure 5.46. ‘Save’ operations performed by taking different file sizes using JDBC DAO.................................................................................................................................75
Figure 6.1. Maven 3.0.3 running in Mac OS X. .................................................................78
Figure 6.2. How to start Apache Tomcat application server in the system.......................78
Figure 6.3. Tomcat 7.0 running in Mac OS X......................................................................79
Figure 6.4. Project Structure according to Maven archetype in Eclipse IDE....................81
Figure 7.1. North America job trends in Java EE frameworks...........................................87
Figure 7.2. Latest trends in document-based databases by percentage growth for the past seven years.................................................................88
Figure 7.3. Latest trends in the front-end technologies by percentage growth for the past seven years.................................................................88
CHAPTER 1

INTRODUCTION

1.1 JAVA ENTERPRISE APPLICATIONS

Today JEE (Java Enterprise Edition) is extensively used in enterprise applications. The general way of modeling an enterprise application is with models that include multiple layers such as a client layer, web presentation layer, business logic layer, and database layer. However, because of the increase in complexity of an enterprise application, the multilayered model failed to sustain the complexities. This pattern can be seen in the JEE specifications. My research led me to the trends of open source tools. These open source tools radically reduced the development time and complexity for the applications. In fact, some of these tools have evolved and matured enough to be used as viable replacements to popular JEE options. Many open source tools appeared in all layers and aspects within the process of building a web application.

1.2 SPRING AND HIBERNATE FRAMEWORKS

To solve the object relational complexity, Hibernate—an ORM tool—was widely used when working with a data persistence layer [1]. On the other hand, for front-end and business logic layers, Spring—an open-source framework—supporting the MVC (model, view, and controller) design model for building web application running on a Java platform was widely used. Spring AOP (Aspect Oriented Programming) that complements object-oriented programming by providing modularization of cross cutting concerns such as transaction management, logging, security, performance metrics, and email notifications that cut across multiple types and objects, have also become widely-used. Spring and Hibernate tools have significantly reduced the application development time and complexity. Furthermore, instead of using Java heavyweight objects, these technologies enabled POJOs (plain old Java objects) that were deployed in lightweight containers. To understand the trend of opting for open source tools as better alternatives over traditional heavy weight JEE options, I started my research by studying the ORM model in Java, Hibernate that will be
studied in Chapter 3. For the higher layers, in Chapter 4, Spring web application
development framework will be introduced. After that, applying the knowledge to real-world
application, TMS (Thesis Management Service) as a web application will be built based on
Spring and Hibernate technologies with an option of switching between different front-end
technologies like JSP (Java Server Pages) and ExtJs (Extended Java Scripting). The whole
process of design, implementation and the experience along with it will be described in
Chapter 5. Finally, I will sum up my gained experience as well as some overview of current
trends in the web application development in the coming chapters.

Figures 1.1 [2] and 1.2 [2] show the current trends in both front and back-end
technologies. Its quite evident from the graphs that Hibernate seems to be the popular ORM
(object relational mapper/mapping) framework of choice for enterprise data access layer.
Similarly, Spring MVC seems to be the popular choice for implementing the enterprise web
applications. Hence, I have chosen these two open source technology stack as one of the
parts of my thesis study.

Figure 1.1. North America job trends in front-end frameworks. Spring MVC and ExtJs
show significant growth each year for the past seven years. Source: Indeed.com. Job
CHAPTER 2

OBJECT RELATIONAL MAPPING

2.1 THE RELATIONAL PROBLEM

The first thing to notice is, in object-oriented programming, everything is an object. In a relational database, everything looks like rows and columns (table). Surely we can see that an object and a table are two different concepts. However, all Java enterprise applications deal with both objects and relational data, and any programmer can certainly say that these are two different models. The relational model deals with relations, tuples, and sets [3]. The object-orientated model, however, deals with objects, their attributes, and associations to each other [3]. As soon as we want to make objects persistent using a relational database, one issue appears: There is a gap between these two models, the so-called object-relational gap [3]. An ORM is a tool to bridge that gap. In this chapter, I am going to describe the theory behind ORM and my experience on picking up an appropriate ORM tool. Let us see how Java dealt with it initially.

2.2 PURE JDBC MAPPING

JDBC stands for Java Database Connectivity, which is a standard Java API (Application Program Interface) for database-independent connectivity between the Java programming language and a wide range of databases.

2.2.1 Boilerplate Coding

All enterprise applications, including the user interface, are designed around the relational model and SQL-based relational operations like CRUD (create, read, update and delete) [3]. The core drawback of JDBC is that it does not allow us to store objects directly to the database. We must convert the objects to a relational format. Similarly, when rows were returned from the database, we must convert each result set row into an instance of that event [3]. In order to overcome these drawbacks, nowadays, many technologies are supporting ORM for JEE framework. This approach involves a considerably low level of code reuse. JDBC may be an excellent solution for smaller applications, but when comes
to larger applications it has to face a lot of challenges. SQL can be fine-tuned in every aspect, but the drawbacks, such as lack of portability and maintainability, are significant, especially in the long run [3]. Another disadvantage is larger applications make heavy use of stored procedures, shifting some of the work out of the business layer and into the database [3].

2.2.2 General Interaction with RDBMS

General steps include:

1. First, load the JDBC driver (RDBMS specific), which communicates with the database.
2. Second, open the connection to database and then sending the SQL statements to get the results back.
3. Third, create JDBC Statement object. This object contains SQL query.
4. Fourth, execute the statement that returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
5. Processing the result set.
6. Closing the connection.

Figure 2.1 and Figure 2.2 show how to open a connection to send the SQL statements and get the results back.

2.2.3 Data Specific SQL Queries

In traditional JDBC the application uses database specific code in a large amount. The reason for writing large amount of code is to map table data to application objects, and vice versa is to map table fields to object properties [3]. When the properties of the table change, the developer has to hunt down all the queries in order to refactor the newly changed property. The code snippet in Figures 2.3 and 2.4 show the procedure to map the table columns to Java model manually in JDBC mapping. Figure 2.3 shows how JDBC maps the result set to the Java model. Figure 2.4, code snippet showing the difficulty of displaying the JDBC result set. The JDBC mapping shown in the Figure 2.4 clearly indicates the need for a simplistic ORM methodology as discussed in the next section.
public class StudentDetailsDaoImpl implements StudentDetailsDao {

    private static final Logger log = LoggerFactory.getLogger(StudentDetailsDaoImpl.class);
    private final String INSERT_SQL = "insert into " + TMSConstants.TMS_TABLE_NAME1.STUDENT_DETAILS + " (adname, content, content_type, created, filename, name, redid, topic, email) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    private final String DELETE_SQL = "delete from " + TMSConstants.TMS_TABLE_NAME1.STUDENT_DETAILS + " where id=?";
    private final String UPDATE_SQL = "update " + TMSConstants.TMS_TABLE_NAME1.STUDENT_DETAILS + " set adname=?, content=?, content_type=?, created=?, filename=?, name=?, redid=?, topic=?, email=? where id=?";
    private final String DISPLAY_ALL = "select * from " + TMSConstants.TMS_TABLE_NAME1.STUDENT_DETAILS;
    private final String SEARCH_CRITERIA = "select * from " + TMSConstants.TMS_TABLE_NAME1.STUDENT_DETAILS + " where redid=? or adname=?";
    private final String FIND_BY_PK = "select * from " + TMSConstants.TMS_TABLE_NAME1.STUDENT_DETAILS + " where id=?";

    JdbcUtil jdbcUtil = null;

    public StudentDetailsDaoImpl() {
        jdbcUtil = new JdbcUtil();
    }

    public void save(StudentDetails document) {
        Connection conn = jdbcUtil.getConnection();
        PreparedStatement ps;

        try {
            ps = conn.prepareStatement(INSERT_SQL);
            ps.setString(1, document.getAdname());
            ps.setBytes(2, document.getContent());
            ps.setDate(4, jdbcUtil.convertToSqlDate(document.getCreated()));
            ps.setString(5, document.getFilename());
            ps.setString(6, document.getName());
            ps.setString(7, document.getCreated());
            ps.setString(8, document.getTopic());
            ps.setString(9, document.getEmail());

            int status = ps.executeUpdate();
            if (status != 0) {
                log.debug("Inserted record into Contact table :
                        + document.toString());
            } else {
                log.error("Error while inserting record in Contact table");
                throw new DAOException("Inserting user failed, no rows affected.");
            }
        }
    }
}
Figure 2.3. Code snippet showing JDBC code implemented in TMS application to map the Java model objects with relational data.

```java
private StudentDetails mapResult(ResultSet rs) throws SQLException{
    StudentDetails sd = new StudentDetails();
    sd.setAdname(rs.getString("adname"));
    sd.setId(rs.getInt("id"));
    sd.setContent(rs.getBytes("content"));
    sd.setContentType(rs.getString("content_type"));
    sd.setCreated(rs.getDate("created"));
    sd.setFilename(rs.getString("filename"));
    sd.setRedid(rs.getString("redid"));
    sd.setName(rs.getString("name"));
    sd.setTopic(rs.getString("topic"));
    System.out.println(sd.toString());

    return sd;
}
```

