

SOCIAL NETWORKING USING WEB SERVICES

by

VIJAY CHAKRAVARAM

B.Tech, Jawaharlal Nehru Technological University, India, 2013

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Computer Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2016

Approved by:

Major Professor
Dr. Daniel Andresen

Abstract

Web services have expanded to become popular in application development. Web services technology represents an important way of communication between different applications across different platforms. Unlike traditional client/server models, such as a Web application or Web page system, Web services do not provide the user with a GUI. Instead, Web services share business logic, functionality and data through a programmatic interface across a network. Web services are services or functionalities that are exposed to the internet and serves as online or web APIs. These services which are online APIs can be called from your code and use the results in your applications to offer specific functionality to users.

This project consists of two applications the client and the server application. The server application is an online REST API (Web Services developed using REpresentational State Transfer (REST) protocol) which provides all the functionalities as a service across the network that are required to develop a social networking web application.

The client application is similar to any other social networking web application where you can create a profile, delete profile, send messages to your friends, post things, like and comment a post. This applications sends request to the server application using HTTP requests and get the responses. These responses are then consumed in the application to provide the required functionalities to the end user.

Table of Contents

List of Figures	v
List of Tables	vii
Chapter 1 - Introduction.....	1
1.1 Project Description	1
1.2 Motivation.....	2
Chapter 2 - Background.....	3
2.1 Web Services	3
2.2 SOAP Web Services	3
2.3 RESTful Web Services	4
2.3.1 Advantages of RESTful Web Services	4
2.4 Hibernate.....	5
2.4.1 Advantages of using Hibernate.....	5
2.5 JSP	7
2.5.1 Advantages of JSP:	7
Chapter 3 - Setup and Software Requirements	8
Chapter 4 - Client Application Design and Implementation	9
4.1 MVC architecture	9
4.2 Client Application Design	9
4.2.1 DataAccess Class (DAO).....	10
4.2.2 Entity Classes.....	12
4.3 Class Diagram of Client Application.....	13
4.3.1 DataAccess Class	14
4.4 Client Application Use case Diagram.....	17
4.4.1 Uses Cases for User	18
4.4.2 Use Cases for Admin User.....	18
Chapter 5 - Design and Implementation of Server Application	19
5.1 Web API Architecture	19
5.2 Class Diagram.....	21
5.2.1 Entity/Model Classes	22

5.2.2 DataAccess	24
5.3 Hibernate Implementation	27
5.3.1 Hibernate Configuration file	27
5.3.2 Configuration Properties	28
5.3.3 Hibernate Annotations	28
5.3 JAX_RS	30
5.3.1 Jersey- RESTful Web Services	30
5.3.2 JAX_RS Annotations	31
5.3.3 Web API Description	33
Chapter 6 - Securing Web API	35
6.1 SSL Security Implementation	35
6.1.1 SSL certificate and OpenSSL	35
6.1.2 Configuring server.xml for handling SSL security	36
6.1.3 Configuring web.xml	36
Chapter 7 - Screen Shots	38
Chapter 8 - Testing	49
8.1 Testing Web API	49
8.1.1 Unit Testing	50
8.1.2 Testing Using Post REST Client	51
8.1.3 Limitations	52
8.1.4 Performance testing using jmeter-Test1	52
8.1.5 Performance testing using jmeter-Test2	53
8.2 Testing Client Application	55
8.2.1 Unit Testing	55
8.3 Performance Testing Using Mozilla Developers Tool	56
Chapter 9 - Learning and Experience	62
9.1 Learning	62
9.2 Project Development Experience	64
Chapter 10 - Future Work	65
Chapter 11 - Conclusion	66
Chapter 12 - References	67

List of Figures

Figure 1: Server Client Interaction in RESTful Web services [2]	5
Figure 2: Hibernate Architecture [3].....	6
Figure 3: Class Diagram of the Client Application	13
Figure 4: Use case Diagram for Client Application.....	17
Figure 5: Server Application Architecture.....	20
Figure 6: Server Application Class Diagram	21
Figure 7: Entity Class.....	22
Figure 8: Generated Table for the Entity Class	23
Figure 9: Generated XML for the Entity Class.....	23
Figure 10: Hibernate Configuration file	27
Figure 11: Hibernate Annotated Entity Class	29
Figure 12: Generated Table for Annotated Class	30
Figure 13: Annotated Web Service.....	31
Figure 14: Response Of Web Service.....	32
Figure 15: Server.xml configuration file	36
Figure 16: Web.xml of server Application	36
Figure 17: Login Page.....	38
Figure 18: Sign up page	38
Figure 19: Login Fail	39
Figure 20: Admin Home Page	39
Figure 21: User Home Page.....	40
Figure 22: Adding a Post	40
Figure 23: Post Success	41
Figure 24: View all posts	41
Figure 25; See Likes	42
Figure 26: View Profile	42
Figure 27: View comments.....	43
Figure 28: Send Comment	43
Figure 29: Send Comment Success.....	44

Figure 30: View Messages.....	44
Figure 31: Send Message	45
Figure 32: Type Your Message.....	45
Figure 33: Message Sent Success	46
Figure 34: Create Profile.....	46
Figure 35: Profile Creation Success.....	47
Figure 36: Delete Profile.....	47
Figure 37: Deleting Profile Success.....	48
Figure 38: Logout	48
Figure 39: Postman Rest Client Results.....	51
Figure 40: Throughput Graph 1	52
Figure 41: JMeter Test Configuration.....	53
Figure 42: Graph Showing Response Time	53
Figure 43: Graph showing the throughput	54
Figure 44: Time taken to load login Page	57
Figure 45: Time taken to load home page	57
Figure 46: Time taken to display all posts.	58
Figure 47: Time taken to display all messages	58
Figure 48: Time taken to add a post.....	59
Figure 49: Time taken to Send Message.....	59
Figure 50: Time taken to delete a profile.....	60
Figure 51: Time taken to create a profile.....	60
Figure 52: Time taken to logout.....	61

List of Tables

Table 1: Table Showing Test Case	50
Table 2: Test Cases Set 1 for Client.....	55
Table 3: Test Cases Set 2 for Client.....	56

Chapter 1 - Introduction

1.1 Project Description

This project consists of two applications the client and the server application. The server application is a REST API which provides all the functionalities that are required to develop a social networking web application. This API uses Hibernate framework to communicate and perform CRUD operations on the underlying database. This API handles all types of HTTP method calls and accepts data in the form of XML and give responses in XML format. SSL security with basic authentication are implemented to secure the Web API so that only an authentic users can use this API.

The client application is similar to any other social networking web application where you can create a profile and send messages to your friends, post things, like and comment a post. The functionality of deleting profiles can only be done by the admin user.

The client application sends requests to the services provided by this REST API using GET, PUT, DELETE, POST methods of HTTP protocol and gets the responses back in the form of XML. The XML response is parsed and then consumed in the application to produce required functionalities to the end user.

1.2 Motivation

Offices, workplaces and organizations uses different applications to provide ways of communication between their employees, groups and teams. My idea is to provide them with a customized internal social networking application similar to Facebook where the employees can not only send messages between them but also post things on to the wall, like them and comment on them. The organizations can just build their own customized GUI and get all the required functionalities from the Web API. They need not worry about the functionality or maintain any kind of database which reduces the overhead on organizations. This application can be easily embedded in to their company's website.

Chapter 2 - Background

2.1 Web Services

A web service is any piece of software that makes itself available over the internet. Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents. Web services provide a standard means of interoperability between software applications running on a variety of platforms and frameworks. Web services are characterized by their great interoperability and extensibility as they use XML for exchange of data. These services which serves as online APIs can be called from your code and use the results in your applications. So, technically you send a request to a web service and consume the response in your application.

There are two different types of Web Services depending upon the design and their architecture. They are SOAP based web services and RESTful web services.

2.2 SOAP Web Services

SOAP stands for Simple Object Access Protocol. These are the web services which use XML messages for exchange of data between applications across the network. These XML messages should follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats. SOAP is a W3C recommendation. In Java EE 6, JAX-WS provides the functionality for implementing web services using SOAP stands for Simple Object Access Protocol.

2.3 RESTful Web Services

REST stands for REpresentational State Transfer Protocol ^[1]. Web Services which are developed following REST protocol are called as RESTful web services. REST web services which works on http protocol. In RESTful web services the client communicates with the API using GET, POST, PUT, DELETE methods of http protocol and gets the response in the form of XML/JASON etc. RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web based applications. JAX-RS which stands for Java API for XML Restful Web Services is an API for implementing RESTful Web Services. REST allows applications to transfer data between then in different formats like XML, JASON, TEXT etc. REST is language and platform independent.

There are different implementations available for JAX-RS such as Jersey, RESTEasy, Restlet, Apache CXF etc. I have used Jersey implementation of JAX-RS in my project to develop the client application.

2.3.1 Advantages of RESTful Web Services

- RESTful web services are lightweight and have better performance.
- RESTful web services are built to work best on Web.
- Responses from RESTful web services can be easily parsed and consumed in application.
- In REST architecture style data, functionality are considered resources and are accessed using Uniform Resource Identification (URL).

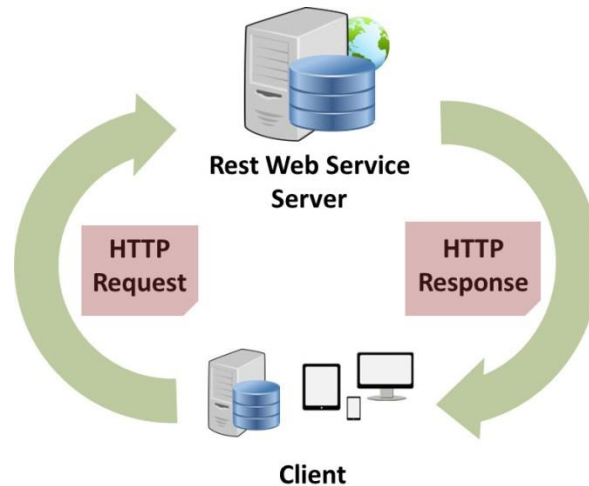


Figure 1: Server Client Interaction in RESTful Web services ^[2]

2.4 Hibernate

Hibernate is a high-performance Object/Relational Mapping tool. ORM is a programming technique for converting data between relational databases and object oriented programming languages such as Java. Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

2.4.1 Advantages of using Hibernate.

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.

- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Manipulates Complex associations of objects of your database.
- Hibernate does not require an application server to operate.
- Minimize database access with smart fetching strategies.
- Provides simple querying of data.

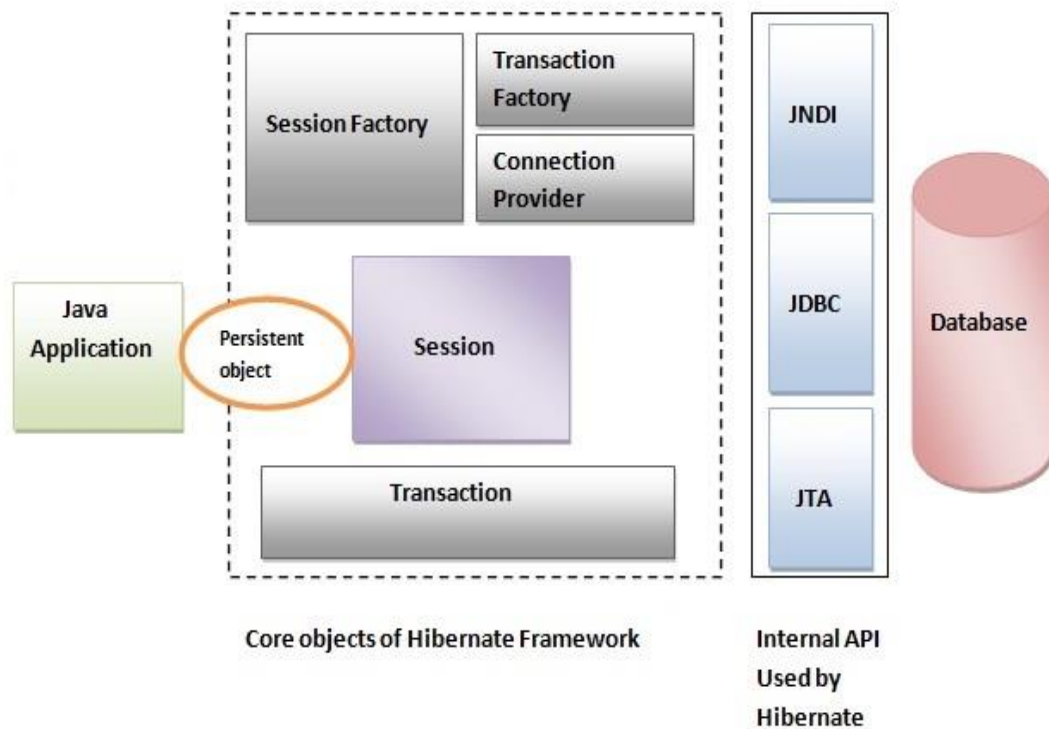


Figure 2: Hibernate Architecture ^[3]

2.5 JSP

Java Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with `<%` and end with `%>`.

A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

2.5.1 Advantages of JSP:

- JSP pages easily combine static templates, including HTML or XML fragments, with code that generates dynamic content.
- JSP pages are compiled dynamically into servlets when requested, so page authors can easily make updates to presentation code.
- Java Server Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.

Chapter 3 - Setup and Software Requirements

The following are the tools/software I have used for my project development

- *Integrated Development Environment (IDE)*: I have used Eclipse Luna with J2EE prospective
- *Database*: Oracle Database Expression Edition 11g
- *Hibernate Package*: hibernate-relese-5.0.2.Final
- *Jersey Package*: jersey-achive-1.19 (for developing RESTful web services and REST Client)
- *JAX-B*: jaxb-ri-2.2.7: JAX-B Stands for Java Architecture for XML Binding which is used to convert java Objects to XML documents and vice versa which is used at client side.
- *Operating System*: Windows 8.1
- *Java Development Kit*: JDK 1.8.0_40
- *Server*: apache-tomcat-8.0.27
- *REST Client*: Postman chrome plugin.

Chapter 4 - Client Application Design and Implementation

4.1 MVC architecture

Model View Controller or MVC ^[4] as it is popularly called, is a software design pattern for developing web applications. It logically separates the presentation/view part, controller and the data access layers.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.

- **Model** - The lowest level of the pattern which is responsible for maintaining data.
- **View** - This is responsible for displaying all or a portion of the data to the user.
- **Controller** - Software Code that controls the interactions between the Model and View.

4.2 Client Application Design

In my Client application, the entire logic for view part is programmed using JSP. The same JSP handles the events from the user and sends requests to DataAccess class to get the required functionality. After receiving the response from the DAO, the same JSP is also responsible for passing the control to another JSP. So, in my client application the JSPs play the role of the View and the Controller.

Inside the JSP the view part is handled by embedded html code. JSP capable of building the html code dynamically. It also uses CSS code for styling and JavaScript for client side form data validations.

4.2.1 DataAccess Class (DAO)

The DAO class handles all the requests coming from the JSPs to get data from database or manipulate data in the database. But, the client application does not have a database of its own. So, it dynamically constructs an URL and sends the requests to the RESTful web services to get the required functionality. It also performs the functionality of marshalling and Un-marshalling.

The DataAcces class makes use of the POJO (Plain Old Java Object) classes or the model classes for the purpose of marshalling and un-marshalling.

Marshalling and Un-marshalling

JAX-B (Java Architecture for XML Binding) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to un-marshal XML back into Java objects. JAXB mostly is used while implementing web services or any other such client interface for an application where data needs to be transferred in XML format.

The following diagram shows the control flow and architecture of the client application

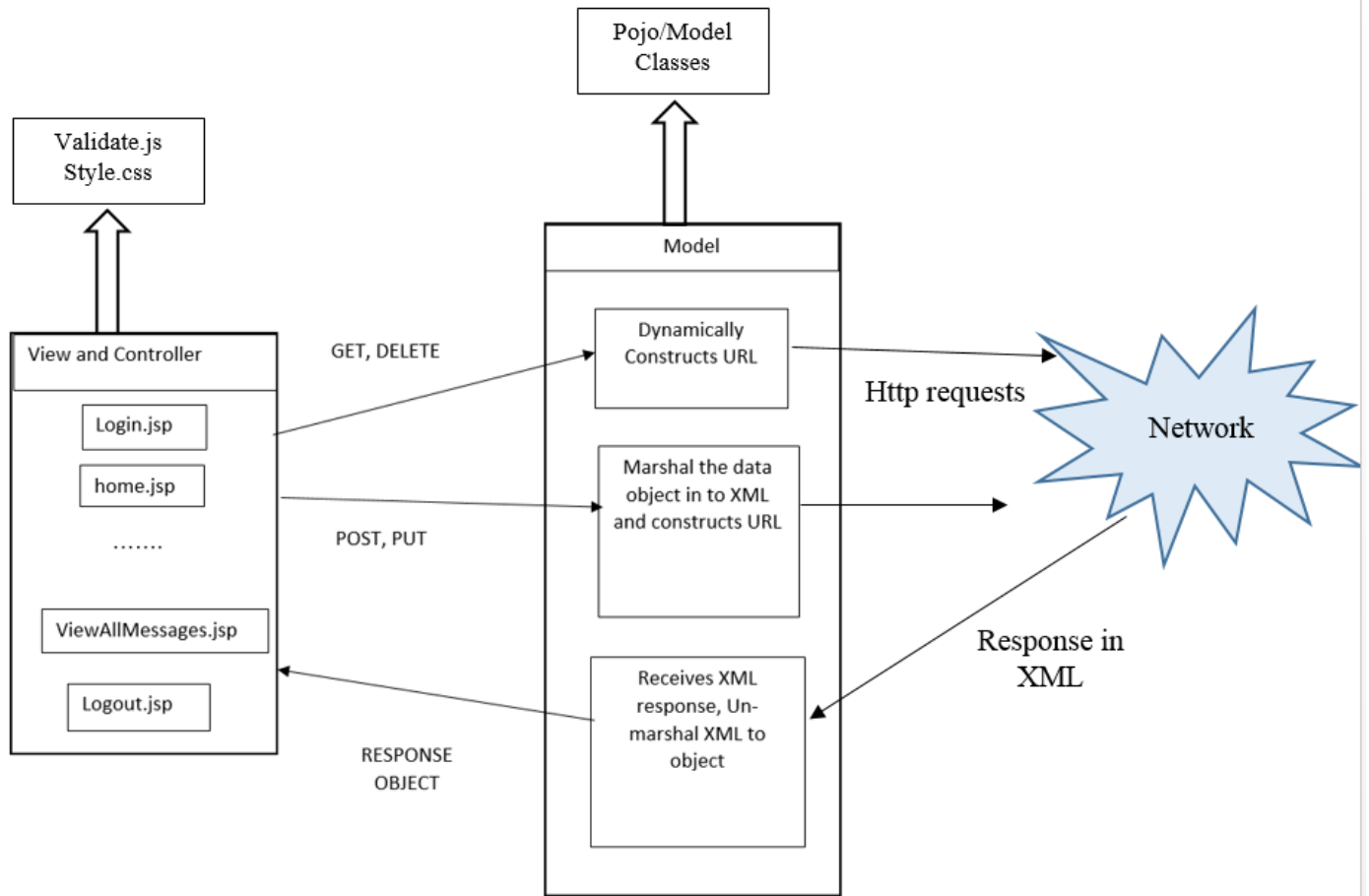
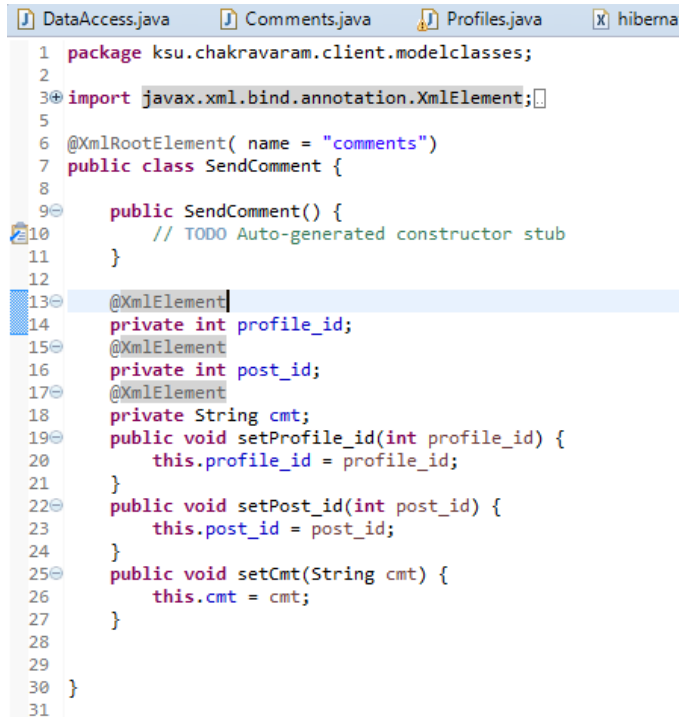


Figure 3: Client Architecture

4.2.2 Entity Classes

These classes does not have any functionality in them. They just only have member variables and getters and setters for those variables. The following is the code snippet of an entity class.



```
1 package ksu.chakravaram.client.modelclasses;
2
3 import javax.xml.bind.annotation.XmlElement;
4
5
6 @XmlRootElement( name = "comments")
7 public class SendComment {
8
9     public SendComment() {
10         // TODO Auto-generated constructor stub
11     }
12
13     @XmlElement
14     private int profile_id;
15     @XmlElement
16     private int post_id;
17     @XmlElement
18     private String cmt;
19     public void setProfile_id(int profile_id) {
20         this.profile_id = profile_id;
21     }
22     public void setPost_id(int post_id) {
23         this.post_id = post_id;
24     }
25     public void setCmt(String cmt) {
26         this.cmt = cmt;
27     }
28
29 }
30
31
```

Figure 4: Entity Class

The above entity class with XML annotations can be converted in to XML by marshalling. DataAccess class converts the above class in to XML which looks like below.

```
<comments>
  <cmt>hello</cmt>
  <post_id>301</post_id>
  <profile_id>101</profile_id>
</comments>
```

Figure 5: Entity Class XML

So, XML and Entity classes are interconvertible.

4.3 Class Diagram of Client Application

The class diagram shows all the different classes present in the application and the relationships between them. In my client application I have 12 Entity classes and one Data Access class. The DataAccess class uses the entity classes for marshalling and un-marshalling.

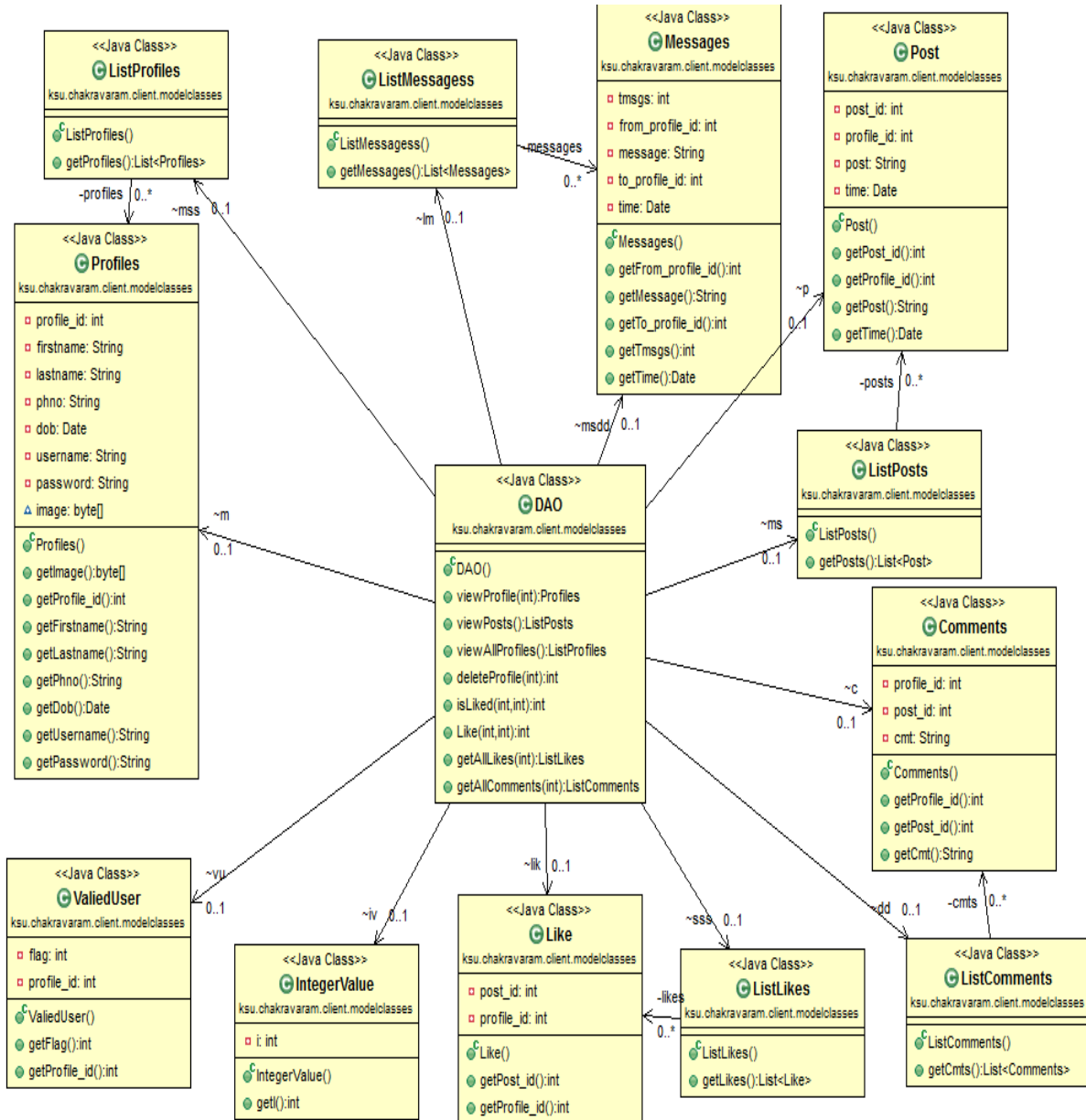


Figure 6: Class Diagram of the Client Application

4.3.1 DataAccess Class

This class accepts request from the JSPs and then construct a dynamic URL and call RESTful web service. Upon receiving the response in XML format, convert them to objects and gives them back to JSPs. The following are the various functions present in the DAO class.

➤ **public Profiles viewProfile(int id)**

This function accepts profile_id as input then calls RESTful web service by constructing the URL. Once the GET request is made on the URL it receives XML response and convert it to Profiles object which is returned to the JSPs.

URL: "http://localhost:8081/WebServices_Final1/backend/profiles/"+id

➤ **public ListPosts viewPosts()**

This function calls RESTful web service by constructing the following URL which gives list of all the posts in form of XML. This XML response is convert it to ListPosts object containing list of Post objects which is returned back to the JSPs.

URL: "http://localhost:8081/WebServices_Final1/backend/profiles/allposts"

➤ **public ListProfiles viewAllProfiles()**

This function calls RESTful web service by constructing the following URL which gives list of all the Profiles in form of XML. This XML response is convert it to ListProfiles object containing list of Profiles objects which is returned back to the JSPs.

URL: http://localhost:8081/WebServices_Final1/backend/profiles/allprofiles

➤ **public int deleteProfile (int id)**

This function accepts profile_id as input then calls RESTful web service by constructing the URL. Once the DELETE request is made on the URL it receives XML response containing the object of IntegerValue. The IntegerValue object is convert it to integer which is returned to the JSPs.

Value 1 implies successful deletion of profile and 0 implies failure in deletion.