Figure 2.4. Code snippet for showing result set handling in TMS application.

```java
static public void displayResultSet(ResultSet rs)
    throws SQLException
{
    int i;

    // Get the ResultSetMetaData. This will be used for the column headings
    ResultSetMetaData rsmd = rs.getMetaData();

    // Get the number of columns in the result set
    int numCols = rsmd.getColumnCount();

    // Display column headings
    for (i = 1; i <= numCols; i++)
    {
        if (i > 1) System.out.print(",");
        System.out.print(rsmd.getColumnLabel(i));
    }
    log.debug("\n--------------------------------");

    // Display data, fetching until end of the result set
    while (rs.next())
    {
        // Loop through each column, getting the
        // column data and displaying
        for (i = 1; i <= numCols; i++)
        {
            if (i > 1) System.out.print(",");
            System.out.print(rs.getString(i));
        }
        log.debug("\n");
        // Fetch the next result set row
    }
}```
2.3 What is ORM?

What is ORM? Object-Relational Mapping is a programming technique for mapping two different domains, one with database and the other with object-oriented programming language. As discussed earlier, relational database has data represented in the form of tables and object-oriented programming will have data in the form of objects. Mapping between these two domains requires strong mechanism for the domains to refer each other [3].

2.3.1 The Cause – Model Mismatch

The row and column representation of data in a relational system is different from the flow of objects used in object-oriented Java applications [3]. We often see these object relational differences in all enterprise applications, and cost of this difference has been underestimated [3]. The differences between the object-oriented system and relational system become evident when an SQL-based relational database management system is used as a data source [3]. For instance, a series of objects cannot be saved to a database table; it must be stripped (disassembled) and persisted to columns of manageable SQL data types [3]. The paradigm mismatch can be seen as the following problems:

- Types: Granularity refers to the relative size of the types [3]. For example, what if one wanted to add a new data type to the database to store an instance of a Java class representing a student in our case (which has name, red-id, advisor, etc.) in a single column [3].
- Inheritance: In Java, we implement inheritance using super classes and subclasses [3]. How about implementing a standard support for super tables and sub tables in SQL [3].
- Identity: The problem of object identity, this happens when we need to check whether two objects are identical [3]. It is easy in the object-oriented world (checked with a==b or equality by value equals()), but in relational world, the object of identity is expressed as the primary key [3].
- Associations: In the relational model, associations represent the relationships between entities [3]. Object-oriented languages represent associations using object references; but in the relational world, an association represents a foreign key column, with copies of key values (and a constraint to guarantee integrity) [3].
- Data navigation: There is a fundamental difference in the way to access data in Java and in a relational database [3]. In Java, when we access a student’s Red-ID information for instance, we could call something like ‘Student.getStudentDetails().getRedIDNumber()’ [3]. This is how objects are accessed in object-oriented world, and it is often described as walking the object
network [3]. You travel from one object to another, following the instances. Unfortunately, this isn’t an easy way to retrieve data from an SQL database [3].

2.3.2 Need of ORM

So with such an extensive object-relational mismatch problems, finding solutions for those problems are the need of the hour. Statistics shows that the main purpose of up to 30 percent of the Java application code written is to handle the tedious SQL/JDBC (Boilerplate Coding) and manual bridging (mapping) of the object-relational paradigm mismatch [3].

Figure 2.5 shows the gap between the object model and the relational database, and how ORM filled it by mapping.

![Figure 2.5. Mapping between object models and relational database.](image)

2.3.3 ORM Design – Domain Model

The primary classification of domain model can be divided into two categories:

- Entities
- Services

Entities preserve the state of the application, and Service is responsible in performing operations that require more than one entity. It links the outside world of the domain model, breaking through dependencies.

2.3.4 How ORM Managed It?

What are the reasons for using an ORM tool? It clearly eliminates the use of SQL (boilerplate coding). It does away with mundane data persistence coding tasks. Hibernate is a promising example of a decent ORM tool. These are the benefits I have observed by using ORM.
2.3.4.1 DEVELOPMENT PRODUCTIVITY

Using an ORM tool helps eliminate most of the banal work like opening a database connection, getting statement and querying on it, and lets the developer concentrate on the business problem. ORM when used with the appropriate tools, will significantly reduce development time; hence increase the overall productivity [1].

2.3.4.2 CODE MAINTAINABILITY

Using object/relational persistence significantly reduces lines of code. Fewer lines of code reduce the plumbing work required and make the system more transparent. Most important, a system with less code is easier to refactor [1]. This makes the process easier to maintain and debug, allowing more effective usage of object orientation on the Java side, and helps reduce changes to the model [1].

2.3.4.3 VENDOR INDEPENDENCE

This is the fun part. ORM can abstract the application from underlying SQL database. This brings certain amount of portability to the application. So, the developer does not have to worry much about the underlying SQL database there by giving database-vendor freedom to the developer [1].

2.3.5 Criteria for Choosing an ORM Tool

There are many ORM tools available in the market and depending on the design and complexity of an application one should choose a proper ORM tool-based on technical and non-technical criteria.

2.3.5.1 TECHNICAL CRITERIA

When choosing an ORM tool to suit the needs of an application, several criteria should be kept in mind, and certainly it will depend on a lot factors, like flexibility, performance and even financial aspects (commercial – noncommercial) [1]. In terms of functionality and technical features, the following criteria specific to ORM should be considered [1].
2.3.5.1.1 Basic Features

- Handle one-one, one-many, many-one relation [1].
- Support for transaction management.
- It should be object-oriented [1].

2.3.5.1.2 Extended Features

- Support for dynamic queries, searches based on filters (like criteria API) [3]. It is important to be able to use a powerful query language like HQL (Hibernate Query Language) [3].
- Easy switching between the databases. The application should be able to switch between different data persistence implementations easily.

2.3.5.1.3 Flexibility

- Providing flexibility on query language other than SQL, like HQL.
- Concurrency management.
- Support for the data types specific to the database management system (identity columns, sequences, auto increments) [3].

2.3.5.1.4 Ease of Use

- Support for GUI-based mapping tool.
- Easy generation of the classes. This can expedite the development.
- Easy generation of the database schema.

2.3.5.1.5 Optimizations, Performance, Design

- Control on loading the data. Lazy loading (the loading of some data should be under the control of developer) [3].
- Mapping only desired columns. We may need the blob fields only at certain point, under certain conditions, and so it is better to load them only at that time [3].
- Queries should be cache dynamic, so that there will be less number of hits to the database [3].
- Automatic synchronization of modified columns [3].
- Multiple row column Updates or deleting large number of rows at a time [3].

2.3.5.1.6 Compatibility

- Easy maintainability.
- Easy switching.
2.3.5.1.7 Additional Features

- Freedom in the design of the classes. Everything should be a POJO.
- Supports external mapping file or not? Supports attributes (annotations) in code or not? [1]
- Advantages of external files: mapping entirely externalized, no plumbing code in the classes [1]. Like in TMS application all the necessary database connection details are given in a separate db-config.xml file.
- Support for annotations, thanks to Java 5.0, which support new metadata style called annotations [1].
- Should be able to update the database based on the requirement, like ‘saveorupdate()’, ‘merge()’, ‘update()’, etc. [1].

2.3.5.2 NON-TECHNICAL CRITERIA

Beside the technical criteria there are also a lot of other non-technical criteria that could influence the final choice:

- Open source or commercial.
- Performance and flexibility.
- Scalability and memory conception.
- Ease of use, time to be up and running.
- Documentation and active forum support
- No beta or even alpha releases.
- Less prone to bugs and frequent updates, bug fixes, evolutions.
- Source code provided or not.
- Support for multiple platforms (Windows and Linux).
CHAPTER 3

HIBERNATE-ORM FRAMEWORK

3.1 HIBERNATE REVOLUTIONS

Need of an alternative solution for retrieving data from the database became mandatory after observing the traditional JDBC coding problems. It has been observed that a powerful methodology should come into existence for mapping the object and the relational data. The solution called ORM came in to existence and has been widely accepted since Hibernate is an open source implementation of it [3].

Hibernate not only takes care of the mapping between the Java classes to database tables (and from Java data types to SQL data types) [3], but it reduces the development time when compared to JDBC manual data handling, and also we can query and retrieve the data required for the application [3].

Hibernate relieves the developer from 95 percent of common data persistence related programming tasks [3]. For applications, which use stored-procedures, Hibernate may not be the viable solution to implement business logic in the database [3]; it works well with the application, which encompasses object oriented domain model, and Java middle-tier, which holds the business logic [3]. The main relief for the java developer is not to worry about vendor-specific SQL code as Hibernate will take care of the result set mapping with the Java objects [3].

In order to feel the advantage of using Hibernate a real world application that involves daily CRUD operations is implemented in the application called TMS. Detailed explanation on how Hibernate is implemented in TMS is discussed in Chapter 5: “Design and Implementation”.

3.1.1 Top Level Hibernate Workflow

1. No need to worry about the connection management. Hibernate will take care of opening the connection to database.
2. If we need to handle complex queries, we can convert HQL statements to database specific statement.
3. No need to specially handle the result set, Hibernate provides the result set.
4. After receiving the result set we just have to map the relational data with Java objects, which are directly used by Java application.

   Hibernate takes care of connection management with the help of properties file. Automatic mapping is performed on the basis of the properties defined in hbm XML file defined for particular Java object [4].

### 3.1.2 How Hibernate Works Internally

1. First it loads the configuration file by creating the configuration object [4].
2. It will automatically load all the hbm XML mapping files [4].
3. After creating configuration object, session factory is created from the configuration object.
4. On session factory, we have to get a session.
5. Then we query with HQL on that session.
6. The last thing to do is to execute the query to get the list containing Java objects [4].

### 3.1.3 The Hibernate Core

It is the base for providing the service or persistence; mapping in Hibernate is done via native API, which uses metadata stored in the XML file, lately with annotations [3]. It uses HQL for querying complex data as well as programmatic query interfaces for Criteria (special queries for retrieving data on conditions). Now, in Hibernate most of the HQL statements are object oriented and there is no need to write the HQL code [3].

Hibernate can be used independently or can be integrated with any JDK friendly framework. It can be used in any Java enterprise applications as long as we configure data source for Hibernate [3]. In TMS application for the persistence layer (DAO [Data Access Object]), Hibernate APIs and queries, and metadata required for the Hibernate to map both Hibernate XML file along with Hibernate annotations [1] was implemented easily. Applications metadata is defined by using annotations. Thanks to JDK 5.0, which understands the metadata through annotations instead of XML configuration: type-safe annotations are embedded directly in the Java source code [1]. For example, to write the condition on the property, which should never be null, that the column like ‘red-id’ should be unique, etc. can be directly mentioned in the domain model. Because of the
annotations the above-mentioned constraints on the domain model can be declared in the Bean itself [1].

3.1.4 Hibernate Entity Manager

Hibernate Core gets the Java Persistence API (JPA) features by the help of a wrapper called Hibernate Entity Manager. Hibernate Entity Manager utilizes Hibernate annotations for getting JPA features [3]. All the tables for the application are configured using annotations instead of XML file configuration.

3.2 Hibernate Basic Architecture

Hibernate architecture has three main components:

1. Managing the connections in Hibernate: All the connections are managed by connection management service. The most expensive part in any database transaction is getting the database connection. It requires a lot of resources for opening and closing the database connections [1].

2. Managing the transactions in Hibernate: In order to execute parallel statements there should be a methodology to look after the management of resources, transaction management will take care of parallel execution of statements [1].

3. Managing the ORM: Object relational mapping maps the relation data to Java object model [1]. We can select, insert, update and delete the records from the table using ORM. Once the object is passed to ‘Session.save()’, based on the object state Hibernate executes the necessary query [1]. For simple ORM, simple architecture is used (see Figure 3.1 [5]). For complex operations, complex architecture with the ORM, connection management and transaction management are used [1]. Figure 3.1 demonstrates that view from high-level.

For simple ORM, simple architecture was used because for the simple architecture, the application can use JDBC connections and manage its own transactions [1]. Therefore, Hibernate is not needed for connections and transactions management [1]. This approach uses a minimal subset of Hibernate’s APIs (see Figure 3.2 [5]) [1].

For complex operations Hibernate takes care of all the necessary connections and transactions [1]. That is why it uses complex architecture (see Figure 3.3 [5]) [1].

Definitions of objects in full architecture defined in the Hibernate documentation:

- SessionFactory (‘org.hibernate.SessionFactory’): The application gets the session instances from the Session Factory [1]. It is a heavy weight instance, which will be instantiated once per database [1]. It is a thread safe cache that takes care of all the connections [1].


- **Session (’org.hibernate.Session’)**: The Session is responsible for managing the operations like retrieving and storing the objects. It is a single-threaded, short-lived object that manages the persistence of an object by wrapping the JDBC connection upon which the query is executed [1].

- **Persistent objects and collections**: These objects are simple JavaBeans/POJOs; the objects in persistence are associated with (exactly one) session [5]. Objects are detached and can be used freely in any application layer (e.g. directly as data transfer objects to and from presentation) once this session gets closed [1].

- **Transient and detached objects and collections**: Instances of POJOs that are not currently associated with a Session are called transient or detached objects [1]. This happens in two conditions, either the application might have instantiated the objects but not yet persisted or they might have been instantiated by a closed Session [1].

- **Transaction (’org.hibernate.Transaction’)**: It is a single-threaded, short-lived object that takes care of all the transactions [1]. A session might span several transactional threads [1].

- **ConnectionProvider (’org.hibernate.connection.ConnectionProvider’)**: A connection pool is used to minimize the number of connections opened between application and database [1]. In order to use the user defined connection pooling the
ConnectionProvider has to be abstracted from the underlying Datasource or DriverManager [1].

- TransactionFactory (‘org.hibernate.TransactionFactory’): It provides transactional instances [1].

3.3 \textbf{CONNECTION POOLING}

Since, opening a connection to a database is expensive than executing an SQL statement, closing it and utilizing it for another connection is very important. So, a connection pool is used to manage the number of connections opened between application and database. By doing so we can return the connections back to the pool when completed, otherwise, the application will run out of available connections. Hibernate supports a variety of connection pooling mechanisms. The TMS application uses c3p0 Connection pooling that comes with the Hibernate; we can configure it by adding connection-pooling properties in hibernate.cfg.xml. We can see c3p0 configuration in Chapter 5, db-config.xml.

3.4 \textbf{HIBERNATE OBJECT STATES}

All the instances of persistent POJOs will have any of the following states:

- Transient: All the newly instantiated objects are transient in nature [1]. That means Hibernate Session will not associated with these objects [1].

- Persistent: All the objects that are saved or loaded in the database with an identifier will have persistent state [1]. These objects will be under Session scope [1]. Hibernate detects any changes made to these objects, and synchronizes them with the database [1].

- Detached: Once the objects persist, and their Session gets closed then all those objects will be in detached state [1].

3.5 \textbf{MAPPING FILE STRUCTURE BYANNOTATIONS IN HIBERNATE}

In earlier versions of Hibernate, for a class to know the information about the data to be stored into the database and mapping the objects with relational data, a configuration file (XML file) is used which had all the necessary information. But in modern configuration, Hibernate takes the advantage of Java 5.0 annotations and using simple yet powerful annotations we can do above-mentioned configuration. Figure 3.4 shows the beauty of Hibernate mapping annotation-based configuration, which is used in TMS application’s implementation.
Figure 3.4. Code snippet showing Java model (entity class) mapping using Hibernate annotations, used in TMS application.

```java
@Entity
@Table(name="STUDENT_DETAILS",
uniqueConstraints = @UniqueConstraint(columnNames = {"redid"})))
public class StudentDetails {

    @Id
    @GeneratedValue
    @Column(name="id")
    private Integer id;

    @Column(name="name", nullable = false)
    private String name;

    @Column(name="topic", nullable = false)
    private String topic;

    @Column(name="filename")
    private String filename;

    @Column(name="content", nullable = false)
    @Lob
    private byte[] content;

    @Column(name="content_type")
    private String contentType;

    @Column(name="created")
    @Temporal(TemporalType.DATE)
    private Date created;

    @Column(name="redid", nullable = false)
    private String redid;

    @Column(name="adname", nullable = false)
    private String adname;

    @Column(name="email", nullable = false)
    private String email;
```

Typically in traditional Hibernate configuration, after creating the table in the database and corresponding Java class in the application layer, for example in TMS application the ‘STUDENTDETAILS’ table should be mapped to Java class and the column names in the ‘STUDENTDETAILS’ table should be mapped to the Java variables in the ‘StudentDetails’ class. The ‘HibernateMapping’ files (*.hbm.xml) will have all the mapping information.
Because of Hibernate 3 annotations (thanks to Java 5.0), which provides a powerful metadata and elegant API to simplify the object and relational table mapping. We can remove the ‘HibernateMapping’ file by using below mentioned annotations that directly map the table with the Java class. Now, lets look at the typical way of mapping the table column and the Java model class.

3.6 HIBERNATE CONFIGURATION

Hibernate configuration can be done by using GUI-based tools. However, as a developer, there is no much difference in comparison with XML mapping or other configuration files. Figure 3.5 shows the Hibernate configuration management, connection pooling and transaction management are done manually without using GUI, implemented in TMS application.

3.7 HIBERNATE QUERY LANGUAGES

Hibernate Query Language or HQL is extremely powerful object oriented query language. HQL is much similar to SQL, but it operates on persistent objects instead of relational tables and columns. Hibernate automatically translates the object oriented query into conventional SQL query and execute it against underlying database. It also supports polymorphism and associations.

Hibernate uses Criteria API and native SQL to provide QBC (Query by Criteria) and QBE (Query by Example). In the implementation of the DAO objects in TMS application, we can see how HQL is implemented.

Use of HQL includes:

- SQL queries will be represented in the form of objects. So one can take full advantage of objects and can use classes and properties instead of tables and columns.
- The HQL will return the query results in the form of objects. There is no need to populate the data from these result sets.
- HQL fully supports polymorphic queries. Instances of super object along with subclasses of the persistence class can be retrieved using these queries.
- Hibernate Queries are easy to understand and can be learned easily. They are easy to implement in the applications.
- HQL supports pagination for setting number of records per page, fetch join with dynamic profiling, inner/outer/full joins, and Cartesian products. It also supports
projection, aggregation (max, avg) and grouping, ordering, subqueries and SQL function calls [1].

- Subqueries are queries within another query. Hibernate supports subqueries if the underlying database supports it [1].

- HQL queries does not depend on the database, they are independent of the database: So, if the application needs to change the back-end database then there will not be any affect on the HQL Queries [1]. Figure 3.6 shows some of the examples of HQL.

### 3.8 Criteria API in Hibernate

Criteria is an object oriented query API where as HQL means string concatenation. That means we can take all the advantages of object oriented API in querying the database. Criteria are the powerful and elegant alternative to HQL, well adapted for dynamic search functionalities where complex Hibernate queries can be generated ‘on-the-fly’ [1]. The real
usage will be, when trying to search the records based on users choice, for example, either by RED-ID or by advisor’s name (see Figure 3.7).

```java
@SuppressWarnings("unchecked")
public List<StudentDetails> findByCriteria(String redid, String adname) {
    Criteria criteria = sessionFactory.getCurrentSession().createCriteria(StudentDetails.class);
    Criterion crit1 = Restrictions.ilike(TMSConstants.STUDENT_DETAILS.RED_ID, redid);
    Criterion crit2 = Restrictions.ilike(TMSConstants.STUDENT_DETAILS.ADNAME, adname);
    criteria.add(Restrictions.or(crit1, crit2));
    return criteria.list();
}
```

Figure 3.7. Criteria used in TMS application for searching the record by ‘red-id’ or by ‘advisor name.’

The session gives the criteria object upon which the criteria query is performed, it uses ‘createCriteria()’ method to identify the primary class (and thus model) on which the query is performed. After creating the criteria restrictions, projections, and orderings are attached to the query using the factories [6]. The factory for creating the criterion instances to
narrow down the objects (rows) we want from criteria queries is provided by
‘org.hibernate.criterion.restrictions’ class [6]. These criteria are used to determine which
persistent objects from the database are included in the results of your query. Some of the
criterion restrictions are shown in Table 3.1 [7].

Table 3.1. Properties Used in Criterion Restrictions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RestrictionallEq</td>
<td>This is used to apply an ‘equals’ constraint to each property in the key set of a Map</td>
</tr>
<tr>
<td>Restrictionbetween</td>
<td>This is used to apply a “between” constraint to the named property</td>
</tr>
<tr>
<td>Restrictioneq</td>
<td>This is used to apply an “equal” constraint to the named property</td>
</tr>
<tr>
<td>Restrictionlike</td>
<td>This is case-sensitive “like”, similar to Postgres ilike operator</td>
</tr>
<tr>
<td>Restrictionor</td>
<td>This returns the disjunction of two expressions.</td>
</tr>
<tr>
<td>RestrictionidEq</td>
<td>This is used to apply an “equal” constraint to the identifier property.</td>
</tr>
<tr>
<td>Restrictionlike</td>
<td>This is used to apply a “like” constraint to the named property.</td>
</tr>
<tr>
<td>Restrictionlt</td>
<td>This is used to apply a “less than” constraint to two properties</td>
</tr>
<tr>
<td>Restrictionnot</td>
<td>This returns the negation of an expression</td>
</tr>
</tbody>
</table>

Source: RoseIndia. Properties of Criterion Restrictions, n.d.

3.8.1 Advantages of Hibernate

- Very less boilerplate coding required. No ugly code for getting the database connection and executing the queries in between the Java code. This saves a lot of development and debugging time of the developer.

- Writing JDBC statement, setting the parameters, executing query and processing the result by hand is lot of work, Hibernate will save all tedious efforts.

- It is non-intrusive, almost completely transparent, persistence even for POJOs.

- Seems to have a very good tooling support, for example lot of IDE’S like Eclipse, NetBeans, etc. which provide lot of tooling for better Hibernate development and configuration.

- It seems Hibernate has a decent documentation; it has active Hibernate community forums and lot of books and tutorials dealing with Hibernate.

- In terms of performance, its code is robust, almost bug-free.
- It has a very good query language (HQL), with the option of falling back to SQL when needed.
- An open-source project.

### 3.8.2 Disadvantages of Hibernate

- I found out that there is a reasonable learning curve to learn Hibernate concepts as it has lots of API to learn.
- Error resolving and debugging is little difficult, as Hibernate abstracts the underlying ORM functionality and its API.
- I found out that Hibernate is slower than pure JDBC as it generates lot of SQL statements during runtime.
- It seems it requires good knowledge of database design in order to have a deep knowledge of database design in order to use Hibernate efficiently.
- Configuration of caching, fetching strategy is little complex even for small applications.
CHAPTER 4

SPRING FRAMEWORK

4.1 BEFORE SPRING REVOLUTION

Earlier enterprise applications were based on JEE standards. JEE application server was used to deploy these applications irrespective of the operating systems. Several aids were provided to the application running on these application servers. It offered services, such as transaction management, messaging, mailing service, a directory interface (JNDI), etc. All JEE compatible code could make use of these services, as long as the code is written against the interfaces defined in the JEE specifications [8]. Coding to these standards became heavy because of using JEE standards. Writing a component (EJB [Enterprise JavaBean]) requires a set of deployment descriptors (configured in XML files), home interfaces, remote/local interfaces, etc. and most of the deployment descriptors are retailer specific, so ‘migrating an application between the retailers became hard’ [8].

The second problem is, there was the JNDI (Java Naming and Directory Interface) searching problem. When a component requires another component, the component itself has to search for other component that it depends on. This convention happens by name, so the name of the dependency is hardcoded in the code or deployment descriptor [8]. Apart from this, letting components communicate with each other from JEE application servers of different retailers was always tough.

And finally, a lot of cases, components might not need all the services provided by the application server but since there was no independent API to build components, all the JEE components became heavy weight, forcing the developers to implement multiple interfaces, deployment descriptors, which resulted in bloating the application. So, most of the developers supported programming in plain old Java objects instead of writing unwanted XML configurations.

This is where the Spring framework came in to the picture. Spring framework brought a lightweight container instead of JEE heavy weight container. This is a container where the lifecycle of the POJOs were maintained. The components now were
as light as possible. Spring’s container provided all the necessary wiring between the components. Unlike JEE components, POJOs (Beans) can use required services without sticking to a model.

4.2 HISTORY OF SPRING FRAMEWORK

Spring framework is released by Rod Johnson with the publication of his book Expert One-on-One JEE Design and Development in October 2002 [9], wrote the first version. Under Apache 2.0 license the first version of the framework is released in June 2003. Since then Spring has undergone two significant revisions: Spring 2.5, released in November 2007 and Spring 3.0 released in August 2009. Any Java Application can use the core features of Spring framework on top of the JEE platform. It became popular in the Java world as an alternative to, replacement for, to the EJB (Enterprise JavaBeans) [9].

4.3 SPRING FRAMEWORK

Spring is an open source framework that integrates well with all Java technologies/APIs that can make use of simple POJOs [10]. The Spring framework easily handles the complexity of an enterprise application development. It is a lightweight solution and a potential all in one container for building the enterprise applications. Spring framework provides lightweight infrastructure to support Java applications. Spring container handled problems of handling unnecessary configurations so that the developer can focus on the application’s core development. All we need is simple POJOs and apply enterprise services non-invasively on them. Spring gives the developer the freedom to use only those parts that are necessary. We can use the IoC (Inversion of Control) container to integrate with several front-end technologies like Struts, JSF, etc., and also integrates well with several ORM frameworks like Hibernate, iBatis, Open JPA, or the JDBC abstraction layer [10]. The Spring framework supports declarative transaction management, remote access to our logic through RMI or web services, and various options for persisting the data. It offers a full-featured MVC framework, and enables the application to handle cross cutting concerns by integrating AOP transparently into the code [10].
4.4 SPRING MODULES

As per the Spring source documentation the Spring framework consists of features organized into about 20 modules. These modules are grouped into core container, data access/integration, web, AOP, instrumentation, and test, as shown in Figure 4.1 [10].

![Spring Framework Runtime](image)

4.5 SPRING 3.0 FEATURES AND ENHANCEMENTS

Java 5 features like generics; varrags, annotations and other language improvements were well adapted in Spring 3.0. Spring uses generic collections and maps, consistent use of generic FactoryBeans, and also consistent resolution of bridge methods in the Spring AOP API [11]. TransactionCallback and HibernateCallback interfaces can declare a generic result value now [11]. Another important enhancement is using @Annotations; annotations are metadata and should be processed by the compiler at compile time [11].

Java 5.0 provides the ability of annotating Java elements and creates custom annotation types. The compiler and the framework tools can then read and process annotated data. Spring adapts this feature to the fullest, and it replaced XML-based configuration into
annotation-based configuration, which enabled developers to configure Beans inside the Java source file. Spring framework provides different custom Java5+ annotations [11]. These annotations can be used in transactional, AOP, etc. There are core Spring Annotations, Spring MVC Annotations, AspectJ Annotations, JSR-250 Annotations, Testing Annotations, etc. So, using the advantages of Java 5.0 features, mainly annotations [11], Spring 3.0 made the application configuration easy by introducing Spring annotation configuration along with the XML configuration.

Out of the above mentioned modules, the application TMS uses the following three important core functionalities along with Spring IoC:

- Spring MVC
- Spring AOP
- Spring Mail

### 4.6 The Spring MVC 3.0 Architecture

MVC design pattern-based web frameworks became famous in the past few years; MVC was originally conceived at XEROX PARC around the 1978-79 time frames and was later implemented in the Smalltalk-80 class library [12].

#### 4.6.1 Basic Spring MVC Workflow

Spring’s web MVC framework was designed based on a DispatcherServlet that communicates with the handlers (view resolution, locale and theme resolution as well as support for uploading files) via handler mapping. The default handler is a very simple controller interface, just offering a ModelAndView handle request (request, response) method (see Figure 4.2 [12]).

Controllers handle all incoming HTTP requests from a web browser. The use of the controller is it decouples the model and the view as well as it provides a channel for data exchange. Now, because of decoupling the model can worry only about the data but not the view (data is rendered in the view). The view, on the other hand can worry only about the data rendering but the model or business logic. This gives the application flexibility to change the view without changing the model [12].

Figure 4.3 [13] shows the step-by-step requests for processing the workflow in Spring MVC.

4.6.2 Client Request Processing Workflow in Spring MVC

Below is the detailed explanation on how the client requests are processed in Spring MVC approach.

4.6.2.1 THE DispatcherServlet

Spring MVC framework is a request driven, constructed around DispatchServlet that dispatches the request to controllers via handler mapping and facilitates exception handling. It follows a “front controller” pattern. This servlet integrates completely with Spring IoC container that allows it to use Spring features [14].

DispatcherServlet an actual Servlet (it inherits from the HttpServlet base class) declared in the web.xml of the web application. Using URL mapping, declared in the same web.xml file maps all the application requests to ‘DispatcherServlet.’ A web application can have any number of ‘DispatcherServlets.’ Each servlet will operate in its own namespace, loading its own application context with mappings, handlers, etc. Figure 4.4 shows the web.xml snippet with DispatcherServlet mapped in it [14].

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

<url-pattern>*.do</url-pattern>
```

Figure 4.4. web.xml configuration for invoking DispatcherServlet.