URL: "http://localhost:8081/WebServices_Final1/backend/profiles/delete/"+id

➤ **public int isLiked (int pfid, int pid)**

This function accepts profile_id, post_id as inputs then calls RESTful web service by constructing the URL. Once the GET request is made on the URL it receives XML response containing the object of IntegerValue. The IntegerValue object is convert it to integer which is returned to the JSPs. Value 0 implies not liked and 1 implies liked.

URL: http://localhost:8081/WebServices_Final1/backend/profiles/isliked/"+pfid+"/"+pid

➤ **public int Like (int pfid, int pid)**

This function accepts profile_id, post_id as inputs then calls RESTful web service by constructing the URL. Once the GET request is made on the URL it receives XML response containing the object of IntegerValue. The IntegerValue object is convert it to integer which is returned to the JSPs. Value 1 implies success and 0 implies failure in liking the post.

URL: "http://localhost:8081/WebServices_Final1/backend/profiles/like/"+pfid+"/"+pid

➤ **public ListLikes getAllLikes (int post_id)**

This function accepts post_id as inputs and then calls RESTful web service by constructing the following URL which gives list of all the likes for a post in form of XML. This XML response is convert it to ListLike object containing list of Like objects which is returned back to the JSPs.

URL: "http://localhost:8081/WebServices_Final1/backend/profiles/getlikes/"+post_id

➤ **public ListComments getAllComments (int post_id)**

This function accepts post_id as inputs and then calls RESTful web service by constructing the following URL which gives list of all the comments for a post in form of XML. This XML response is convert it to ListComments object containing list of Comments objects which is returned back to the JSPs.

URL:

http://localhost:8081/WebServices_Final1/backend/profiles/getcomments/"+post_id

4.4 Client Application Use case Diagram

A use case diagram is a graphic depiction of the interactions among the elements of a system. In my client application I have two different type of users the Admin user and the normal user. The following diagram gives the details of how they interact with the system.



Figure 7: Use case Diagram for Client Application

4.4.1 Uses Cases for User

- *Sign Up*: The user can create a new account of his own by giving the details.
- *Login*: With the login credentials given at the time of sign up, the user can login in to the system.
- *Add Post*: The user can add a post will be visible to all the other users.
- *View All Posts*: The user can see all the posts posted from different profiles.
- *See All Comments For Post*: The user can see all the comments for a particular post.
- *Comment on Post*: The user can comment on any post.
- *See All Likes For Post*: The user can see who all liked a particular post.
- *Like Post*: The user can like a post.
- *View All Messages*: The user can see all the messages he received from different users.
Send Message: the user can send messages to a particular profile.
- *Logout*: The user can logout from the system.

4.4.2 Use Cases for Admin User

The admin user can perform all the activities that are done by the normal user. Apart from those the admin user can *create a new profile* for a user and can also *delete a profile* from the system.

Chapter 5 - Design and Implementation of Server Application

5.1 Web API Architecture

The server application is a Web API which contain three different type of classes. The Resource Class, DataAccess Class and the Entity/Model Classes.

Resources Class is the starting point of the application in which the web services are implemented. Every HTTP request coming from the browser with the matching URL comes to this class. Depending up on the URL pattern, type of request (GET, DELETE, and PUT) this program sends request to the DataAccess Class. This program also passes the parameters and XML data that come along with the URL and sends request to DataAccess Class.

The Resources class receives the response in the form of object/ integer value and then converts it in to xml format using the model/**POJO classes**. This XML response is sent back to the browser.

The web.xml is configured to send all the incoming HTTP requests to this program which matches the defined pattern.

The **DataAccess Class** speaks to the data base using Hibernate. This program is capable of performing CURD (create, update, retrieve and delete) operations on the data base using hibernate frame work.

The following diagram shows the control flow and architecture of the server application

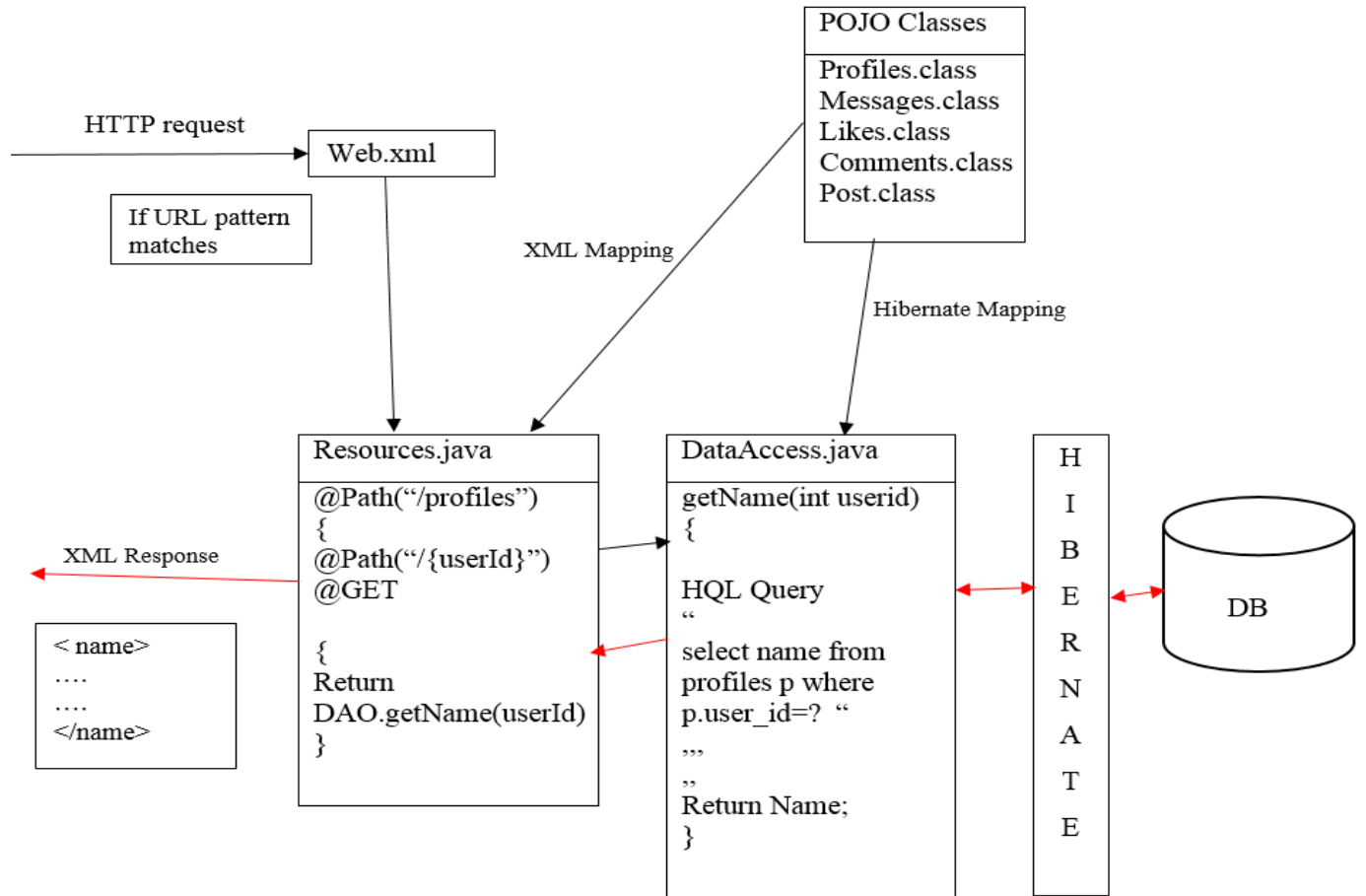


Figure 8: Server Application Architecture

5.2 Class Diagram

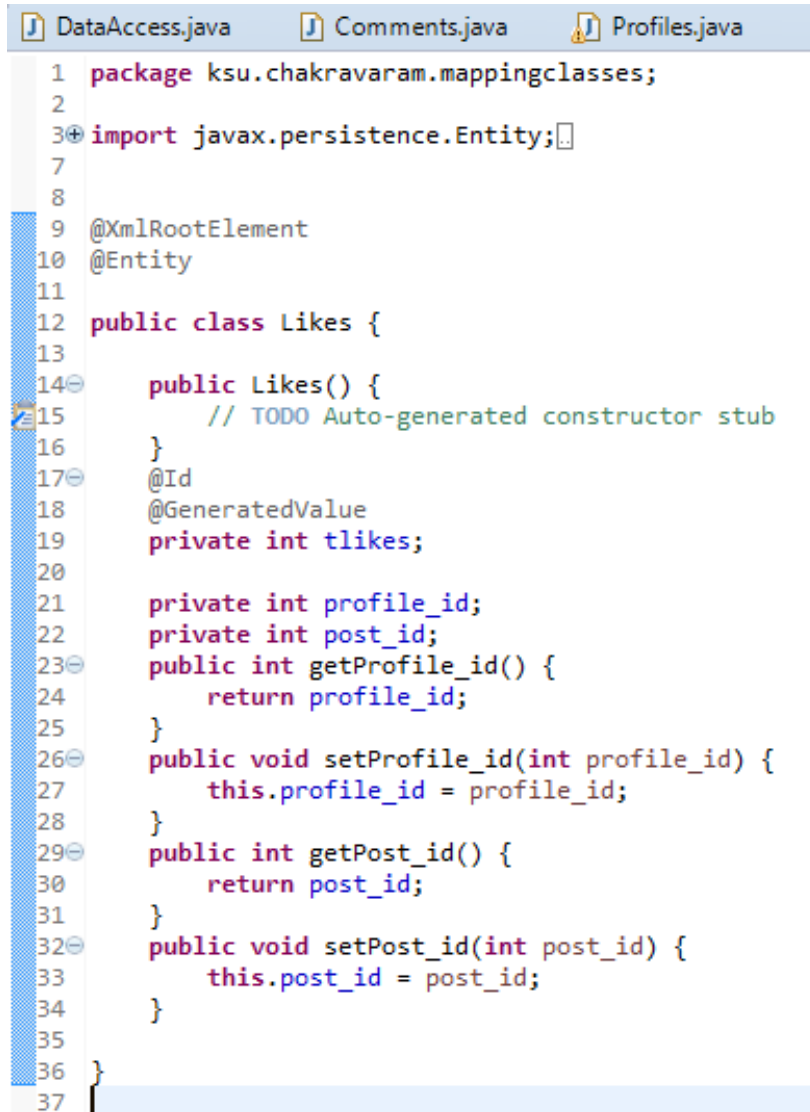
The class diagram shows all the different classes present in the application and the relationships between them. In my server application I have 5 entity/POJO classes, 1 DataAccess class and one Resources class. The incoming requests are handled by the Resources class and calls appropriate methods of DataAccess class. DataAccess class perform curd operations using Hibernate and the model classes and then returns objects to the Resources class. The resources class converts these objects in to XML and sends response back to the browser.



Figure 9: Server Application Class Diagram

5.2.1 Entity/Model Classes

These classes do not have any functionality in them. They just only have member variables and getters and setters for those variables. The following is the code snippet of an entity class.



```
1 package ksu.chakravaram.mappingclasses;
2
3 import javax.persistence.Entity;
4
5
6
7
8
9 @XmlElement
10 @Entity
11
12 public class Likes {
13
14     public Likes() {
15         // TODO Auto-generated constructor stub
16     }
17     @Id
18     @GeneratedValue
19     private int tlikes;
20
21     private int profile_id;
22     private int post_id;
23     public int getProfile_id() {
24         return profile_id;
25     }
26     public void setProfile_id(int profile_id) {
27         this.profile_id = profile_id;
28     }
29     public int getPost_id() {
30         return post_id;
31     }
32     public void setPost_id(int post_id) {
33         this.post_id = post_id;
34     }
35
36 }
37
```

Figure 10: Entity Class

All the entity classes are used by DataAccess class to map them to tables and by Resources class to convert objects in to XML files. The following are the code snippets for the generated XML and the generated table for the entity class.

```
SQL> conn
Enter user-name: system
Enter password:
Connected.
SQL> describe likes;
Name                               Null?    Type
-----
TLIKES                             NOT NULL NUMBER<10>
POST_ID                             NOT NULL NUMBER<10>
PROFILE_ID                           NOT NULL NUMBER<10>
SQL>
```

Figure 11: Generated Table for the Entity Class

```
<likes>
  <post_id>301</post_id>
  <profile_id>101</profile_id>
</likes>
...
```

Figure 12: Generated XML for the Entity Class

5.2.2 DataAccess

The DataAccess Class speaks to the data base using Hibernate. This program is capable of performing CRUD (create, update, retrieve and delete) operations on the data base using hibernate frame work. The following are the various methods present in it.

➤ **public int getProfileId (String username, String password)**

Accepts username, password and returns profile_Id if it's a valid username and password.

HQL Query: "select e.profile_id from Profiles e where e.username=? and e.password=?"

➤ **public Profiles getProfile (int pid)**

Accepts profile_id and returns all the profile details of the given profile_id

HQL Query: select e from Profiles e where e.profile_id=?"

➤ **public List<Messages> getMessages (int pid)**

Accepts profile_id and returns all the messages available for that profile_id.

HQL Query: "select e from Messages e where e.to_profile_id=?"

➤ **public int deleteProfile (int pid)**

Accepts profile_id and deletes that particular profile from the database.

HQL Query: delete from Profiles e where e.profile_id=?"

➤ **public int createProfile (Profiles p)**

Accepts object of Profile and saves it in the form of a record in Profiles table in the database. The profile_id is automatically incremented by getting the max value of the profile_id from the profiles table using the following query.

HQL Query: "select max (e.profile_id) from Profiles e"

➤ **public int createPost (Posts p)**

Accepts object of Post and saves it in the form of a record in Posts table in the database. The post_id is automatically incremented by getting the max value of the post_id from the posts table using the following query.

HQL Query: "select max (e.post_id) from Posts e"

➤ **public List<Posts> getAllPosts ()**

Gets all the posts in the form on list of Post objects by executing the following query.

HQL Query: "select e from Posts e"

➤ **public List<Profiles> getAllProfiles()**

Gets all the profiles in the form on list of Profiles objects by executing the following query.

HQL Query: "select e from Profiles e"

➤ **public int sendMsg (Messages p)**

Accepts object of Messages and saves it in the form of a record in Messages table in the database.

➤ **public int isLiked (int proid, int pid)**

Accepts profile_id, post_id as parameters and returns 1 if there exist a record in the Posts table else returns 0 by executing the following query.

HQL Query: "select e from Likes e where e.profile_id=? and e.post_id=?"

➤ **public int like (int proid, int pid)**

Accepts profile_id, post_id as parameters and constructs an object of Class Like and saves it as a record in the database.

➤ **public int hasLikes (int pid)**

Accepts post_id as parameter and returns 1 if a record exists in Likes table, else returns 0

by executing the following query.

HQL Query: "select distinct e from Likes e where e.post_id=?"

➤ **public List<Likes> getLikes (int pid)**

Accepts Post_id and returns list of all the likes for that post by executing the following query.

HQL Query: "select distinct e from Likes e where e.post_id=?"

➤ **public int newCmt (Comments p)**

- Accepts object of comments and saves it as a record in Comments table in the database.

➤ **public List<Comments> getComments (int pid)**

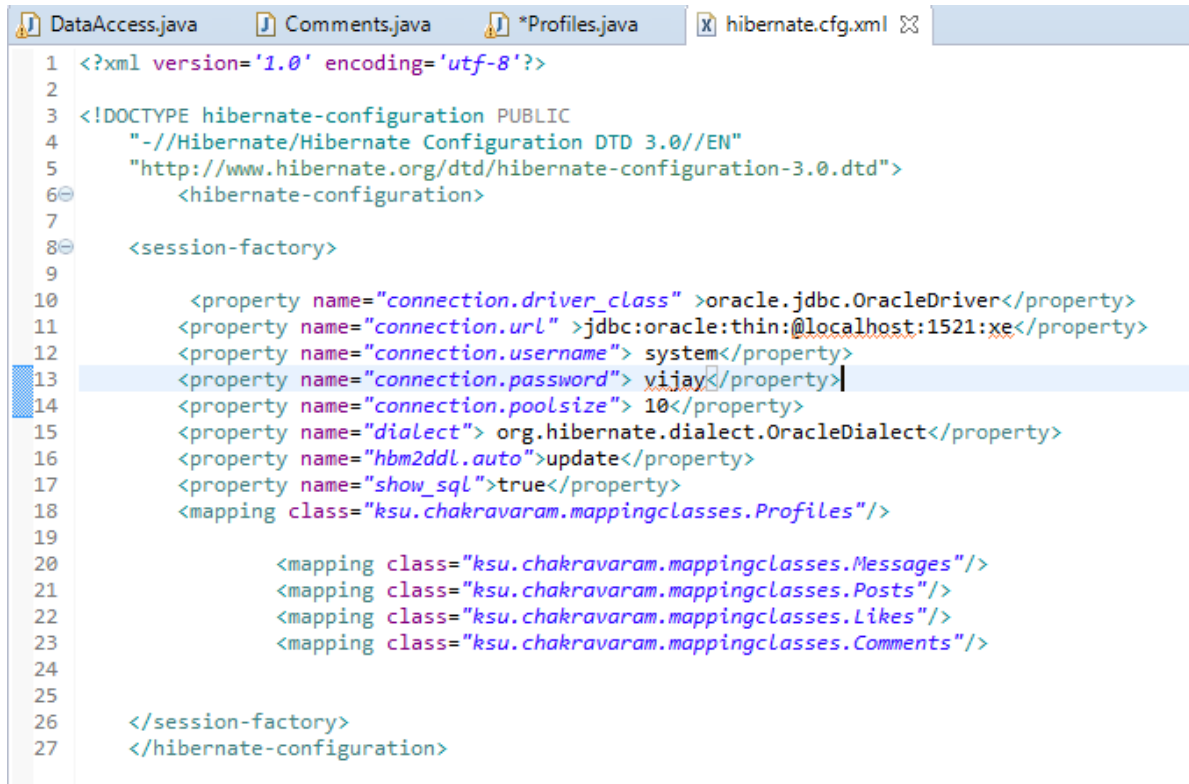
Accepts post_id as parameter and gets list of all the comments for that post by executing the following query.

HQL Query: "select e from Comments e where e.post_id=?"

5.3 Hibernate Implementation

5.3.1 Hibernate Configuration file

For every database schema we need to define “hibernate.cfg.xml” which is hibernate configuration file. In this XML file we define the details of the underlying database which will be used by hibernate to speak to the database. We, also provide the locations of the POJO classes which will be used by hibernate to convert them in to corresponding tables in the database. The “hibernate.cfg.xml” looks like below



```
1 <?xml version='1.0' encoding='utf-8'?>
2
3 <!DOCTYPE hibernate-configuration PUBLIC
4     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
5     "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
6     <hibernate-configuration>
7
8     <session-factory>
9
10        <property name="connection.driver_class" >oracle.jdbc.OracleDriver</property>
11        <property name="connection.url" >jdbc:oracle:thin:@localhost:1521:xe</property>
12        <property name="connection.username" > system</property>
13        <property name="connection.password" > vijay</property>
14        <property name="connection.poolsize" > 10</property>
15        <property name="dialect" > org.hibernate.dialect.OracleDialect</property>
16        <property name="hbm2ddl.auto" >update</property>
17        <property name="show_sql" >true</property>
18        <mapping class="ksu.chakravaram.mappingClasses.Profiles"/>
19
20        <mapping class="ksu.chakravaram.mappingClasses.Messages"/>
21        <mapping class="ksu.chakravaram.mappingClasses.Posts"/>
22        <mapping class="ksu.chakravaram.mappingClasses.Likes"/>
23        <mapping class="ksu.chakravaram.mappingClasses.Comments"/>
24
25
26    </session-factory>
27 </hibernate-configuration>
```

Figure 13: Hibernate Configuration file

5.3.2 Configuration Properties

Using the Property tag inside session-factory tag we give the following details.

- *Driver Class*: “oracle.jdbc.OracleDriver” as I am using oracle database (which changes from database to database).
- *Connection URL*: “oracle.jdbc.OracleDriverjdbc:oracle:thin:@localhost:1521:xe” as I am using thin driver running in hocalhost at port number 1521.
- *Username*: The username of the database
- *Password*: The password of the database for the corresponding user.
- *Dialect*: “org.hibernate.dialect.OracleDialect” as using Oracle database.
- *Hdm2ddl.auto*: Set to “update” which automatically creates tables if they are not present in the database.
- *Show_sql*: Set to “true” which displays the queries which gets executed in the console.
- *Mapping class*: Set these to all the different POJO classes which have to be mapped to the tables.

5.3.3 Hibernate Annotations

The following are the hibernate annotations I have used in my application. Hibernate Annotations are used to map a particular class to a database table, they make POJO class a persistent entity. Consider the following example.

- *@Entity*: This tag declares the class as an entity (i.e. a persistent POJO class).
- *@Id* : This tag declares the identifier property (primary key) of this entity
- *@Column (Unique=true)*: This id imposes unique constraint on the column.
- *@GeneratedValue*: This tag automatically generates a unique value to the column.

```

package ksu.chakravaram.mappingclasses;

import java.util.Date;
@XmlRootElement
@Entity
public class Profiles {

    public void Profiles()
    {

    }
    @Id

    private int profile_id;
    private String firstname;
    private String lastname;
    private String phno;
    private Date dob;
    @Column(unique=true)
    private String username;
    private String password;
    private byte[] image;
}

```

Figure 14: Hibernate Annotated Entity Class

Here in the above piece of code, the class Profiles is mapped to the Profiles table with columns profile_id, firstname, lastname, phno, dob, username, password and image. The primary key of the table will be profile_id and username will have unique constraint.

When executed the a table named Profiles is created in the database and hibernate will take care of the data types depending on the datatypes of the variables present in the POJO class. The table looks like below.

```

SQL> conn
Enter user-name: system
Enter password:
Connected.
SQL> describe profiles;
Name                                     Null?   Type
-----
PROFILE_ID                             NOT NULL NUMBER(10)
DOB                                     DATE
FIRSTNAME                              VARCHAR2(255)
LASTNAME                                VARCHAR2(255)
PASSWORD                                VARCHAR2(255)
PHNO                                     NOT NULL NUMBER(10)
USERNAME                                VARCHAR2(255)
IMAGE                                   BLOB

```

Figure 15: Generated Table for Annotated Class

5.3 JAX-RS

Java API for RESTful Web Services (**JAX-RS**) ^[5] is a Java programming language API that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern.

5.3.1 Jersey- RESTful Web Services

In order to simplify development of RESTful Web services and their clients in Java, a standard and portable JAX-RS API has been designed. Jersey RESTful Web Services framework is open source, production quality, and framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation.

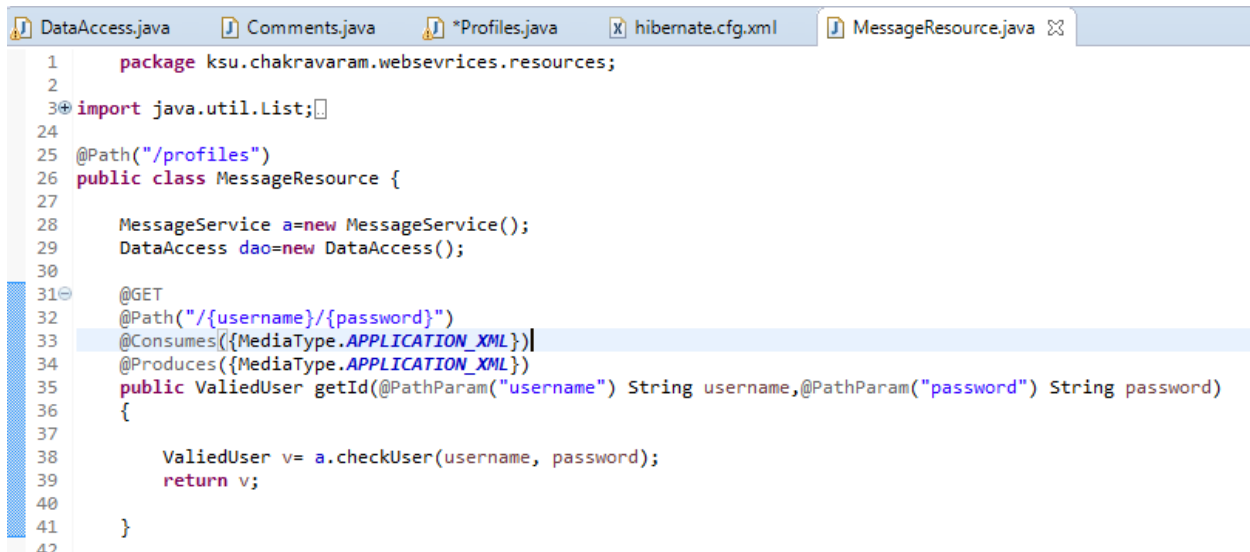
Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides its own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension SPIs so that developers may extend Jersey to best suit their needs.

5.3.2 JAX-RS Annotations

These annotations ^[6] helps us to declare our functionalities as web services. The following are the annotations I have used in my application.

- @Path : This tag specifies the relative URL path
- @PathParam: This tag helps to access the parameters inside the program sent by the user in the request URL.
- @Consumes: This tag imposes the accepted request format.
- @Produces: This tag species the type of response format.
- @GET, @POST, @DELETE (HTTP methods): These are *resource method designator* annotations defined by JAX-RS and which correspond to the similarly named HTTP methods.

Consider the following snippet of the code.



```
1 package ksu.chakravaram.websevrices.resources;
2
3 import java.util.List;
4
25 @Path("/profiles")
26 public class MessageResource {
27
28     MessageService a=new MessageService();
29     DataAccess dao=new DataAccess();
30
31 @GET
32 @Path("/{username}/{password}")
33 @Consumes({MediaType.APPLICATION_XML})
34 @Produces({MediaType.APPLICATION_XML})
35 public ValiedUser getId(@PathParam("username") String username,@PathParam("password") String password)
36 {
37
38     ValiedUser v= a.checkUser(username, password);
39     return v;
40
41 }
42
```

Figure 16: Annotated Web Service

In the above program the function getId () accepts username, password and returns ValiedUser object. In order to access this functionality, the URL must be as follows.

http://localhost:8081/WebServices_Final1/backend/profiles/vijay/vijay and this should be a GET request. And from this URL we are sending the parameters username and password as “vijay”.

The response will be as follows.

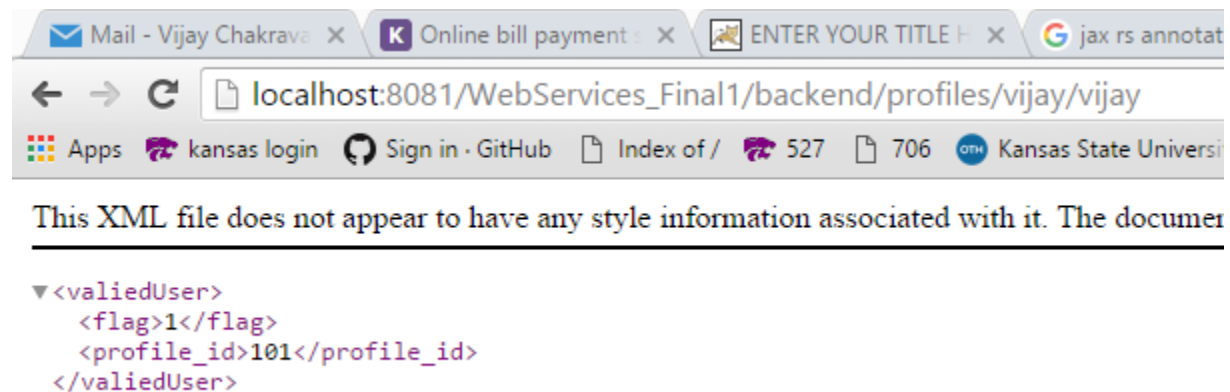


Figure 17: Response of Web Service

Returns object of ValiedUser as response. Flag value="1" denotes valid user and with profile_id as 101.

5.3.3 Web API Description

The following are the different web services present in the API.