In the configuration shown in Figure 4.4, first, all requests ending with /*.do will be handled by the ’thesis-manager’ DispatcherServlet. Close associated term with the DispatcherServlet is the WebApplication Context. It usually represents a set of Configuration files that are used to provide configuration information to the application like various Bean definitions.
The framework will by default, on initialization of a DispatcherServlet, look for a WebApplicationContext by the file named [servlet –name] – servlet.xml in the WEB-INF directory inside the application package (see Figure 4.5 [15]). As the application grows this configuration file might get bigger, and it is always better split this information across multiple configuration files by using Listener Servlet called Context Loader represented by ‘org.springframework.web.context.ContextLoaderListener.’ Figure 4.6 shows the code snippet to add application context files apart from the default application context [14].


```xml
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    <!--WEB-INF/spring/applicationContext-security.xml-->
    /WEB-INF/spring/app-config.xml
  </param-value>
</context-param>

<!--filter-->[

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

Figure 4.6. Configuration to declare multiple application context files with <context-param>.
4.6.2.2 Handler Mapping

As a part of DispatcherServlet, handler mapping maps the incoming requests to appropriate handlers i.e. controllers. This component will be invoked by the incoming request to get the respective controller, which is responsible to handle the request. There are different ways to declare these mappings [14]:

- BeanNameUrlHandlerMapping
- ControllerClassNameHandlerMapping
- SimpleUrlHandlerMapping

Spring MVC 3.0 utilizes the functionality of annotations and the above declarations will be replaced with standard class level @RequestMapping annotation. The HTTP request method (“GET/POST”) will be handled inside @RequestMapping by virtue of method level annotations. Sometimes application might use HTTP redirect back to the client, before view gets rendered. This is desirable when one controller will be called for POSTed data, and the response gets delegated to another controller. Figure 4.7 shows ‘@RequestMapping’ annotation that maps the web requests on specific handler classes or handler methods.

```java
@RequestMapping(value = "/updateContact", method = RequestMethod.POST)
public String updateStudentDetails(
    @ModelAttribute("editContact") StudentDetails document,
    BindingResult result,
    @RequestParam(value = "file", required = false) MultipartFile File,
    SessionStatus status) {

    validator.validate(document, result);
    if (result.hasErrors()) {
        return "editContact";
    }
    documentService.update(document, File);
    status.setComplete();
    return "redirect:viewAllContacts.do";
}
```

Figure 4.7. Configuration to map a particular URL with ‘@RequestMapping’ Spring MVC annotation.

4.6.2.3 Controllers

Controllers interpret user input and transform such input into a sensible model, which is represented to the user by the view [14]. Spring provides different kinds of controllers based on the application requirement [14].
The controller interface provides the basic responsibility required for every controller; namely handling a request and returning a model and a view. The following are the controller components available in the Spring Distribution:

- SimpleFormController
- AbstractController
- AbstractCommandController
- CancellableFormController
- AbstractCommandController
- MultiActionController
- ParameterizableViewController
- ServletForwardingController
- ServletWrappingController
- UrlFilenameViewController.

TMS is a form-based web application utilizing the features of SimpleFormController. Spring MVC handle all forms using SimpleFormController. It provides configurable form and success views, and ‘onSubmit()’ chain for submission and finally returning back the ModelAndView Object to the DispatcherServlet.

In Spring MVC 3.0, the controller component will be defined by the ‘@Controller’ (see Figure 4.8) annotation indicating the class has the role of a controller.

```java
@Controller
public class StudentThesisController {
}
```

Figure 4.8. Controller configuration with ‘@Controller’ Spring annotation, it tells the container that ‘StudentThesisController’ is a controller class.

**4.6.2.4 MODEL AND VIEW**

ModelAndView or ModelMap class adds the model object to be displayed in a view. It uses traditional map implementation that automatically generates a key for an object when object gets added to it. The view is the String view name, which gets resolved by the ViewResolver object or by the view object, which can be specified directly [14]. The model is a map, allowing the use of multiple objects keyed (see Figure 4.9) by name [14].
Figure 4.9. Code snippet showing ModelAndView given to DispatcherServlet, it then resolves the logical view (‘showContacts’) with a particular view technology (‘showContacts.jsp’).

4.6.2.5 THE VIEWRESOLVER

Spring MVC provides a way to render the views; model objects will be rendered in the browser by view resolvers without holding to a specific view technology [14]. Spring provides support for JSPs, ExtJs, Velocity templates, etc. ViewResolver provides a mapping between view names and actual views [14].

4.6.2.6 VIEWS

The logical view name returned by the controller is resolved to a particular view technology by view resolver [14]. For instance, CS-TMS uses JSPs as views (see Figure 4.10) that are actually processed via the servlet engine. This is handled by InternalResourceViewResolver, which will issue an internal forward to the actual view.

Figure 4.10. Code snippet showing ‘ViewResolver’ which resolves logical view to a particular view technology.

4.7 SPRING AOP

Cross cutting concerns in the application like logging, security, performance metrics, email service and transaction processing, etc. were handled by the Spring framework by providing abstraction in an application through mean of AOP. It is sensibly new and not a replacement for object oriented programming. the following are some major AOP concepts, which are, not Spring specific:
• Aspect: A POJO with modularization, whose implementation might cut across multiple objects. For example, in TMS logging, performance metrics and sending an email after successfully saving a new record is implemented as Aspect.

• Joinpoint: Point during the execution of a program, such as method invocation [16].

• Advice: Action taken by AOP at Joinpoint. There are four types of Advices available in Spring [16].
  1. Around advice: Advice that surrounds a Joinpoint such as a method invocation. This is the most powerful advice that performs custom behavior before and after the method invocation. It is responsible to choose whether to proceed to the Joinpoint or to cut short execution by returning to its own return value, or throwing an exception [16].
  2. Before advice: Advice that executes before a Joinpoint, but which doesn’t have the ability to prevent execution flow proceeding to the Joinpoint [16].
  3. Throws advice: This advice will be executed if a method throws an exception. Spring provides strongly typed throw advice, and can be used to catch the exceptions without using Throwable or Exception [16].
  4. After advice: Advice that executes after a Joinpoint completes normally, example method without exception [16].

• Pointcut: A set of joinpoints specifying when an advice should be fired. The pointcut expression tells the AOP framework when and where to intercept AOP. AOP framework allows developers to specify pointcuts, for example, using regular expressions [16].

Some of the AOP annotation declarations in TMS application:

• Declaring an Aspect: Aspect classes (POJOs) are annotated with @Aspect annotation (see Figure 4.11) for the application to know that this particular class is an Aspect for the application.

• Declaring a Pointcut: Pointcut signature is a regular method definition with return type as void (see Figure 4.12) and is declared using the @Pointcut annotation. The below point cut expression indicates the executing of any method defined in the DAO package.

• Declaring around Advice: Around advice is declared in an aspect using the @Around annotation (see Figure 4.13). The first parameter of the advice method should be of type ProceedingJoinPoint [16]. Within the body of the advice, calling ‘proceed()’ on the ProceedingJoinPoint that causes underlying method to execute [16].

```java
@Aspect
public class ApplicationMetricsAspect {
}
```

Figure 4.11. Code snippet showing ‘@Aspect’ annotation used in TMS application, the container will know that ‘ApplicationMetricsAspect’ is the aspect oriented class.
4.8** SPRING MAIL**

Spring provides a higher level of abstraction to the developer for sending electronic mail and takes care of all the low level resource handling on behalf of the client [17]. The ‘org.springframework.mail’ package is the base level package for the email support, which has core interface ‘MailSender’ for sending emails. ‘From’ and ‘to’ properties of email is encapsulated in SimpleMessage class [17]. Emails higher level of abstraction for checked exceptions over the lower level mail system is with the MailException [17].

All the special JavaMail features are added in the ‘org.springframework.mail.javamail.JavaMailSender’ package. Figure 4.14 shows the code snippet of email functionality used in TMS application.

```java
@Service
public class STSMailService {

    public void sendMail(String from, String[] to, String subject, String msg) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(from);
        message.setTo(to);
        message.setSubject(subject);
        message.setText(msg);
        System.out.println("Sending email "+ msg.toString());
        mailSender.send(message);
    }
}
```

Figure 4.14. Code snippet showing Spring email service configuration in TMS application.
4.9 Spring IoC Container

IoC also known as Dependency Injection (DI) is used to manage the life cycle of the Beans and their dependencies in the application. The IoC container will look after the lifecycle, events and configuration of Beans that reside inside the container. The core principle of DI is: The objects in the higher level modules should not depend on the objects in the lower level modules. Both should depend upon abstraction, that is, application – should not depend on lower level components. This is because the lower level changes might affect the higher level components [18]. So in IoC, the objects define their dependencies on the other objects they work with, only through constructor arguments or properties that are set on the object instance after it is constructed. It is the container responsibility to wire these objects at runtime. The ‘org.springframework.beans’ and ‘org.springframework.context’ packages are the basis for IoC container [18].

Spring IoC reads the metadata on the Java elements to instantiate, configure and assemble the objects. The developer configures this metadata in two different ways:

1. Annotation-based configuration
2. Java-based configuration.

The TMS application uses annotation-based configuration. It is an alternative to XML setups, which rely on byte code metadata for wiring up components instead of XML declarations. The configuration is done inside the component class with a wide variety of annotation options.

@Autowired marks a constructor, field, and setter method as to be autowired by Spring IoC container. The @Qualifier annotation is used to control which Bean should be autowired on a field. ‘@Qualifier’ annotation is useful if the application has one service layer and two different DAO implementations. It is very easy to switch between the DAOs, just by changing the qualifier name. Figures 4.15 and 4.16 show the ‘@Autowired’, ‘@Qualifier’ annotations used in TMS application.

```java
private StudentThesisService documentService;
```

Figure 4.15. Code snippet showing Spring IoC ‘@Autowired’ annotation used in TMS application.
Figure 4.16. Code snippet showing `@Qualifier` annotation used in TMS application.
CHAPTER 5

DESIGN AND IMPLEMENTATION OF TMS SYSTEM

5.1 OVERVIEW

Since this research was started to study the advantages of open source frameworks in enterprise level Java applications, I thought implementing an application involving user interface with a design around the relational model and SQL-based relational operations like CRUD (create, read, update, and delete) might accomplish the desired purpose [3]. TMS is a simple prototype version of any online (banking, finance, health, wireless, etc.) CRUD application built on Spring and Hibernate framework. This application is used for managing students past thesis dissertation information. TMS has a number of functionalities such as viewing, adding, editing and deleting student record with an option of uploading and downloading thesis dissertation document. A search criterion is provided for searching the record either with students ‘RED-ID’ or ‘Advisor Name.’ It is a user-friendly advanced web application; once a new record has been saved into the database with a valid Gmail id then the student will receive email notification with all the persisted information.

5.2 TMS APPLICATION ARCHITECTURE

The Java web application being developed requires a user interface and SQL-based CRUD operations. The main aspects to consider while designing this application would be:

- The ability of the application to handle the volume of incoming client requests.
- The layer at which the business logic of this application should reside.
- The ability of the application to handle cross cutting concerns like logging, email service, performance metrics, etc.
- The ability of the application to handle the data persistence.
- The ability of the application to handle container level services such as transactions.

This application was built to understand the difference between Hibernate and traditional JDBC methodologies. For this, both Hibernate and traditional JDBC frameworks were implemented to handle and persist the data in exactly the same way. To understand the
advantages of Hibernate over JDBC and the differences between them, their simplicity of coding and their performance metrics were studied. That means the application should be able to use any one of these frameworks inside the application and get the same effect in terms of functionality [1].

The next thing to consider is the user interface construction using front-end technologies like JSP, ExtJs, and VELOCITY, etc. This mainly involves taking input from the user and submitting the same using form submission. The main design considerations for this would be:

- The flexibility of the application to be able to switch between either of the front-end technologies without affecting the connectivity to the subsequent layers.
- So, a good framework should be able to handle all the above mentioned considerations. But most of the present frameworks being used today adhere to only a few of these considerations. Generally, to overcome this a combination of these frameworks were implemented to achieve the desired result. We can achieve the same without all the hassles using only one popular open source framework called Spring MVC, which integrates, and implements all the functionalities of the above mentioned frameworks. The only drawback to the Spring MVC framework is the inability to perform data persistence. As a result, it has to be used in combination with the popular ORM framework-Hibernate or the age old JDBC API.

Spring MVC framework was used in TMS application because it offers the desired layered architecture, offers interceptors as well as controllers, making it easier to handle the client requests, offers better integration with view technologies like JSP, ExtJs, etc., provides a better decoupling between the layers. And also it integrates well with Hibernate and provides a clear separation between the model, views and controller. Figure 5.1 shows the complete architectural design of TMS application.

Table 5.1 illustrates the Layers used in TMS web application in order to attain our goals.

### 5.3 TMS Functional Specifications

- A homepage with CRUD functionalities for managing student information.
- Functionality to edit existing record and updating the information with the same in to the database.
Figure 5.1. Complete architectural design of TMS application.

Table 5.1. Technology Stack Used in TMS Application

<table>
<thead>
<tr>
<th>Presentation Layer</th>
<th>Controller Layer</th>
<th>Service Layer</th>
<th>Data Access Object Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSP, ExtJs</td>
<td>SpringMVC</td>
<td>Spring Annotated Services, Spring AOP</td>
<td>Hibernate / JDBC</td>
</tr>
<tr>
<td></td>
<td>DispatcherServlet</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Functionality to upload and download student’s thesis document in to the database.
- Functionality to search a particular record with RED-ID or advisor’s name.
- Functionality to send Gmail information to concerned student after successfully adding the record.
- Functionality to switch between two different front-end technologies (views) namely JSPs and ExtJs.
- A metrics page using Spring AOP to study the difference between Hibernate and JDBC transactions.
5.4 USE CASE FOR TMS APPLICATION

Figure 5.2 shows the use cases used in TMS application.

![Use case diagram for TMS application.](image)

5.5 TMS DATA MODEL DESIGN

We have two tables in the database (MySql):

- ‘STUDENTDETAILS’ table contains all the required information (see Figure 5.3) of each student including the file content, where as
- ‘STUDENTDAO_METRICS’ table contains all the metric information (see Figure 5.3), which holds the cross cutting information about the application.

5.6 SEQUENCE DIAGRAMS FOR TMS APPLICATION OPERATIONS

Figures 5.4 through 5.8 show the sequence diagrams for adding, updating, deleting and viewing all record operations performed in TMS application.

5.7 TMS APPLICATION FLOW AND CONFIGURATIONS

As seen from the sequence diagrams in Figure 5.3, all incoming requests from the client are intercepted by the Spring MVC dispatcher servlet, which maps the request to appropriate controller-using handler mapping. To invoke the ‘DispatcherServlet’ for client
requests, the web configuration file should contain a definition that invokes the ‘DispatcherServlet.’

5.7.1 Step 1

The first step when implementing Spring MVC is to configure the DispatcherServlet and handler mappings in the web.xml file. And also all the information about the components is configured first for the application to run successfully. Below are the detailed explanations on how to configure the applications DispatcherServlet and the components in the web.xml file.

5.7.1.1 Spring MVC Configuration

Spring MVC is enabled by adding Spring’s DispatcherServlet configuration as stated in web.xml (see Figure 5.9). Next comes the handler mapping, Spring uses different ways to configure handler mapping, TMS uses ‘SimpleUrlHandlerMapping’ as shown in Figure 5.9. From the definition of servlet-mapping tag, it tells that whatever be the client request (represented by *.do meaning any URL with .do extension), invoke the servlet by name ‘thesis-manager.’ In this case ‘thesis-manager’ is nothing but an instance of DispatcherServlet ‘org.springframework.web.servlet.DispatcherServlet.’ Once the DispatcherServlet is invoked, it tries to map the requests to appropriate controllers using handler mapping.
Figure 5.4. Sequence diagram for adding a new record in to the database.

For the ‘DispatcherServlet’ to map the requests to the appropriate controller, Spring MVC has to know the ways to scan the controllers in the application. So, the information has to be configured somewhere in the application that is called application context.

### 5.7.1.2 SPRING APPLICATION CONTEXT

All the information about application’s Bean (POJO) configuration is configured in the application context file. By convention, the Dispatcher Servlet will try to look in the
Figure 5.5. Sequence diagram for updating the record into the database.

Figure 5.6. Sequence diagram for deleting a record from the database.
Figure 5.7. Sequence diagram for viewing all the records from the database.

Figure 5.8. Sequence diagram for viewing all the records from the database in ExtJs view.
WEB-INF directory for the file ‘<servlet-name>-servlet.xml.’ By default, TMS application’s servlet will look for ‘thesis-manager-servlet.xml’ file in the WEB-INF directory. But as the application grows this default configuration file might get bigger as it has to hold a lot of application’s configuration information. So, it is wise to divided the application context across multiple files. This can be done using context loader represented by ‘org.springframework.web.context.ContextLoaderListener’, highlighted in black in the (see Figure 5.9).

As discussed in the above paragraph about the application context, it contains all the Bean definitions, which should be defined in the application context XML file. Now using Spring annotations, we can skip the traditional way of defining Bean definitions, Bean wiring, method of deployment and ways to scan the components in application context file there by significantly reducing the amount of XML Bean configuration.
5.7.1.3 **Spring MVC Annotation-Based Configuration**

<context:component-scan> names a package “com.sdsu.tms” to be scanned for classes with the ‘@Components’ to deploy. Subpackage is automatically scanned too.

<mvc:annotation-driven/> declares explicitly to support annotation-driven MVC controllers (i.e. @RequestMapping, @Controller, etc.). Once the application scans the controller, it has to process the request (see step 2).

Because of Spring’s support for annotations (thanks to Java 5.0), lot of plumbing work (when compared with traditional XML configuration) was drastically reduced. Features like annotation-driven dependency injection, auto-detection of Spring components on the classpath using annotations rather than XML for metadata, annotation support for lifecycle methods, a new web controller model for mapping requests to annotated methods has increased the development as well as the productivity of the application [19]. Figure 5.10 shows the code snippet of application context used in TMS application.

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:ctx="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"

<context:component-scan base-package="com.sdsu.tms"/>

<mvc:annotation-driven />
```