Method	URL	Return Object	Description
@GET	/username/{password}	ValidUserObject	Returns profile_id if the username and password are valid
@GET	/profile_id	Profile Object	Gives All the details of the given profile_id
@GET	/messages/profile_id	List<Messages>	Gives all the messages for the given profile_id
@GET	/allposts	List<Posts>	Gives all the posts
@GET	/allprofiles	List<Profiles>	Gives details of all the profiles
@GET	/getcomments/post_id	List<Comments>	Gets all the comments for a given post_id
@GET	/isliked/profile_id/post_id	IntegerResponse	Tells whether the given post is liked by the given profile
@GET	/like/profile_id/post_id	IntegerResponse	Stores the record in likes table
@GET	/haslikes/post_id	IntegerResponse	Tells whether the given post_id has any likes or not.
@GET	/getlikes/post_id	List<Likes>	Gives all the like for a post_id
@Delete	/delete/profile_id	IntegerResponse	Deletes the given profile
		Accepting Object	
@POST	/create	Profile	Creates a new profile
@POST	/newpost	Post	Adds a new post
@POST	/sendmsg	Message	Sends message to a profile
@POST	/sendcmt	Comment	Adds a comment to a post

The following two services are secured under SSL and basic user authentication

Method	Secured URL	Security Constraints	Description
@GET	/message/{profile_id}	Need to Accept SSL certificate, and need username and password	Gets all the messages for a given profile_id
@DELETE	/del/{profile_id}	Need to Accept SSL certificate, and need username and password	Deletes the given profile

Chapter 6 - Securing Web API

The URL using which the user access the web API are to be secured as we don't want the API to be used only by the authenticated users. In my project I have used SSL security with basic authentication for securing my web services.

6.1 SSL Security Implementation

6.1.1 SSL certificate and OpenSSL

SSL (Secure Sockets Layer) Implementation is the standard **security** technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. In order to implement SSL security we need an SSL certificate and get it signed by a third party. So, in my application I have used I have used Open SSL ^[7] to create a self-signed SSL certificate.

OpenSSL is an open source project that provides a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library. The details of generating a certificate can be known from the website.

Once the certificate is generated we get a public and a private key which have to be configured in servers.xml of the underlying server.

6.1.2 Configuring server.xml for handling SSL security

By un-commenting the shown piece of code below and configuring ^[8] its values must make the server ready to handle SSL security.

```
-->
<!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443
This connector uses the NIO implementation that requires the JSSE
style configuration. When using the APR/native implementation, the
OpenSSL style configuration is required as described in the APR/native
documentation -->

<Connector port="8443" protocol="HTTP/1.1"
    maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
    SSLCertificateFile="C:/OpenSSL-Win64/bin/server.crt"
    SSLCertificateKeyFile="C:/OpenSSL-Win64/bin/server.key"
    clientAuth="false" sslProtocol="TLS" />
```

Figure 18: Server.xml configuration file

6.1.3 Configuring web.xml

The web.xml of the server application has to be configured to define the secured URL patterns and levels of security. This can be done by using the security constraint tag in web.xml. The web.xml of my application looks like below.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>hello</web-resource-name>
    <url-pattern>/backend/profiles/del/*</url-pattern>
    <http-method>DELETE</http-method>
  </web-resource-collection>

  <web-resource-collection>
    <web-resource-name>hello</web-resource-name>
    <url-pattern>/backend/profiles/message/*</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>

  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
  <user-data-constraint><transport-guarantee>CONFIDENTIAL</transport-guarantee></user-data-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

Figure 19: Web.xml of server Application

- *<web-resource-collection>*: This tag define the URL patterns and the methods which are to be secured.
- *<auth-constraint>*: This tag defines the role of the user who can access the secured URLs.
- *<login-config>*: This tag defines the type of authentication. In my application I have used basic authentication which asks for a username and password when the URL is accessed.

So, In order to access the secured URL one must accept the certificate, his role has to be “admin” and must have a username and password as mentioned in the web.xml of the server.

Chapter 7 - Screen Shots

1. *Login Page*: The user can login to the system from this page. Username and password fields are validated.