Figure 5.10. Code snippet of application context (app-config.xml) used in TMS application.

5.7.2 **Step 2**

Once the request comes to the DispatcherServlet it invokes a specific controller based on the handler mapping for further processing. The controller interprets user input and transforms the input into a sensible model.
5.7.2.1 **CONTROLLER LAYER**

Package ‘com.sdsu.tms.controller’: This package contains all the Java classes belonging to controller layer. You will find two controller classes in this package that are used in TMS application.

5.7.2.2 **CLASS DIAGRAM**

Figure 5.11 shows the class diagram for the controller package used in TMS application.

![Class Diagram](image)

**Figure 5.11. Class diagram for the package ‘com.sdsu.tms.controller’ in TMS application.**

Typically, once the request comes to the controller, it interprets user input and transforms the input into a sensible model. In TMS application, there are two controllers’ implementations (see Figures 5.12 and 5.13), which were used to process the client requests. The following are the two controllers used in the application:
@Controller
public class StudentThesisController {
    Logger log = loggerFactory.getLogger(getClass());

    @Autowired
    private StudentThesisService documentService;

    @Autowired
    private DocumentFormValidator validator;

    protected ApplicationContext springAppContext = null;

    @RequestMapping("/searchContacts")
    public ModelAndView searchStudentDetails(
        @RequestParam(required = false, defaultValue = "") String redid,
        @RequestParam(required = false, defaultValue = "") String adname) {
        ModelAndView mav = new ModelAndView("showContacts");
        List<StudentDetails> document = documentService.searchDocuments(redid.trim(), adname);
        mav.addObject("size", document.size());
        mav.addObject("SEARCH_CONTACTS_RESULTS_KEY", document);
        return mav;
    }
}

Figure 5.12. Code snippet of ‘StudentThesisController’ for in TMS application.

@Controller
public class ExtJsStudentThesisController {
    @Autowired
    private StudentThesisService studentThesisService;

    private MetricsService metricsService;

    @RequestMapping(value="/extJSStudent/view.do")
    public @ResponseBody Map<String,?> extends Object> view(@RequestParam int start, @RequestParam int limit,
        @RequestParam(required=false) Object filter) throws Exception {
        try{
            List<StudentDetails> contacts = studentThesisService.getAllStudentDetails();
            //int total = studentThesisService.getTotalContacts();
            //throw new Exception();
            return ResponseProcessor.mapOK(contacts, 22);
        } catch (Exception e) {
            return ResponseProcessor.mapError("Error retrieving Contacts from database.");
        }
    }
}

Figure 5.13. Code snippet of ‘ExtJsStudentThesisController’ used in TMS application.
- ‘StudentThesisController’ is used for processing the entire request coming from the JSP as client.
- ‘ExtJsStudentThesisController’ is used for processing the entire request coming from the ExtJs as client.

The reason behind implementing two different controllers is to know how easy or difficult it is to change the user interface, how much code refactoring is required and how easy or hard it is to integrate two different user interfaces (JSP and ExtJs). We should also analyze how much effect the data persistence layer has on the controller after changing the user interface.

5.7.2.3 ANNOTATIONS USED IN THE CONTROLLER CLASS

- @Controller annotation: The @Controller annotation designates an ordinary (POJO) class as a web controller. The moment the Spring container sees this annotation it knows that this class is a controller for this application and designates all the features that a controller will have.

- @RequestMapping annotation: As mentioned earlier about the generic Handler Mapping, it will map any URL request with *.do extension to a particular controller which can handle the requests. As one can notice, there are two controllers, which can handle *.do URL requests, one, which process the JSP client requests, and another one to process the ExtJs client requests. So, in order to divide these requests with specific controllers, Spring provides a beautiful annotation called @RequestMapping. The @RequestMapping annotation maps the web requests on specific handler classes or handler methods. The above example maps the URL paths “/searchContacts.do” to searchDocuments(), “/extJSStudent/view.do” to getAllStudentDetails().

- @RequestParam annotation: We can map the parameters of any request to a specific controller method using Spring MVC annotation. The @RequestParam annotation binds HTTP request parameters to method in the controller. In the code snippet shown in Figure 5.12, the @RequestParam annotation takes the ‘redid’ and ‘adname’ request parameters from the URL and attach it to String redid, String adname method arguments. For example, the method shown in Figure 5.12 will process the request with following format: /searchContact?redid=SomeValue, where “SomeValue” becomes the “redid” argument. @RequestParam has two other optional attributes: required and default value. ‘required’ is a Boolean that is applied to the value attribute, which defaults to true. When true, then exception will be thrown if the request parameter is null from the URL request string. When required is false, we can avoid exception and the input argument is set to null.

From the class diagram (see Figure 5.11) shows the methods used in the two controllers ‘StudentThesisController’ and ‘ExtJsThesisController’ for processing the client requests. The beauty of using layered architecture is both these Controllers are referencing to
the same service layer (business logic) interface in order to process the request. The reason why controllers are referencing to the service layer was discussed in the coming (see Section 5.7.2). So, in future if anybody wants to change the data persistence layer, there will not be any effect on the controller layer as the controller layer will be unaware of data persistence layer. That is the advantage of using Spring MVC as it allows the accessibility of adding layers in the application. Similarly, if the client wants to change the user interface, for example, changing the user interface from JSP to ExtJs (just by removing the controller in our case), there will not be any code refactoring on the data persistence layer or business logic layer (service layer), as it is unaware of the controller layer. This shows the advantage of Spring MVCs decoupling.

5.7.3 Step 3

The controller layer forwards the request to the service layer and that is where the business logic of the application is applied to the request. It acts as an intermediate object between controller layer and the data tier.

5.7.3.1 SERVICE LAYER

Package ‘com.sdsu.tms.service’: This package contains all the Java classes belonging to service layer. we have only one service layer for TMS application.

5.7.3.2 CLASS DIAGRAM

Figure 5.14 shows the class diagram for service layer package used in TMS application.

As mentioned earlier in Section 5.2, TMS has complex configurations involving two different user interfaces and two different data persistence implementations, if the business logic resides in the controller layer then change in business logic will directly affect the following layers and test for loose coupling, less code refactoring, pluggability and code reusability will fail. It is always better separate the business logic from the web-tier (controller layer). The controller should only handle incoming request and delegate all the response processing to a more abstract layer, which in this case is the Service Layer. This is the reason why we added a separate service layer in the application.
5.7.3.3 ANNOTATIONS USED IN THE SERVICE LAYER

- @Service annotation: All the components are scanned by <context:component-scan> property in the application context file. The service component will be detected by the application via @Service annotation, a business service façade.
Figure 5.15. Code snippet of ‘StudentThesisService’ service object used in TMS application.

```java
@Service
public class StudentThesisServiceImpl implements StudentThesisService {
    Logger log = LoggerFactory.getLogger(getClass());

    @Autowired
    // @Qualifier(value="studentHibernateDao")
    // @Qualifier(value = "studentJDBCDao")
    StudentDetailsDao dao = null;

    @Transactional
    public void save(StudentDetails document, MultipartFile file) {
        transformUserSentFile(file, document);
        try {
            dao.save(document);
        } catch (Exception e) {
            e.printStackTrace();
            throw new ThesisServiceException(e);
        }
    }
}
```

- **@Autowired annotation**: As seen in the earlier chapter, (see Chapter 4, Section 4.9) the advantage of using dependency injection/IoC, @Autowired annotation is used to auto inject the ‘studentJDBCDao’ or ‘studentHibernateDao’ at runtime into any @Autowired field, method parameter or constructor parameter, which matches this type.

- **@Qualifier annotation**: For the metrics, TMS application implements two different DAOs; one with Hibernate implementation and the other with JDBC. In order to implement a particular DAO, the @Qualifier annotation is used which identifies the component by name. We can switch between the components; for example, changing the Qualifier name can do changing from ‘studentHibernateDao’ to ‘studentJDBCDao.’ This works well if we have one service component and more than one DAO components.

- **@Transactional**: The transaction management was well abstracted in Spring across JDBC, Hibernate, and JPA. It integrates well with various data access abstractions.

By looking at the class diagram (see Figure 5.14) of Service layer, we can notice that it exposes the business logic, which performs the necessary CRUD as well as other use case operations in TMS application. This layer has a reference to the DAO Interface by the name ‘StudentDetailsDao’, which will be implemented by ‘StudentDetailsHibernateDao’ and ‘StudentDetailsJDBCDao’ in the data persistence layer. So, if the client wants to change the backend database then there won’t be any affect on the service layers as it is unaware of actual DAO implementation. It also acts as a shield for the controller layer, so change in data persistence layer won’t affect the controller/web-tier layer. Another advantage of Spring annotated service layer is it is minimum transactional life span. In the service layer, all the
methods represent transactional unit of work this means one method call and many POJOs, their interaction will be performed under single transaction. As an added benefit, transaction configuration can be centralized by moving the transactions to a single layer makes.

5.7.4 Step 4

The service layer where the business logic resides needs to talk to the DAO layer for the data retrieval. These DAO classes are used extensively for communicating with the data tier.