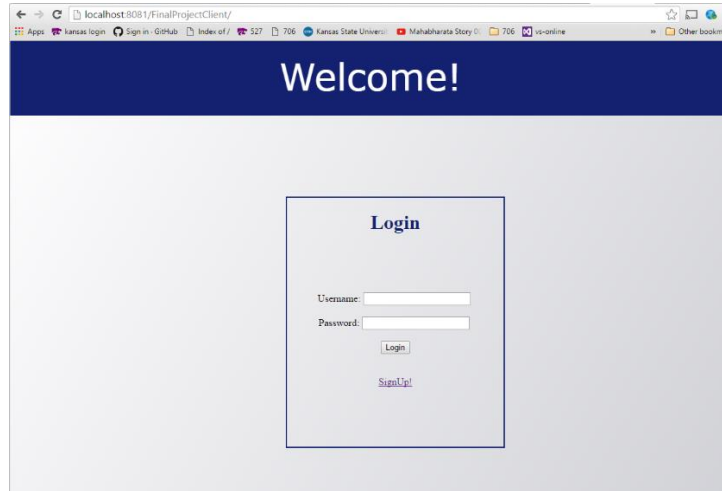


Figure 20: Login Page

2. If a user doesn't have an account, one can sign up here by providing the details. All the fields are validated.

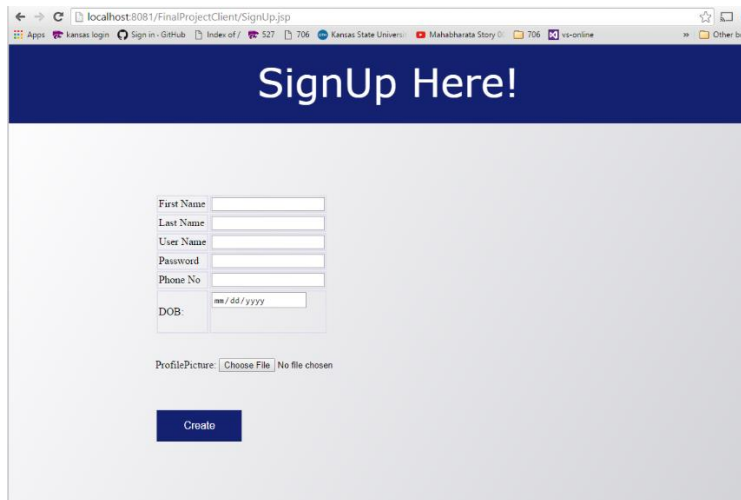


Figure 21: Sign up page

3. If the user enters invalid credentials, he will be directed to login fail page with an error message.

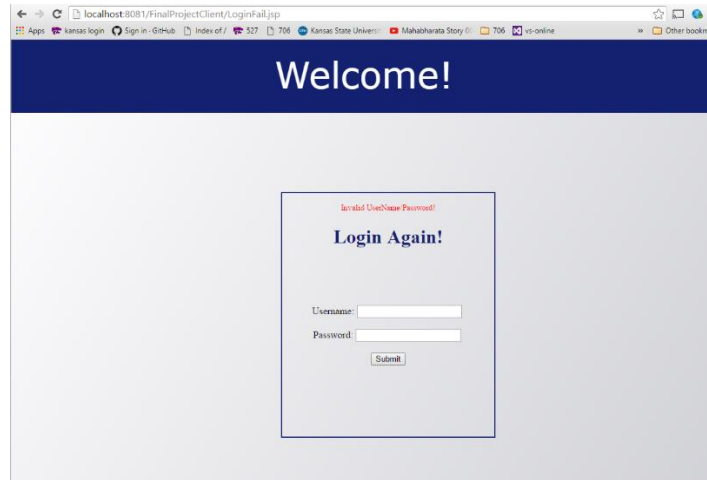


Figure 22: Login Fail

4. Upon successful login the admin users get the following admin login page, with their name on the top.

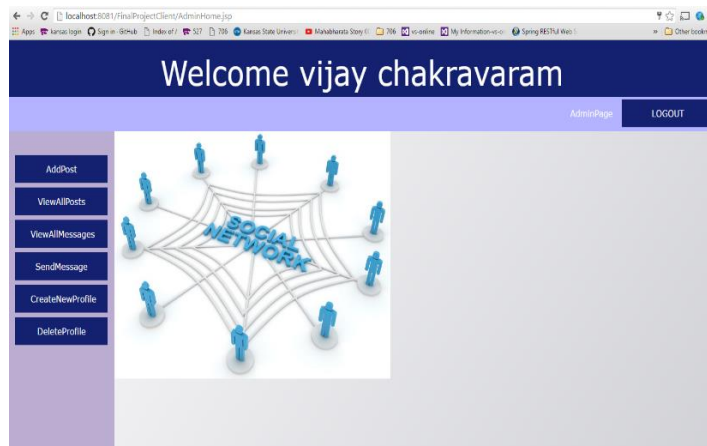


Figure 23: Admin Home Page

5. If the user is not an admin the following will be the home page.



Figure 24: User Home Page

6. The user can type a post here. The text field cannot be empty.

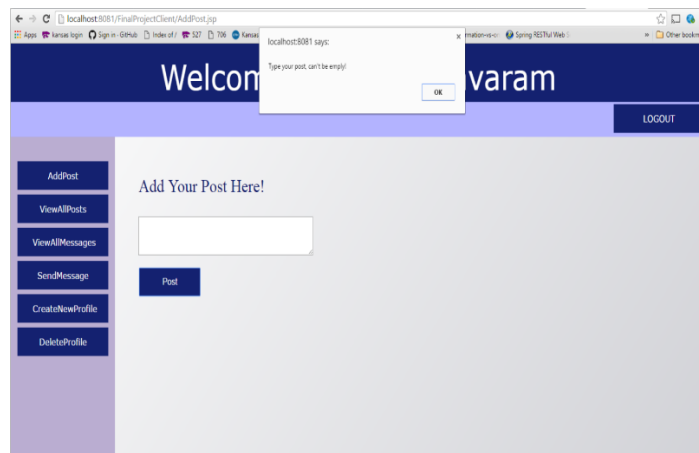


Figure 25: Adding a Post

7. The following appears if the post is successful.

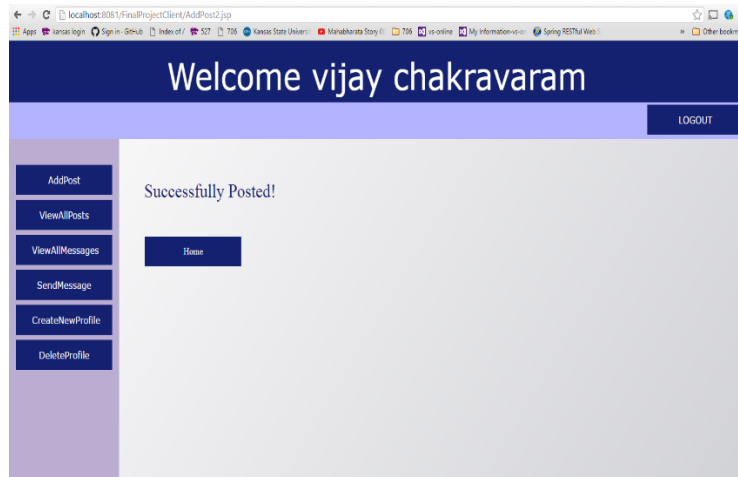


Figure 26: Post Success

8. The user can view all the posts here.

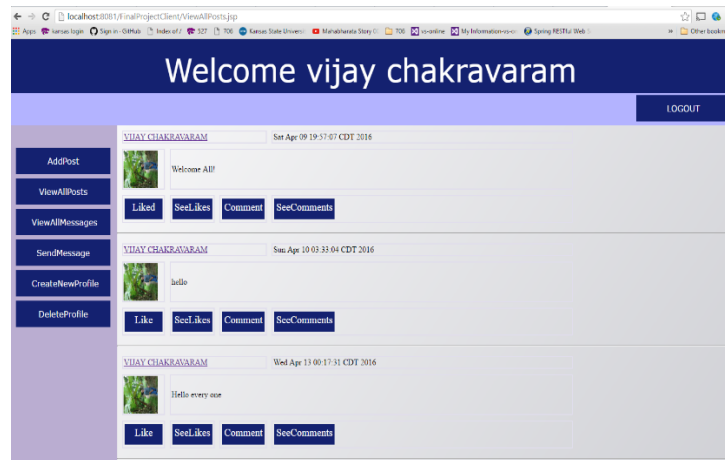


Figure 27: View all posts

9. The user can see all the likes by clicking See Likes button.

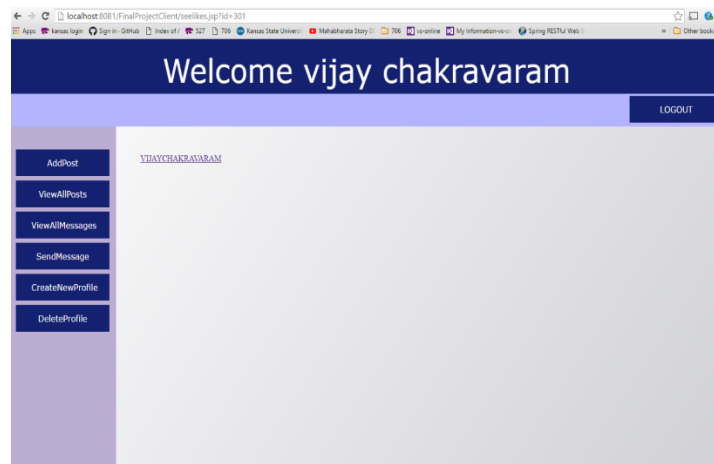


Figure 28; See Likes

10. At any place the user can see the profile of the person.

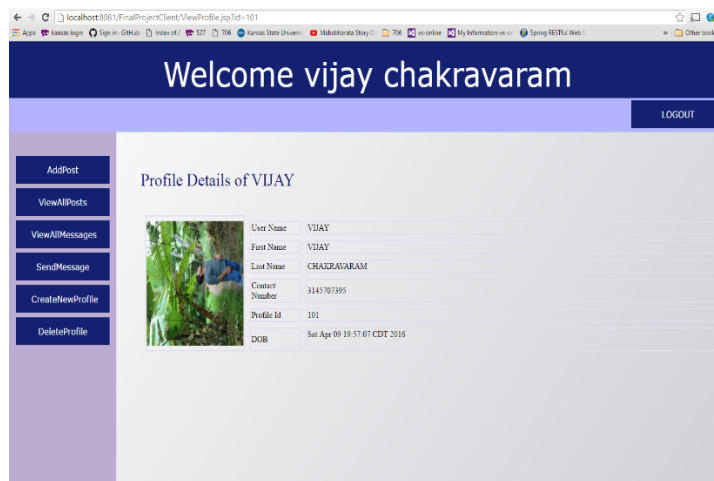


Figure 29: View Profile

11. The user can see all the comments by clicking see comments button.

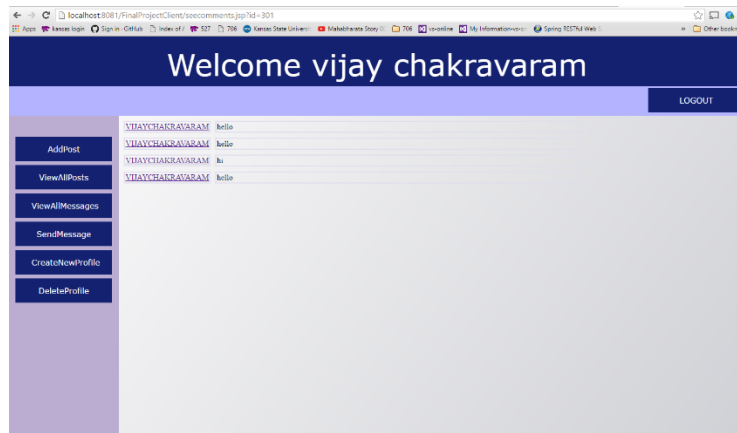


Figure 30: View comments

12. The user can add a comment to a particular post.

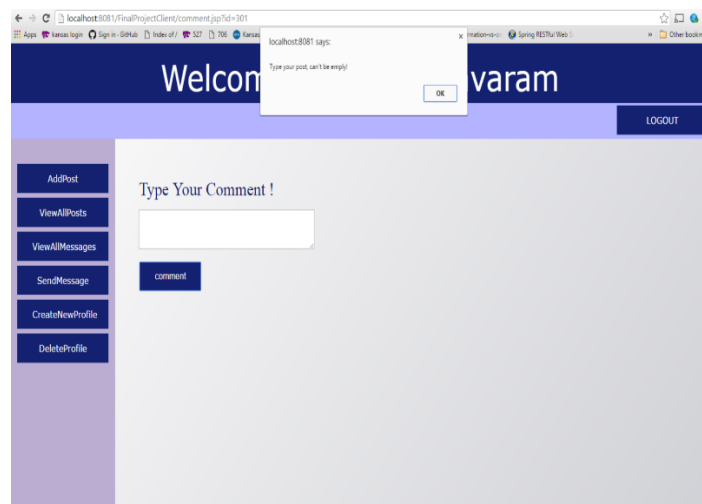


Figure 31: Send Comment

13. Following appears up successfully adding a comment.

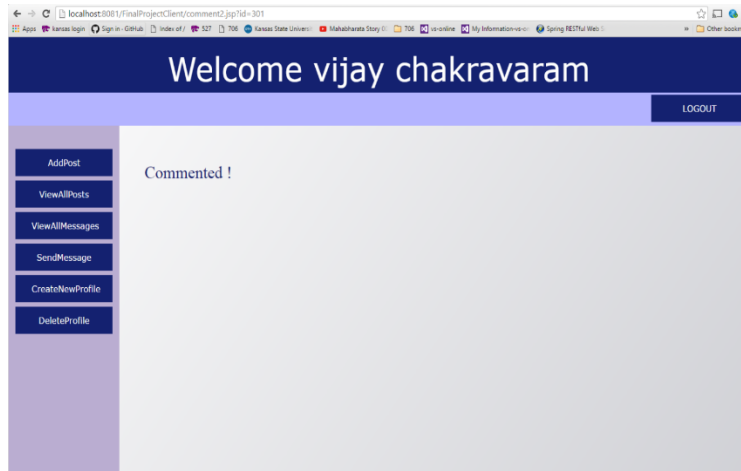


Figure 32: Send Comment Success

14. The user can see all the message by clicking See All Messages button.

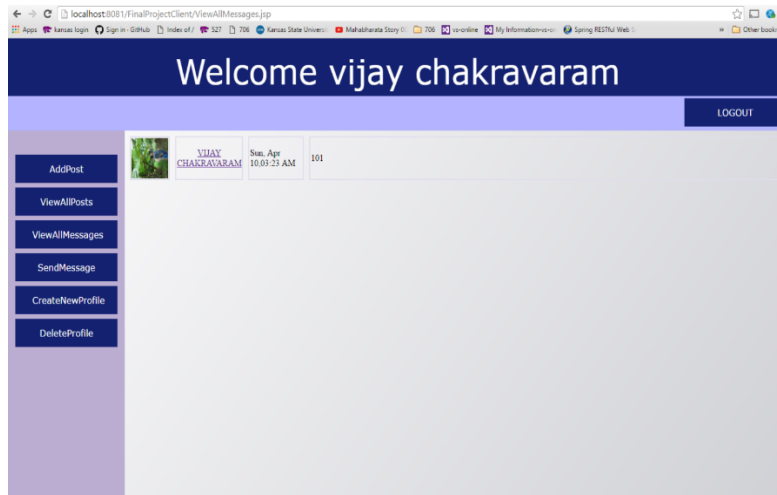


Figure 33: View Messages

15. User can select a profile to send messages.

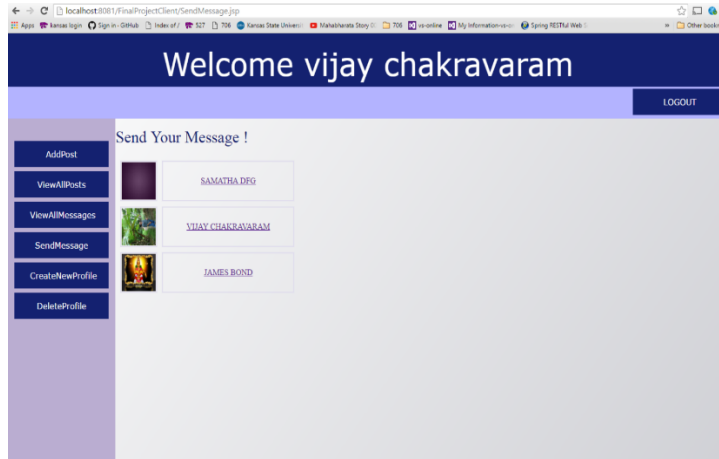


Figure 34: Send Message

16. User can type the message here.

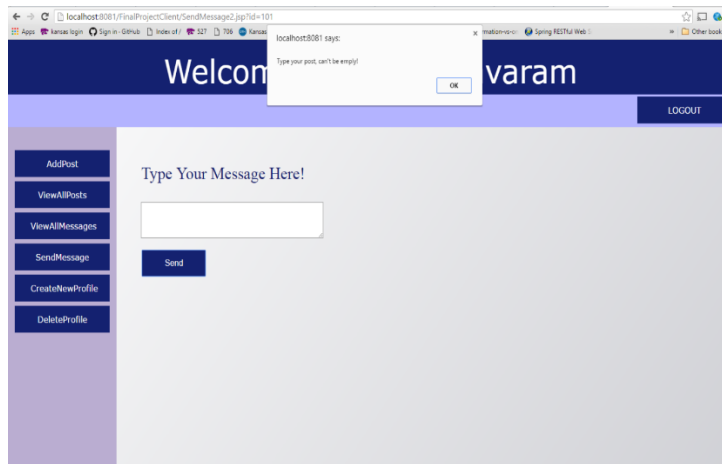


Figure 35: Type Your Message

17. The following appears on sending the message successfully.

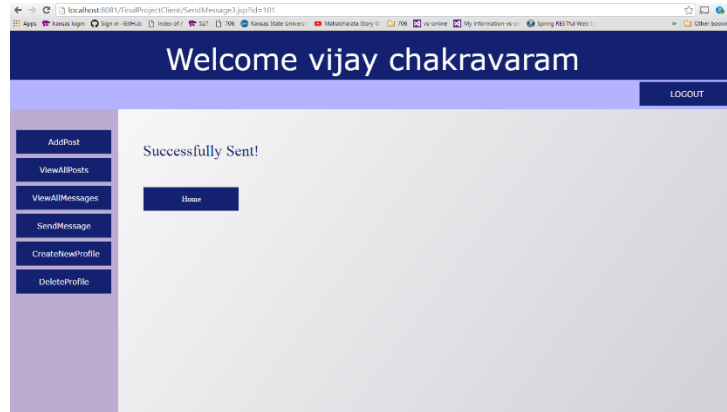


Figure 36: Message Sent Success

18. The admin user can create a profile by clicking *Create New Profile* button.

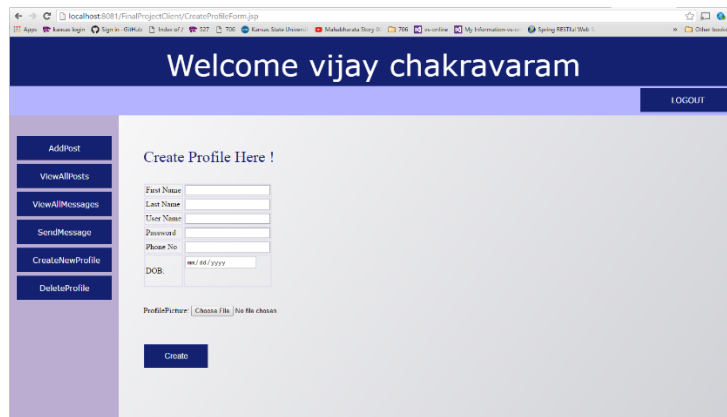


Figure 37: Create Profile

19. The following appears on successful creation of a profile.

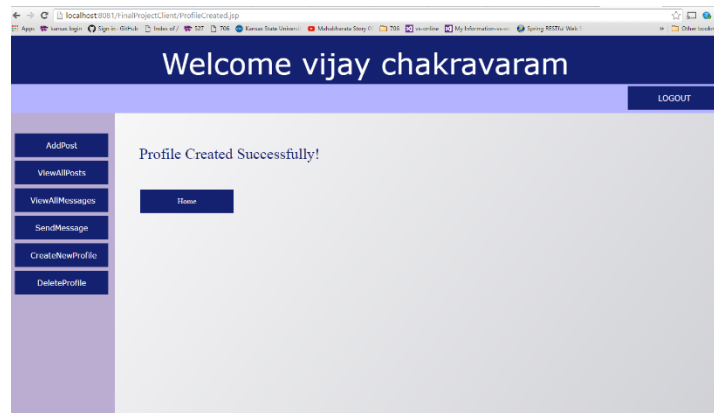


Figure 38: Profile Creation Success

20. The admin user can select the profile to delete it from the system.

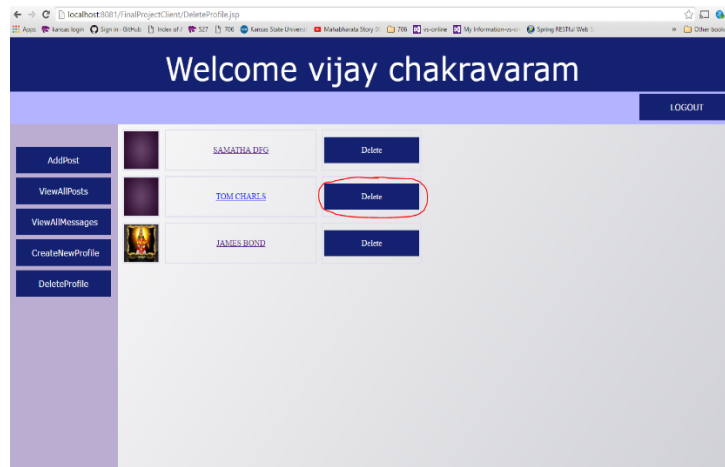


Figure 39: Delete Profile

21. The profile gets deleted from the system.

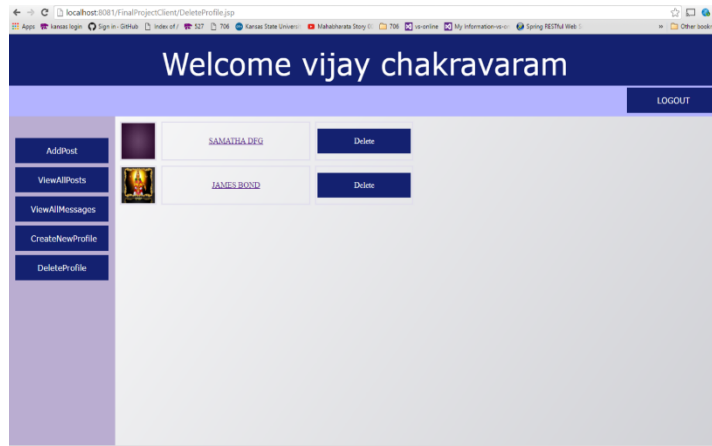


Figure 40: Deleting Profile Success

22. Finally everybody can logout form the system by clicking logout button.

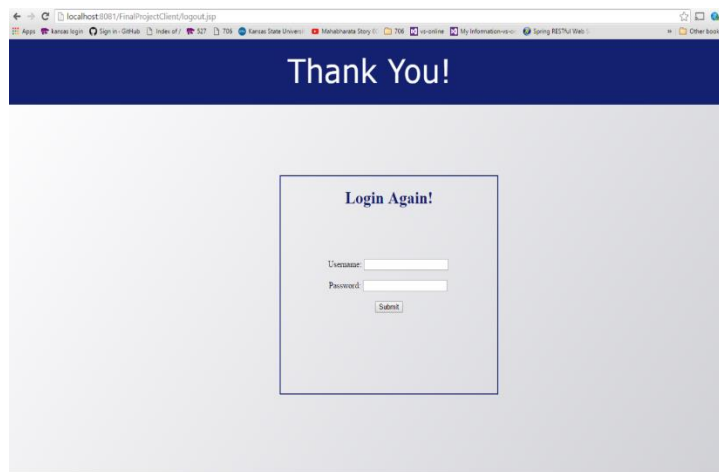


Figure 41: Logout

Chapter 8 - Testing

I have tested the functionality of my application at every stage of development. I first started with writing hibernate code and mapping entity classes to the tables and conformed whether the tables are created as expected or not.