5.7.4.1 DAO Layer

Package ‘com.sdsu.tms.dao’: This package contains all the necessary DAO classes used in the TMS application.

5.7.4.2 Class Diagram

Figure 5.16 shows the class diagram of DAO package used in TMS application.

![Class diagram for 'com.sdsu.tms.dao' package used in TMS application.](image)
As discussed in Section 5.2, TMS application will have two persistent and query API in order to implement Hibernate and JDBC. As Spring does not provide its own persistence framework, DAO design pattern (core JEE pattern) was implemented; it states that all persistent operations should be performed via DAO classes. These classes are used exclusively for communicating with the data tier. The purpose of this pattern is to separate persistence-related code from the application’s business logic, which makes more manageable and maintainable code, letting the developer change the persistence strategy flexibly, without changing the business rules or workflow logic. Figure 5.17 shows the code snippet of ‘StudentDetailsDao’ interface used in TMS application.

```java
public interface StudentDetailsDao {
    @Transactional
    public abstract void save(StudentDetails document);
    @SuppressWarnings("unchecked")
    @Transactional
    public abstract List<StudentDetails> list();
    @Transactional
    public abstract StudentDetails get(Integer id);
    @Transactional
    public abstract void remove(Integer id);
    @Transactional
    public abstract void update(StudentDetails document);
    @Transactional
    public abstract List<StudentDetails> findByCriteria(String rid, String adname);
    @Transactional
    public abstract List<StudentDetails> findAll();
}
```

Figure 5.17. Code snippet of ‘StudentDetailsDao’ used in TMS application.

Figures 5.18 and 5.19 are the code snippets of ‘studentDetailsHibernateDaoImpl’ and studentDetailsJDBCDaoImpl’ classes implementing ‘StudentDetailsDao’ interface in TMS application.

5.7.4.3 ANNOTATIONS USED IN HIBERNATE DAO (‘STUDENTDETAILSHIBERNATEDAOIMPL’)

- @Repository annotation: This class will be annotated with @Repository annotation, which specifies the class as DAO. The Spring container auto detects this stereotype annotation by using <context:component-scan> property, in the same way as @Controller, @Service components. All the exceptions in this class will be handled by decorating the class with @Repository annotation.
Figure 5.18. Code snippet of ‘studentDetailsHibernateDaoImpl’ implementing ‘StudentDetailsDao.’

```java
import java.util.List;

@Repository("studentHibernateDao")
public class StudentDetailsHibernateDaoImpl implements StudentDetailsDao {

    @Autowired
    private SessionFactory sessionFactory;

    /* (non-Javadoc)
     * @see com.sdsu.tms.dao.StudentDetailsDao#save(com.sdsu.tms.model.StudentDetails)
     */
    @Transactional
    public void save(StudentDetails document) {
        Session session = sessionFactory.getCurrentSession();
        session.save(document);
    }
}
```

Figure 5.19. Code snippet of ‘studentDetailsJDBCDaoImpl’ implementing ‘StudentDetailsDao.’

```java
@Repository("studentJDBCDao")
public class StudentDetailsJDBCDaoImpl implements StudentDetailsDao {

    private static final Logger log = LoggerFactory.getLogger(StudentDetailsJDBCDaoImpl.class);

    private final String INSERT_SQL = "insert into 
        + TMSConstants.TMS_TABLE_NAMES.STUDENTDETAILS 
        + (adname, content, content_type, created, filename, name, redid, topic, email) values (?, ?, ?, ?, ?, ?, ?, ?);"

    private final String DELETE_SQL = "delete from 
        + TMSConstants.TMS_TABLE_NAMES.STUDENTDETAILS + " where id=?;"

    private final String UPDATE_SQL = "update 
        + TMSConstants.TMS_TABLE_NAMES.STUDENTDETAILS 
        + set adname=?, content=?, content_type=?, created=?, filename=?, name=?, redid=?, topic=?, email=? where id=?;"

    private final String DISPLAY_ALL = "select * from 
        + TMSConstants.TMS_TABLE_NAMES.STUDENTDETAILS;"

    private final String SEARCH_CRITERIA = "select * from 
        + TMSConstants.TMS_TABLE_NAMES.STUDENTDETAILS 
        + " where redid=? or adname=?;"

    private final String FIND_BY_PKKEY = "select * from 
        + TMSConstants.TMS_TABLE_NAMES.STUDENTDETAILS + " where id=?;"

    JdbcUtils jdbcUtil = null;

    public StudentDetailsJDBCDaoImpl() {
        jdbcUtil = new JdbcUtils();
    }
}
```

- **@Transaction annotation:** By decorating the class with @Transaction annotation, the public methods of the class will participate in transactions. The @Transaction on methods indicates which of the transaction is read only.

- **@Autowired:** To use Hibernate in an application, ‘SessionFactory’ is required, which instantiates session objects (see Chapter 3, Section 3.3). By using ‘@Autowired’ annotation the SessionFactory instance is automatically injected to the class. The Spring framework manages the allocation of Hibernate resources, such as SessionFactory.
By looking at the class diagram (see Figure 5.16), we can see both ‘studentDetailsJDBCDaoImpl’ and ‘studentDetailsHibernateDaoImpl’ implementing the single ‘StudentDetailsDao’ interface (Java best practice). Since, the service layer (business logic layer) always talks to ‘StudentDetailsDao’, the class implementing the ‘StudentDaoInterface’ can use any persistence technology without affecting the service as well as the controller layers. Spring can inject (wire) different implementations that use native Hibernate, JDBC, etc., objects. So, this layer provides the flexibility to change the back-end database with minor changes within this layer.

5.7.4.4 HIBERNATE WITH SPRING CONFIGURATION

Hibernate is a powerful framework for persisting the data. Spring, on the other hand supports IoC. The Spring framework provides excellent integration support for ORM framework-Hibernate. This section focuses on Hibernate configuration and how the application gets resource management and transaction support via Spring.

This application has a separate Hibernate configuration file called db-config.xml, which is defined, in applicationContext.xml file (see Figure 5.20). The reason for deliberately defining it in separate file instead of defining it in web.xml is to ease the readability of the file and to make the application context short. Because as the configuration increases with increase in functionality, editing or changing the properties can be conveniently handled for testing purposes (see Section 5.7, The Application Context).

```xml
<!-- Configures Hibernate - Database Config -->
<import resource="db-config.xml" />
```

Figure 5.20. Code snippet showing db-config.xml for database configuration shows the configuration in app-config.xml.

The application can access the data from any database independently by using Hibernate. This makes the application to overcome the use of low level JDBC coding (connection management, connection pooling, dealing with statements and result sets). For Hibernate, all the low level coding like configuring a data source is defined in the XML file. As Hibernate is an independent framework, the application that accesses the database using
Hibernate has to depend on Hibernate APIs like Configuration, SessionFactory and Session. Moreover, the application manually maintains and manages these objects. But with Spring integration (incorporates lots of IoC convenient features), as Spring externalize the state of the object from the application code, Hibernate objects can be used as Spring Beans (see in Figure 5.21) that can access all the functionalities that Spring provides.

```xml
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">value=com.mysql.jdbc.Driver</value>
    <property name="url">value=jdbc:mysql://localhost:3306/HibernateDemo?profileSQL=true</value>
    <property name="username">value=root</value>
    <property name="password">value=</property>
</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="dataSource" ref local="dataSource"/>
    <property name="packagesToScan" value="com.sdsu.tms.model"/>
    <property name="hibernateProperties">
        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
        <prop key="hibernate.connection.useServerPrepStmts">true</prop>
        <prop key="hibernate.jdbc.batch_size">0</prop>
        <prop key="hibernate.dialect.update">true</prop>
    </property>
</bean>

</beans>

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref local="sessionFactory"/>
</bean>
```

Figure 5.21. Properties declared in db-config.xml for Hibernate configuration used in TMS application.

5.7.4.4.1 SessionFactory

The DAO (‘studentDetailsHibernateDaoImpl’) implementation requires a Hibernate SessionFactory (see Chapter 3, Section 3.3) instance (it has all the Hibernate mapping information) to be autowired by Spring. As discussed in Chapter 3, Hibernate uses SessionFactory for concurrency by obtaining session instances from SessionFactory. SessionFactory should be configured as singleton (see Figure 5.21). Here we configure the
SessionFactory Bean using Spring’s AnnotationSessionFactoryBean, which itself a factory for Hibernate session factories. We use the packageToScan Bean property to tell Spring where to look for the Hibernate entities (i.e. classes annotated with @Entity).

### 5.7.4.4.2 Transactions

<tx:annotation-driven> (see Figure 5.21) enables declarative transaction management with the help of ‘@Transactional’ annotation. By ‘txnManager’, Spring uses ‘HibernateTransactionManager’ class for delegating transaction management responsibility to the Hibernate.

### 5.7.4.4.3 Exception Handling

By decorating the class with ‘@Repository’ annotation all the exceptions thrown from Hibernate will be translated into Spring’s DataAccessException hierarchy. This makes the data access exceptions to be consistent irrespective of the persistent technology as well as we can abstract these exceptions from the data layer.

### 5.7.4.4.4 The StudentDetails Class

After the above configurations, we should create a ‘StudentDetails’ class for holding and retrieving data from the ‘STUDENT.Details’ table. The column names in the table ‘STUDENT.Details’ is mapped to instance variables of the Java model class. CS_TMS uses two tables, one for the student details and other for the metrics (see Section 5.5). Let us see the code listing for ‘StudentDetails’ class (see Figure 5.22).

Typically in traditional Hibernate configuration, after creating the table in the database and corresponding Java class in the application layer, the ‘STUDENT.Details’ table should be mapped to Java class and the column names in the ‘STUDENT.Details’ table should be mapped to the Java variables in the ‘StudentDetails’ class. This mapping is done in ‘HibernateMapping’ files (*.hbm.xml).

Because of Hibernate 3 annotations (thanks to Java 5.0), which provides a powerful metadata and elegant API to simplify the Object and relational table mapping. We can eliminate the ‘HibernateMapping’ file by using below mentioned annotations that directly map the table with the Java class.
@Entity
@Table(name="STUDENT_DETAILS", uniqueConstraints = @UniqueConstraint(columnNames = {"redid"}))
public class StudentDetails {
    @Id
    @GeneratedValue
    @Column(name="id")
    private Integer id;
    @Column(name="name", nullable = false)
    private String name;
    @Column(name="topic", nullable = false)
    private String topic;
    @Column(name="filename")
    private String filename;
    @Lazy
    @Column(name="content", nullable = false)
    private byte[] content;
    @Column(name="content_type")
    private String contentType;
    @Temporal(TemporalType.DATE)
    @Column(name="created")
    private Date created;
    @Column(name="redid", nullable = false)
    private String redid;
    @Column(name="adname", nullable = false)
    private String adname;
    @Column(name="email", nullable = false)
    private String email;

Figure 5.22. Code snippet showing ‘StudentDetails’ as the ‘Entity’ used in TMS application.

- @Entity annotation: The @Entity annotation marks the class as an Entity Bean. So the class should at least have a package scope no argument constructor.

- @Table annotation: The application knows the database table used for persisting the data by using @Table annotation. The name of the database table can be declared as an attribute called ‘name’ on the annotation. By default, Hibernate uses the Java model class name as the database table name if we do not mention the name attribute in the annotation. However, unique constraints, schema and catalog options can be declared on the @Table annotation as well.

- @Id annotation: Using @Id annotation can specify the table identifier property for the table. By default, Hibernate uses @Id annotation as default access strategy for mapping. If @Id is over the field, then @Id should be placed on the getter method of the field.

- @GeneratedValue: To uniquely identify each row, generating a primary key is very important in a relational table. Using ‘@GeneratedValue’ annotation can generate this primary key. By default, Hibernate generates its own primary key AUTO.

- @Column: The application knows the mapping between the column and the property or filed (Java model class) by using @Column annotation. Here, ‘redid’ property is
mapped to ‘redid’ column in the “STUDENTDETAILS” table. If @Column is not specified then Hibernate uses property as the column name.

- @Transient: If we do not want the property to be mapped, use @Transient annotation to exclude the property.

5.7.4.5 JDBC with Spring Configuration

Integrating JDBC with Spring is entirely opposite to Hibernate Spring configuration. Unlike, Hibernate, all the connection management, connection pooling, dealing with statements and result sets are to be manually handled by the developer. Storing and retrieving the data is done entirely by using pure SQL statements. But Spring JDBC provides some advantages compared with standard JDBC. It allows automatic database connection clean ups and converts standard JDBC SQLExceptions into RuntimeExceptions, etc. In TMS application, standard JDBC configuration is implemented, as using Spring JDBC template does not bring much difference in terms of JDBC’s boilerplate coding neither it supports ORM. The motivation was to see the advantages of Hibernate over JDBC.

General steps include:

1. Load the RDBMS specific JDBC driver because this driver actually communicates with the database.
2. Open the connection to database, which is then used to send SQL statements and get results back.
3. Create JDBC Statement object. This object contains SQL query.
4. Execute statement that returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
5. Process the result set and maps it to Java model class.
6. Close the connection.

Figures 5.23, 5.24, and 5.25 show the data source declaration in JDBC implemented code used in TMS application.