Then I have created web services on top of hibernate code and developed DataAccess code. Once the Web API is ready, I started using postman application to send requests to web API and conformed the correctness of the results.

Then I have developed the client application which access the Web API.

8.1 Testing Web API

In order to handle all the requests coming to the API, the API should be able to handle the exceptions that might generate at the runtime. I have developed my API which handles all the runtime exceptions. The API also respond to requests containing invalid data or if requesting empty data by sending IntegerResponse containing integer value (0 for false/fail and 1 for success/true) and empty objects.

8.1.1 Unit Testing

The following table shows the different responses given by the API for different test cases.

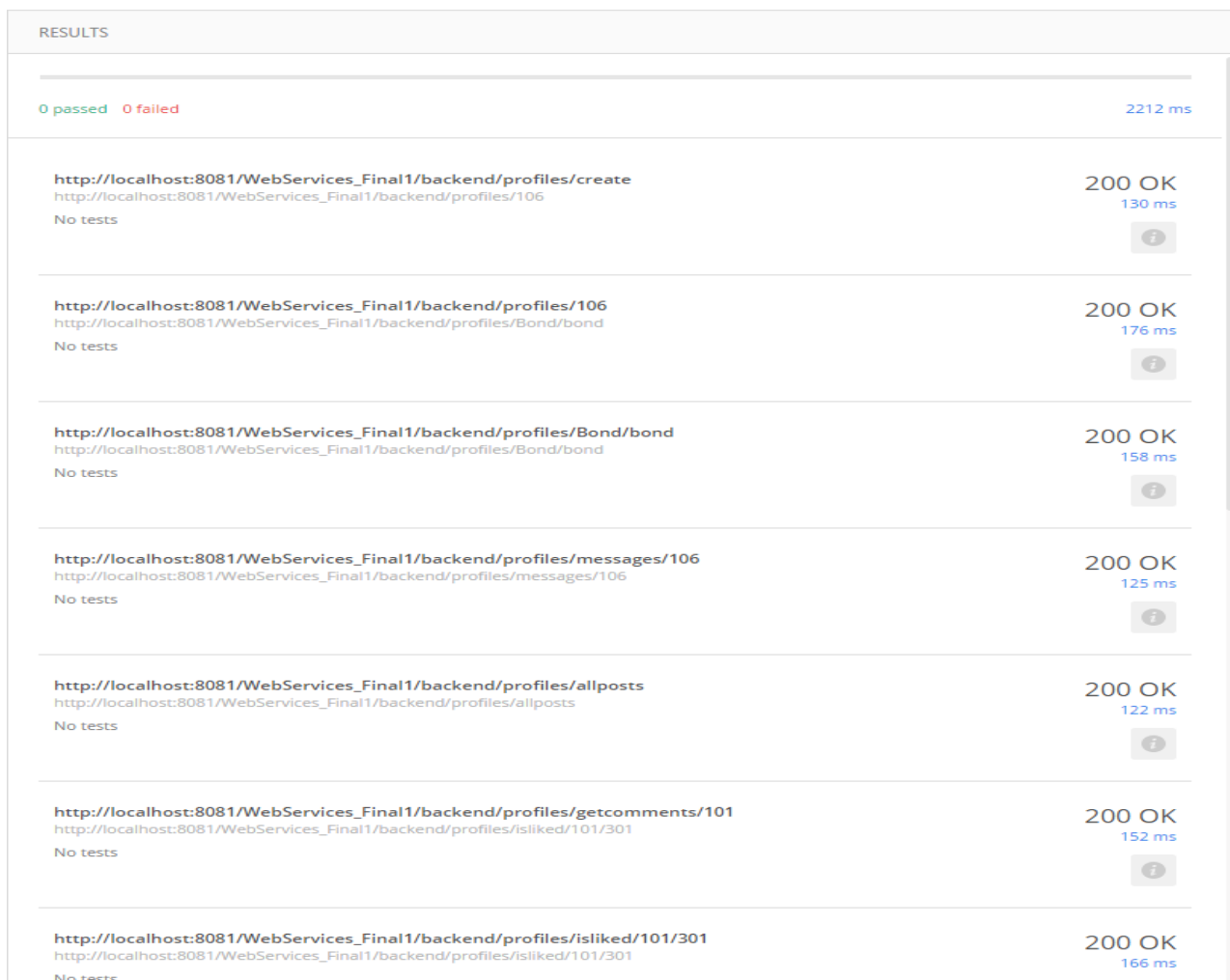
URL Patterns-Input Test Cases	Input	Response
/backend/profiles[/dfklgjdoslfdx]	Provide any Wrong URL	Displays customized error message
/backend/profiles/501	Profile id that's not present	Returns XML of empty profiles class which can be easily handled by the developers
/backend/profiles/delete/501	Delete profile id which is not present.	Returns XML of Integer Response which contains value 0.
/profiles/[wrong username]/[wrongpassword]	Providing wrong username and password	Returns XML of Invalid user object with flag value=0 which implies invalid.
/allposts	Request to display all posts	Returns XML representing empty List object if there are no posts.
/allprofiles	Request to display all profiles	Returns XML of empty List object if there are no profiles.
/getcomments/{wrong post_id}	Request to display comments for non-existing post	Returns XML of empty List object.
/isliked/{profile_id}{post_id}	Asking if non existing profile liked non existing post_id	Returns XML of Integer Response which contains value 0.
/getlikes/{post_id}	Requesting likes for invalid post	Returns XML of empty List object.

Table 1: Table Showing Test Case

8.1.2 Testing Using Post REST Client

Using the postman Rest client tool we can send http requests to our API and get the results. The results can be evaluated for correctness. We can also get the response time for each response from the API which can be used for the evaluating the performance of the Web API.

The following is the image shows response times for different requests. When the collection containing 14 requests is executed, the system took 2212ms to fetch all the requests. So, we can say that the average response time for each response is 170 Milliseconds.



Request URL	Response Status	Response Time
http://localhost:8081/WebServices_Final1/backend/profiles/create http://localhost:8081/WebServices_Final1/backend/profiles/106	200 OK	130 ms
http://localhost:8081/WebServices_Final1/backend/profiles/106 http://localhost:8081/WebServices_Final1/backend/profiles/Bond/bond	200 OK	176 ms
http://localhost:8081/WebServices_Final1/backend/profiles/Bond/bond http://localhost:8081/WebServices_Final1/backend/profiles/Bond/bond	200 OK	158 ms
http://localhost:8081/WebServices_Final1/backend/profiles/messages/106 http://localhost:8081/WebServices_Final1/backend/profiles/messages/106	200 OK	125 ms
http://localhost:8081/WebServices_Final1/backend/profiles/allposts http://localhost:8081/WebServices_Final1/backend/profiles/allposts	200 OK	122 ms
http://localhost:8081/WebServices_Final1/backend/profiles/getcomments/101 http://localhost:8081/WebServices_Final1/backend/profiles/isliked/101/301	200 OK	152 ms
http://localhost:8081/WebServices_Final1/backend/profiles/isliked/101/301 http://localhost:8081/WebServices_Final1/backend/profiles/isliked/101/301	200 OK	166 ms

Figure 42: Postman Rest Client Results

8.1.3 Limitations

Since I am using Hibernate to perform CRUD operations on the data base, hibernate allows only limited number of connections. Hibernate uses built-in algorithm to manage connection pool which is not meant for production or testing purpose. We need to use third party software to manage connection pool for the data base.

8.1.4 Performance testing using jmeter-Test1

When I make a request to the Web API, the API creates connection to the database fetch the results and give them as a response. I have tried this for 250 samples, which means I am sending 250 HTTP GET requests to the API at a time which has to create 250 connections to the database and get the results.

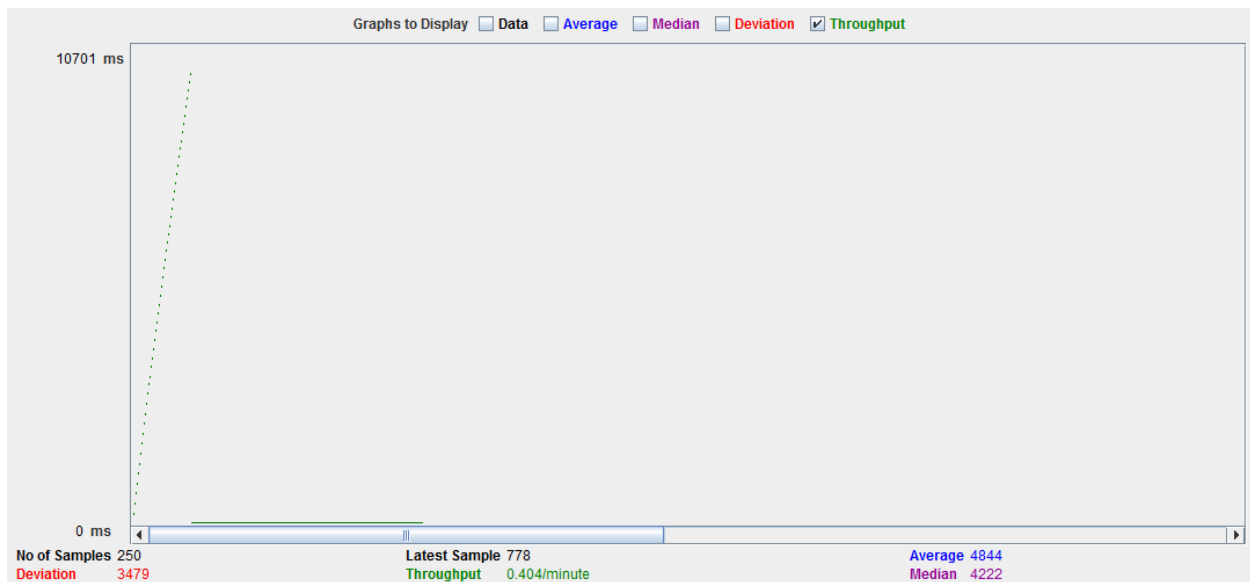


Figure 43: Throughput Graph 1

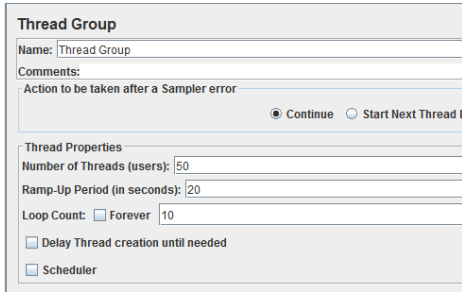
Observations

As Hibernate only supports limited connections per unit time we can see from the graph that around 50 requests are served initially and it dropped later.

Testing with Ramp-up Period and loop count

8.1.5 Performance testing using JMeter-Test2

By setting ramp-up period in JMeter we can give a pause between sending requests to the API. I have set test properties as below where I gave ramp-up period as 20seconds which can be used by hibernate to manage the connection pool.



The screenshot shows the 'Thread Group' configuration window in JMeter. The 'Name' field is 'Thread Group'. Under 'Thread Properties', 'Number of Threads (users)' is set to 50, 'Ramp-Up Period (in seconds)' is set to 20, and 'Loop Count' is set to 10. The 'Action to be taken after a Sampler error' is set to 'Continue'. There are checkboxes for 'Delay Thread creation until needed' and 'Scheduler', both of which are currently unchecked.

Figure 44: JMeter Test Configuration

By configuring the test as above we get the following results

- **Response time:** The response time went down and stabilized.

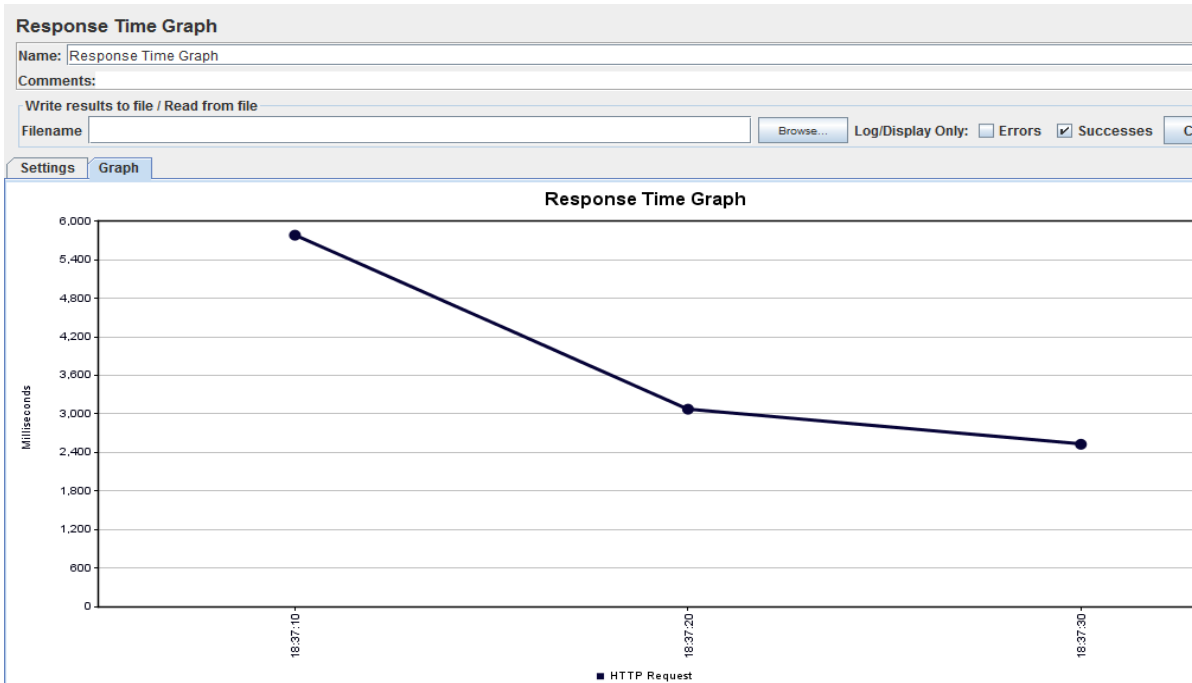


Figure 45: Graph Showing Response Time

Observations: The response time dropped and then stabilized. Initially it takes time for hibernate to get prepared by loading the required components so it took more time to give a response initially. Later, the response time decreased and stabilized as expected.