5.7.5 Step 5

After configuring the application for handling incoming client requests, business logic and data persistence, the next big challenge is to handle the cross cutting concerns of the application. This section will show how TMS application handled the Cross cutting concerns without affecting the subsequent layers.
5.7.5.1 PACKAGE CONTENTS

Package ‘com.sdsu.tms.aop.aspect’: This package will have all the AOP classes required to handle cross cutting concerns in TMS application.

5.7.5.2 CLASS DIAGRAM

Figure 5.26 shows the class diagram of AOP package used in TMS application.
Figure 5.25. Typical way of mapping column name from the result set to Java model class.

```java
private StudentDetails mapResult(ResultSet rs) throws SQLException {
    StudentDetails sd = new StudentDetails();
    sd.setAname(rs.getString("aName"));
    sd.setId(rs.getInt("id"));
    sd.setContent(rs.getBytes("content"));
    sd.setContentType(rs.getString("content_type"));
    sd.setCreated(rs.getDate("created"));
    sd.setFilename(rs.getString("filename"));
    sd.setRedid(rs.getString("redid"));
    sd.setName(rs.getString("name"));
    sd.setTopic(rs.getString("topic"));
    System.out.println(sd.toString());

    return sd;
}
```

Figure 5.26. Class diagram of ‘com.sdsu.tms.aop.aspect’ package used in TMS application.

As you can see in Figure 5.7, Section 5.6, there are some underlying transparent layers implemented in the application, called ApplicationMetricsAspect, StudentMailService and ApplicationMetricsDao. These layers are used to handle cross cutting concerns like logging, email service and the performance metrics (see Figure 5.22) between two different
implementations (Hibernate and JDBC). Generally, implementing cross cutting concern will adversely affect existing business logic, increases the code complexity and duplication. Adding these concerns in the actual business logic which is to perform CRUD operations in our application, will be lost in the crosscutting concern code for logging, email service and performance metrics code and also the service layer (where business logic resides) will get bloated with complex coding. So, in order to cleanly segregate from the business logic, Spring provides a good solution called Spring AOP (see Chapter 4, Section 4.7). By segregation, the business logic doesn’t even know that there are some external implementations that will use business logic methods to handle these concerns.

TMS application has a separate table ‘STUDENT_DAO_METRICS’ which contains all the metric information. A separate DAO (‘ApplicationMetricsDao’) has been implemented to save these records into the table.

5.7.5.3 HOW SPRING AOP DEALT THE CROSS CUTTING CONCERN IN TMS

The pointcut expression tells the AOP framework when and where to intercept. The ‘studentCrudMethods’ pointcut has the regular expression, which directs AOP to intercept on the execution of all methods in the classes in com.sdsu.tms.dao package or any of its subpackages with any return type and with any number of arguments to the method. This basically allows interception of all crud operations on the two DAO classes.

Figure 5.27 shows the code snippets of aspect class with pointcut expression and around advice used on method.

The @Around advice as shown in Figure 5.27 directs the Spring AOP what to do when intercepted. This @Around advice does some house keeping chores of TMS applications including logging, collecting metrics of the application and sends emails to the user.

1. Once the ‘save’ request comes from the service layer, the ApplicationMetricsAspect (Spring AOP class) intercepts the request externally (neither the ‘StudentThesisService’ nor the ‘StudentDetailsHibernateDao’ will not be knowing about this interception) based on the around advice aspect configured (see Chapter 4, Section 4.7) it starts the timer to calculate the time taken by the ‘save’ request to save the details in the database.
@Aspect
public class ApplicationMetricsAspect {

    Logger log = LoggerFactory.getLogger(getClass());

    @Autowired
    ApplicationMetricsDao fileDao = null;

    @Autowired
    StudentMailService mailSender = null;

    @Pointcut("execution(public void com.sdsu.tms.service.StudentThesisServiceImpl.save(..))")
    public void fileUploadDetails() {
    }

    @Pointcut("args(com.sdsu.tms.model.StudentDetails))")
    public void withDocumentAsArg() {
    }

    @Pointcut("execution(* com.sdsu.tms.dao..*(..))")
    public void studentCrudMethods() {
    }

    @Around("studentCrudMethods() && args(studentDetails,...)")
    public Object doBasicProfiling(ProceedingJoinPoint pjp, StudentDetails studentDetails) throws Throwable {
        final MethodSignature signature = (MethodSignature) pjp.getSignature();

        Method targetMethod = signature.getMethod();
        String operationName = targetMethod.getName();
        Class<?> serviceClass = targetMethod.getDeclaringClass();
        String pointCutExpression = pjp.toString();

        log.debug("PointcutExpression String: "+ pointCutExpression);

        long start = System.currentTimeMillis(); // start stopwatch
        log.debug("Going to call the method.");

        Object output = pjp.proceed();

        log.debug("Method execution completed.");

        long elapsedTime = System.currentTimeMillis() - start; // stop stopwatch
        log.debug("Method execution time: "+ elapsedTime + " milliseconds.");

        StudentDaoMetricRecord rec = createProfilingRecord(
            pjp.getTarget().getClass().getSimpleName(), operationName, studentDetails.getName(), new Date(),
            elapsedTime, getFileSizeInKB(studentDetails));

        fileDao.insertProfilingRecord(rec);

        if(TMSConstants.STUDENTDETAILSDAO_METHOD_NAMES.SAVE.equals(operationName)) {
            sendEmail(rec, studentDetails);
        }

        return output;
    }
}

Figure 5.27. Aspect class addressing cross cutting concern in TMS application.
2. Once the ‘response’ from the database comes back from the DAO layer, the around advice method waits for this response to stop the timer and calculates the elapsed time. Simultaneously, after receiving the save ‘response’ from the DAO layer, the ApplicationMetricsAspect will invoke the Spring mail service (Chapter 4, Section 4.8) to send a notification (email) to the Gmail server.

3. All the details including the elapsed time, execution date, operation name, serviceName, User and file size are captured and saved in the ‘STUDENT_DAO_METRICS’ table through ‘ApplicationMetricsDao.’

### 5.8 Step 6: The Presentation Layer (The View)

This is the presentation layer, called as the view of the web application. Spring MVC provides a way to address views; it uses view resolvers to render models (response coming from the service layer to the controller) in a browser without holding to a specific view technology [14]. Spring provides support for JSPs, ExtJs, Velocity templates, etc. ViewResolver provides a mapping between view names and actual views [14]. The controller normally returns a logical view name, which a view resolver resolves to a particular view technology [14]. For instance, TMS uses JSPs as views that are actually processed via the servlet engine. This is handled by InternalResourceViewResolver (see Figure 5.28), which will issue an internal forward to the actual view.

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
  <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
  <property name="suffix" value=".jsp"/>
</bean>
</beans>
```

Figure 5.28. Configuring ViewResolver in application context (app-config.xml). For TMS application ViewResolver is configured in default application context configuration file ‘thesis-manager-servlet.xml.’

Generally, it contains all the technologies required to render the user interface. It includes the layouts, CSS and images along with the front-end technologies, JSPs and ExtJs that are visible to the users [1]. These components will use libraries and functionalities provided by the middle layer interface (service layer) to display appropriate content according to requests. The web interface can be customized and designed for different purposes.
For this application I have designed two front-end interfaces (views) for displaying information in the table (all the previous persisted records) along with the options of search, create, delete, edit, upload and download. The reason behind implementing two views is to check the supportability, compatibility, switch ability and to know how Spring MVC handles JSPs (server side) and ExtJs (client side) technologies.

### 5.8.1 JSP as the User Interface

Forms are an important way of making the web interactive. For JSPs, handling the form is easy. Information from the form is submitted either by HTTP GET or POST (‘HttpRequest’) request methods. And also JSPs and servlets can send information about form parameters, including names and values.

- **Step 1:** User enters the URL: localhost:8180/thesis-manager (see Figure 5.29). Figure 5.30 shows the screen which displays all the details to the user.

- **Step 2:** 5.8(a) The Search System – search made easy! The search button provides easy accessibility to the records. The users can enter the red-id or advisor’s name to search his/her records or any particular record and can access the details along with file download option.

  Figure 5.31 shows the header with ExtJs view, TMS metrics, adding a new thesis and search options. Figure 5.32 shows the result displayed after searching the record with ‘red-id.’ Figure 5.33 shows the option for searching the record with ‘Advisor Name’ and Figure 5.34 shows the results displayed after searching the record with ‘Advisor Name.’

- **Step 3:** 5.8(b) Creating new record. User can create a new record (see Figure 5.35) by giving all the required information in this page and can insert the record into the database. And the user will be prompted with any form errors using validations.

  Figure 5.36 shows newly inserted record in the home page information table.

  Once the student record is saved in the database, an email notification (see Figure 5.37) will be sent to the given student email address. As of now only Gmail users can receive this notification email. In further enhancements, other email APIs will be included.

- **Step 4:** 5.8(c) Editing an existing record. User can edit a particular record (see Figure 5.38) from the table by clicking edit link. The changed record shown in Figure 5.39 will be immediately inserted into the database and is reflected in the home page table.

![Figure 5.29. Enter the URL: localhost:8180/thesis-manager to access TMS application.](image)
Figure 5.30. TMS application’s homepage displayed after entering the URL showing the entire previous student records.

Figure 5.31. Searching record with ‘red-id’ in TMS application.

Figure 5.32. Search results after entering the key ‘red-id.’

Figure 5.33. Searching record with ‘advisors name’ in TMS application.

Figure 5.34. Search results after choosing the ‘advisor’s name.’
Figure 5.35. Screens to add new document along with validations.
Figure 5.36. Newly inserted record in the home page information table.

Figure 5.37. Gmail notification in TMS application, a new email notification will be sent to the student’s Gmail account after successfully saving their records.

Figure 5.38. Editable screen in TMS application, popping out after clicking edit link in the home screen-information table.
Figure 5.39. New record after editing the information, reflected in the home screen table.

- Step 5: 5.8(d) Deleting a record – a simple delete. User can delete his/her record from the database and is removed from the database (see Figure 5.40).
- Step 6: 5.8(e) Uploading and downloading thesis dissertation document – then and there itself! Figure 5.41 shows upload option to upload the document into the database. Figure 5.42 shows the option of opening or saving the document in the system.

Figure 5.40. Deleting a record in TMS application, caution window popup before deleting the record.

Figure 5.41. Option to upload the document from the system.
5.8.2 ExtJs as the User Interface

The CRUD operations shown in Figure 5.41 can be done in a more ajaxified way by using ExtJs as the end client. ExtJs is a pure JavaScript framework that supports Ajax, DHTML and DOM scripting. It uses JSON (Java Script Object Notation) for sending and receiving the data from the server. Ajax makes the interface more responsive and in a request instead of sending the whole page (like JSP), a section of the page is sent to the server to process. Because of this the response time, the traffic to and from the server can be reduced. All the Ajax enabled browsers can render this view.

Figure 5.43 shows the homepage for ExtJs View, user can add or delete their records by clicking add or delete buttons. Each row is editable and so there is no need to add edit buttons to each row unlike JSP view.

Figure 5.44 shows add and edit screens in ExtJs view of TMS application.

5.9 Observations

One of the criteria for developing this application is to observe the advantages of Hibernate over JDBC, apart from code complexity, performance differences were also perceived. So, TMS application was implemented both in Hibernate as well as JDBC in DAO
5.10 COMPARISONS: HIBERNATE VS. JDBC

In order to see the performance differences between Hibernate and JDBC, the application is implemented with both Hibernate and JDBC DAOs in the DAO layer. For this, the application performs five ‘save’ operations with different file sizes. Figure 5.45 and Figure 5.46 show the five ‘save’ operations metrics profiling implemented by Spring AOP.
As we can see, the time taken for saving the record in the database increased linearly with increase in file size for both Hibernate and JDBC. And Hibernate is little faster than JDBC if we consider the average execution time and average file size. Apart from performance differences, Table 5.2 illustrates the basic differences between Hibernate and JDBC.
Table 5.2. Difference between Hibernate and JDBC

<table>
<thead>
<tr>
<th>Hibernate</th>
<th>JDBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Powerful, flexible and reliable ORM tool to map Java classes to database tables.</td>
<td>Developer has to map an object model’s data to its corresponding schema by writing low level JDBC code which increases the code complexity.</td>
</tr>
<tr>
<td>Takes care of this mapping either by XML files or annotations.</td>
<td>The mapping of Java objects with database tables and vice versa conversion has to be written manually.</td>
</tr>
<tr>
<td>There is no need to write the code explicitly by the developer to map database tables tuples to application objects during interaction with RDBMS.</td>
<td>Supports only native SQL, developer has to find out an efficient way to query larger complex data from the database.</td>
</tr>
<tr>
<td>Other than SQL, Hibernate provides powerful HQL for querying large complex data. Data manipulation is easy with HQL.</td>
<td>Developer has to handle result set and convert it into Java objects.</td>
</tr>
<tr>
<td>Lines of code required for managing and maintaining object-table mapping is very less when we use Hibernate. The result set returns to application in the form of Java Objects.</td>
<td>Caching is maintained by hand-coding.</td>
</tr>
<tr>
<td>Provides cache, which is set to application workspace. Relational tuples are moved to cache as a result query. It improves performance by reducing the number of hits to the database if the client reads same data many times.</td>
<td>Requires the developer to manually update the SQL code, which is more prone to errors and difficult to debug.</td>
</tr>
<tr>
<td>Changing the table properties is extremely easy, Hibernate automatically detects the new changes and creates a new set of SQL statements accordingly.</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6

DEVELOPMENT MANUAL

6.1 OVERVIEW

This chapter provides information about setting up the development environment for TMS application.

6.2 DEVELOPMENT ENVIRONMENT

There is a specific development environment set up for every JEE application to install on the system. The steps to set up development environment are given in any JEE/Spring MVC/Hibernate application development websites. This application can be developed on any platform with Java compatibility. The steps for setting development environment are:

Prerequisites:

- IDE: Install Eclipse IDE for Java EE Developers.
- Software Development Kit: Install Java Development Kit (JDK 5 or 6).
- IDE Plugin: Install Apache Build Manager – Maven (M2) plug-in for Eclipse.
- Build Tool: Download and install Maven 3.0 from http://maven.apache.org/download.html
- Database Server: Installing MySQL Database for persisting the application data.
- Application Server: Installing Application Server Tomcat v7.0, plug-in is available for eclipse, you can also look for online help http://www.coreservlets.com/Apache-Tomcat-Tutorial

6.3 APACHE MAVEN – BUILD TOOL MADE EASY

A project management and comprehensive tool called Maven was used to build this project; Maven can manage a project’s build, reporting and documentation from a central piece of information [21]. Maven takes care of complete development effort in the shortest period of time. Features of Maven:

- Easy build process
- Uniform build system
- Better project quality information
- Easy accessibility to project guidelines
- Easy migration to new tools

Project object model (POM) is used to build the project and a set of plug-ins that are shared by all projects using Maven, providing a uniform build system [21]. Maven saves a lot of time for developing a project and once we get acquainted with this tool it automatically saves a lot of time when trying to navigate many projects [21]. All the necessary documentation including, download and install Maven, quick start and using Maven are provided in Apache Maven website: http://maven.apache.org/run-maven/index.html. After successful installation, go to → Terminal/Command prompt and run the command: MVN-version (see Figure 6.1).

![Figure 6.1. Maven 3.0.3 running in Mac OS X.](image)

Check for Java version installed in your system by typing the command: Java-version in your terminal/command prompt (see Figure 6.2). Figure 6.2 shows how to start Apache Tomcat application server in the system. Figure 6.3 shows Apache Tomcat 7.0 application server running in Mac OS X.

![Figure 6.2. How to start Apache Tomcat application server in the system.](image)

### 6.4 APPLICATION CONFIGURATION

- Install Maven plug-in from eclipse, Menu → Go to Help → Market place → New Software option is available for plug-in download.
After installation, open a new Maven Project from file menu and choose Maven ➔ Maven Project.

Choose, the Maven archetype suitable to Spring MVC and Hibernate application, available online.

By convention, enter a group id for the artifact, and packing as WAR.

Following project file structure will be created with pom.xml.

Create necessary packages and folders as show in the file structure with your project name.

Add Java classes inside these packages according to their functionality, it is always better to divide your Java files according to the functionality, example all the DAO layer Java files should be placed under one package.

Required jar dependencies should be added in POM.XML, the .jar files are available from Maven repository http://mvnrepository.com/.

After downloading all the necessary jar files, right click on the project, select ➔ Maven ➔ update configuration for consistency in the Maven repository in your local system.

See your project facets, right click on your project, properties ➔ project facets ➔ and choose dynamic web project, Java version.

Once it has been done, right click on the project select, run as ➔ Maven install for building the project.

All the code compilation will be done in the above process and a new target folder will be created in your project, with the required WAR.

We can deploy the WAR file into Tomcat V7.0 manually by deploying the newly created WAR in to the Application Server, check for more information online or we can use eclipse plug-in to run the project in exploded format which will deploy the project in to the server.
Note: If you want to run the project in eclipse exploded format, copy all jar files into lib folder under WEB-INF.

- After successful deployment, open your browser and enter the following URL http://localhost:8080/project-name 8080 is the default port number for Tomcat Apache Server, you can change it if you want, see for more information online.

Figure 6.4 shows the TMS application’s complete package structure after adding all the necessary Java class, JSPs, .jar files and ExtJs libraries.
Figure 6.4. Project Structure according to Maven archetype in Eclipse IDE.
CHAPTER 7

CONCLUSION

The main purpose of my thesis research is to study the advantages of Spring and Hibernate open source technology frameworks in developing a web application. These are the conclusions made after implementing the TMS application with:

7.1 SPRING FRAMEWORK

Spring framework made the application configuration and development simple, easier and faster because of the following reasons:

1. POJO-based programming:
   - I was able to develop the application with much ease and simplicity because of Spring’s support for POJOs and that is why all the entity Beans used in the TMS application were POJOs.
   - It made TMS application code loosely coupled that is not tightly coupled with the framework because of Spring’s non-invasiveness.

2. Configuration made easy with annotations:
   - I was able to replace Spring XML configuration with annotation-based configuration as Spring adapted Java 5.0 features to the fullest, especially annotations, which enabled the configuration of Beans inside the Java source file itself; makes the code readable, understandable and manageable.
   - Spring’s core, Spring MVC, Spring AOP, testing annotations made the application configuration more productive for example, I configured data source, transactions, TMS mails service, internationalization, and TMS metrics management in the application configuration file.

3. The power of dependency injection:
   - I found out that because of Spring’s dependency injection the application components does not have to instantiate and manage life cycle of dependencies (by calling constructors which results in compile time locking and hampers unit testing).
   - I was able to develop the code in a much simpler and easier way as I do not have to write the code for wiring the dependent objects because lightweight IoC will inject the application dependencies at runtime.
   - I was able to easily swap between different data persistence implementations (Hibernate and JDBC) because of Spring’s DI (Dependency Injection).
• I was able to achieve loose coupling in the application because of Spring’s IoC as objects give their dependencies instead of creating or looking for other dependent objects.

4. Easy transaction management:
• I was able to separate the transaction from the business logic and was able to reduce the number of lines of code by enabling Spring’s declarative transaction management.
• It relieved me from manually handling transactions.
• I was able to easily switch between different transaction implementations as it allows the pluggable transaction managers that are configured via annotations or with XML configuration.

5. Easy access to service and data persistence layers:
• I did not use any ORM application, because ORM is handled by Hibernate that is well coordinated with Spring.
• Spring provided a reliable framework for data access, whether it is with JDBC or an ORM mapping tool like Hibernate.
• I was able to effectively organize the middle layers like service and Data Access object layers.
• I was able to abstract the database calls and was easily done by adding DAO layer.
• I was able to externalize the business logic code from the controller layer by adding service layer that acted as intermediate object between controller and DAO layer. So, change in the database will not be having much effect on controller layer and vice versa.

7.2 SPRING MVC

1. Uses complete functionality of Spring:
• I was able to configure model and the controller layer easily by Spring MVC that utilized the benefits of Spring’s dependency.
• I was able to easily configure the Spring MVC controllers via Spring’s IoC using ‘@Controller’ annotation.

2. Easy URL and model attribute mapping by Spring MVC annotations:
• I was able to use URL mapping for the controller with much easier and simpler way; mapping the web requests to a specific handler class or handler method was done by using Spring MVC annotations.
• Since I have used Spring MVC annotation the mapping web requests on to specific controller method was made easy.
I was able to map the data entered in the form as an object and was referenced easily by mapping a model attributes to the specific, annotated method parameter by using `@ModelAttribute` annotation.

3. Spring MVC support for easy form input validations:
   - I was able to validate the form input data with much lesser code as Spring provided the strong input validation mechanism to perform form input validations within Spring MVC.
   - I was able to achieve these validations by using `BindingResult` which that is passed as a URL request parameter on which validations are checked with the support of inbuilt validation methods at the controller level.

4. Spring’s multipart (file upload) support.
   - I was able to upload files easily in TMS application because of Spring’s out of the box support for MultipartResolver that handled file uploads in web application.
   - Configuring `CommonsMultipartResolver` was easy and I am able to define maximum upload file size just by declaring a property of MultipartResolver configured in application context file.

5. View agnostic:
   - More views can be used with Spring MVC as it supports different view technologies like JSP, ExtJs, Velocity, and Tiles, etc. I was able to easily integrate ExtJs and JSP view technologies in TMS application.
   - It provided simple but powerful tag library, and I was able to use these tag libraries for declaring arguments to JSP.

### 7.3 SPRING AOP

1. Easy segregation of business logic and cross cutting concerns:
   - I was able to handle TMS Applications cross cutting concerns easily with less coding by AOP.
   - I was able to achieve TMS applications logging, email service and performance metric details with simple AOP aspects, by delegating the functionality (concern) across the methods without affecting the actual business and data access layers.
   - There was no code plumbing required for adding aspects.
   - I was able to easily add AOP without changing the code.
   - More importantly, I was able to decouple business object code with the framework.

2. Readable and understandable AOP aspects:
   - As a part of Spring AOP, the pointcut expression I have used were simple regular expressions that directed AOP when and where to intercept the methods.
• Defining what to do after interception was easy and understandable by using @Around advice annotation.

### 7.4 Hibernate Framework

Hibernate not only took care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provided the query and retrieval facilities.

1. Easy integration of Hibernate with Spring:
   • I was able to externalize the state of the object from the application code with Spring integration that incorporated lots of IoC convenient features. So, I was able to use Hibernate objects as Spring Beans that accessed all the functionalities that Spring provides.

2. Object oriented way to express queries:
   • I used the Hibernate supported criteria API that is a nice object oriented way of expressing queries.

3. Easy dynamic search functionalities:
   • I was able to build a powerful and elegant on-the-fly dynamic search functionality queries by using Hibernate’s criteria API on Hibernate persisted databases.
   • Searching with key was simple; criterion API provided different functionalities for search.

4. Easy database schema maintenance:
   • I was able to easily maintain the database schema with Hibernate as it comes with a great tool that generated the database schema automatically by just declaring a property in Hibernate configuration file.
   • I also had the option of generating brand new database each time the application started. I was able to achieve this by defining the property in Hibernate configuration file.

5. Annotation-based configuration:
   • Annotation-based configuration wiped out most of the traditional way of configuring the Hibernate in XML files.
   • I was able to configure the application’s data persistence easily with Spring and Hibernate annotations and the XML configuration.
   • I made every persistent POJO class as an entity just by decorating the POJO class by Hibernate annotation.
   • I was able to defining the table, unique constraints and the columns at the class level itself by Hibernate annotations. I was also able to define schema and catalog as attributes within these annotations.
   • I was able to easily define the property, which should be persisted as Blob, or Clob by decorating the property with Hibernate annotation.
• I was able to automatically generate the identifier property simply by using Hibernate annotation.

6. No boilerplate coding:
• I was able to focus more on the business logic rather than plumbing work. This I believe improved the applications productivity.
• I found it rather easy to inject the SessionFactory, which is required for Hibernate via Spring dependency injection.

7. Separation of data source configuration:
• I was able to reduce the number of lines of code for data persistence as there is separate data source configuration, which made the system more understandable, and also it made code refactoring was easy.
• I was able to easily switch to other relational database. It required few changes in data source configuration in the Hibernate configuration file.

8. Performance with different file upload sizes:
• TMS application has both Hibernate and JDBC data persistence implementations. Using Spring AOP, I was able to capture the execution time and the file size for each file to get uploaded in the database.
• I observed that there was a linear increase in execution time as the file size is increased both in Hibernate and JDBC. However, JDBC took little more time to upload the files than the Hibernate.

9. Good documentation:
• Since it is totally object oriented, learning curve for me was fast and easy.
• I found good documentation, books and forums that helped me to resolve issues with Hibernate.

7.5 FUTURE ENHANCEMENTS

This project is developed as a beginner application for maintaining students information along with thesis dissertation documents. It can be further developed to provide rich interface. And can be ajaxified for easy navigation between the screens without reloading the pages from the server.

The current application supports ‘Gmail’ as sender id with multiple recipients which can send notification to any Gmail user account, it can be further developed to give yahoo or corporate email accounts using their core APIs for sending the notification to email. So the students are free to access their information from any email account.

This application supports blob size for file upload up to 1 MB; it can be further improved by using Mango DB, which is a pure document-based database system. Spring
Security can be added based on users role for accessing limited information. Additional information can be added for easy accessibility like aisle locations for finding dissertation hard copies in the SDSU library.

7.6 FUTURE RESEARCH

This application can be can be used to store the thesis dissertation information of entire SDSU. So, hardware as well as software maintenance can be a nightmare. The better solution will be hosting it in cloud as SaaS (Software as Service). As cloud provides install and operate application software in the cloud users access the software from cloud client. Since many cloud applications do not require software on the client, instead we can use web browser to interact with the cloud application. Another choice can be exposing this application as a web service, who ever wants to use it can implement in their application. Figures 7.1 [2], 7.2 [2] and 7.3 [2] show the current trends in front-end, databases and back-end technologies.

![Chart showing percentage growth of Spring and EJB frameworks](figure.png)


BIBLIOGRAPHY