➤ **Throughput:** The throughput kept growing.

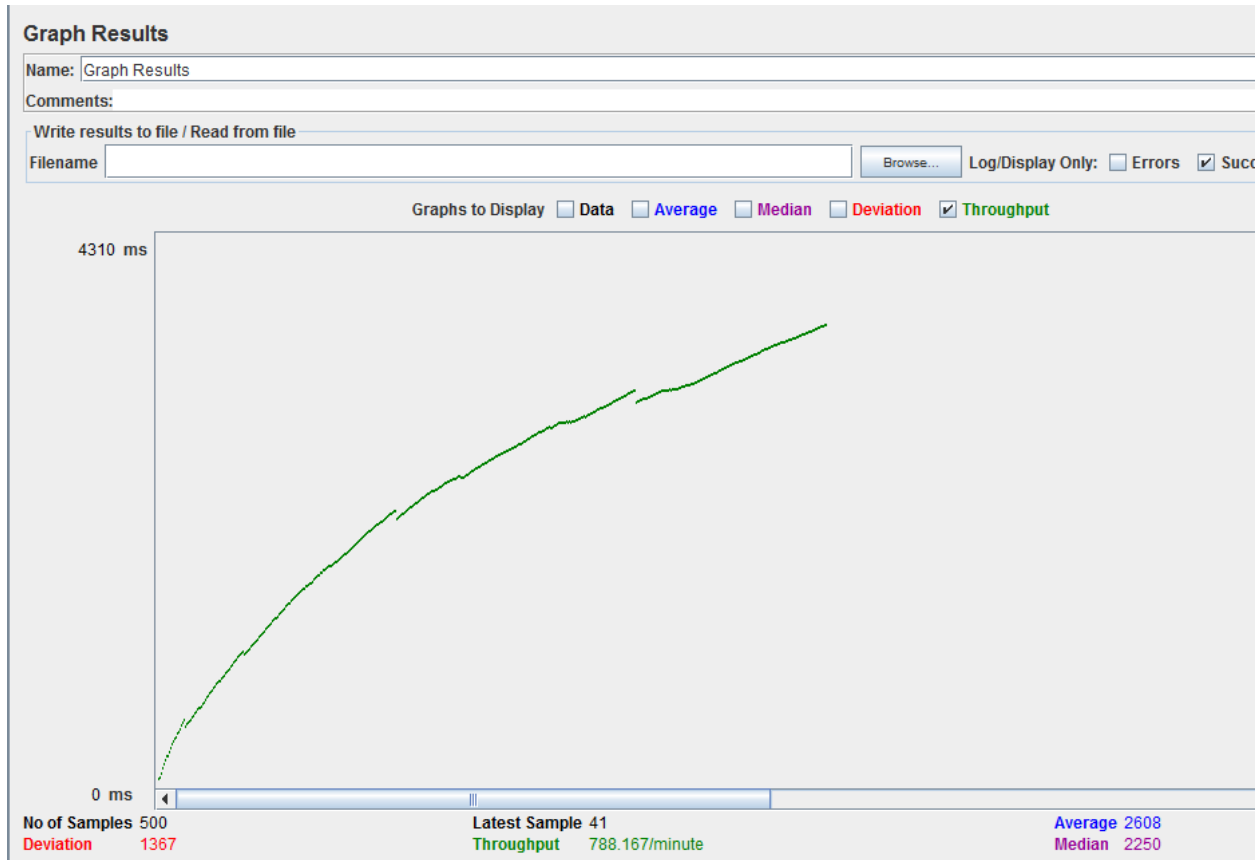


Figure 46: Graph showing the throughput

Observations: The throughput kept increasing and stabilized because of the key factor that hibernate connection pool is able to accept and serve the requests as there is some time gap between the requests.

8.2 Testing Client Application

8.2.1 Unit Testing

I have tested the client application at every stage of development for the correctness of the functionality. Every field is where the user enters data is properly validated using JavaScript.

The performance of the client entirely depends up on the performance of the Web API.

Test Case	Expected Results	Comparing with actual results
Trying to login without providing username/password	Error message	Matches
Trying to login with valid username but wrong password	Error message	Matches
Tring to login with wrong username but correct password	Error message	Matches
Trying to login with wrong credentials	Error message	Matches
Provide correct username and password of admin	Takes to admin home page	Matches
Provides correct username and password of user	Takes to user home page	Matches
Trying to send message with empty text.	Error message	Matches
Trying to upload profile picture not in jpg format	Error message	Matches
Trying to post empty content	Error message	Matches

Table 2: Test Cases Set 1 for Client

Event	Expected Results	Comparing with actual results
Trying to view posts when there are no posts in the database	Display message saying that there are no posts	Matches
Trying to view messages when there are no messages for that profile	Display message saying that there are no messages	Matches
Delete profile by admin user	Profile must be deleted along with all the associated messages, posts, likes and comments.	Matches

Table 3: Test Cases Set 2 for Client

8.3 Performance Testing Using Mozilla Developers Tool

The performance and response times for each operation done by the user are recorded and analyzed for the time taken by different requests and their loading time. I used the Mozilla developer's tool to get the following results.

Each test shows a pie chart for the time taken by different requests to load that particular page from the previous page. The empty cache version shows the results when the request is sent by first time user and the primed version shows the results by using any cache memory if already present in the system. In my case as the graphs would be the same as I am not storing any cache memory on to the browser.

1. To load login Page.

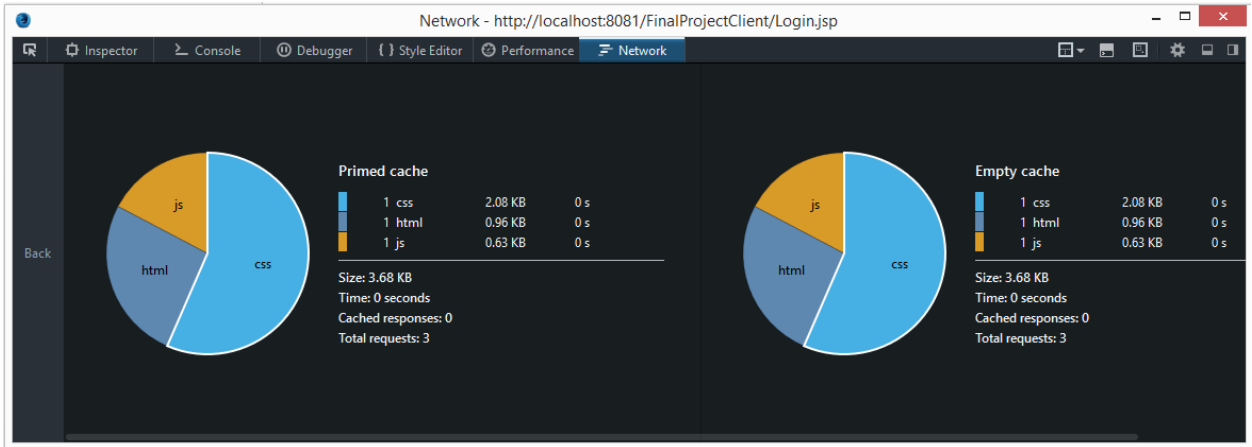


Figure 47: Time taken to load login Page

Observations: The above pie chart shows the total time taken to load the login page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take no time to load the login page.

2. To load home page.

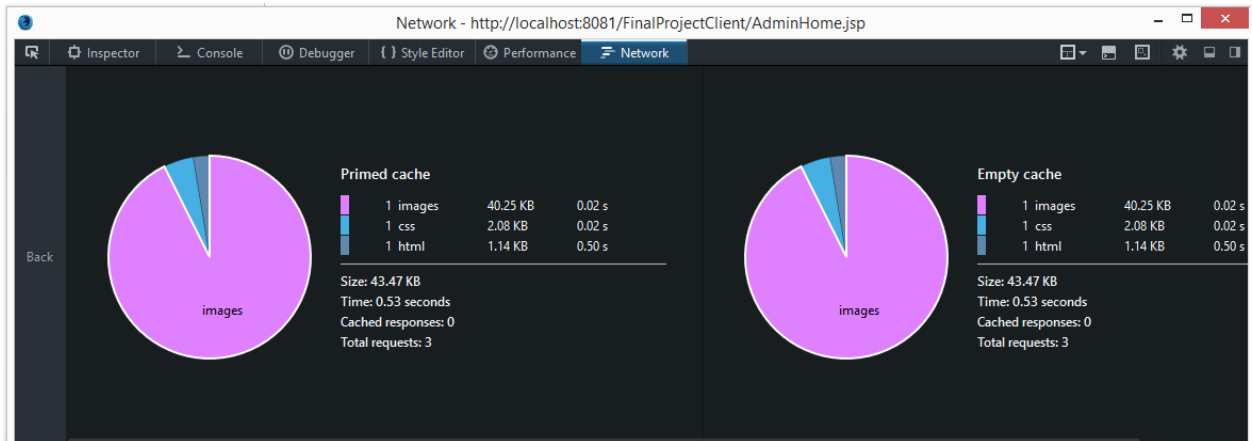


Figure 48: Time taken to load home page

Observations: The above pie chart shows the total time taken to load the admin home page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take 0.53ms to load the page. Out of the data loaded image takes most of it.

3. To display all posts.

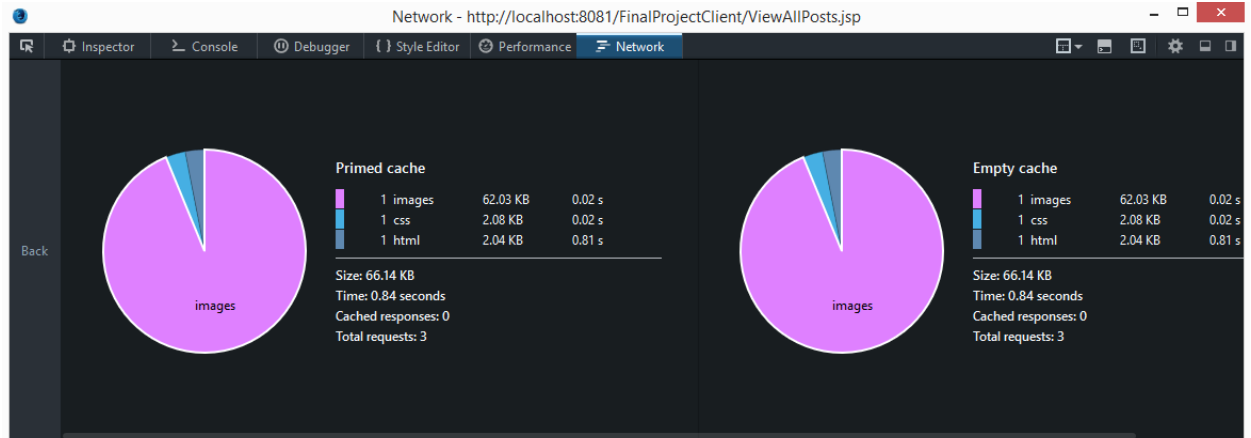


Figure 49: Time taken to display all posts.

Observations: The above pie chart shows the total time taken to display all the posts. It also shows the various elements of the pages, and data, time taken to load those elements. It almost take 0.84ms to load the page. Out of the data loaded image takes most of it.

4. To display all messages.

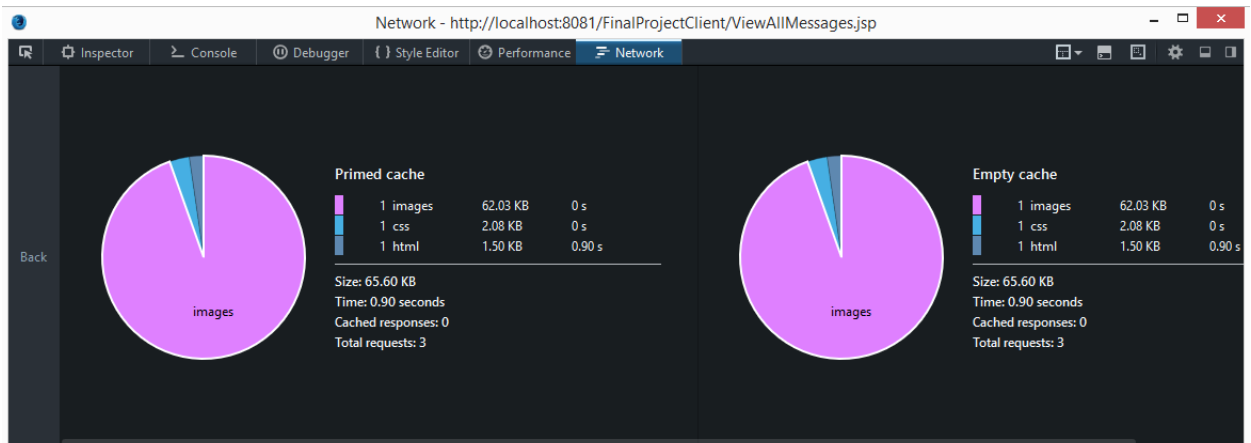


Figure 50: Time taken to display all messages

Observations: The above pie chart shows the total time taken to display all the messages. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take 0.90ms to load the page. Out of the data loaded image takes most of it.

5. To add a post.

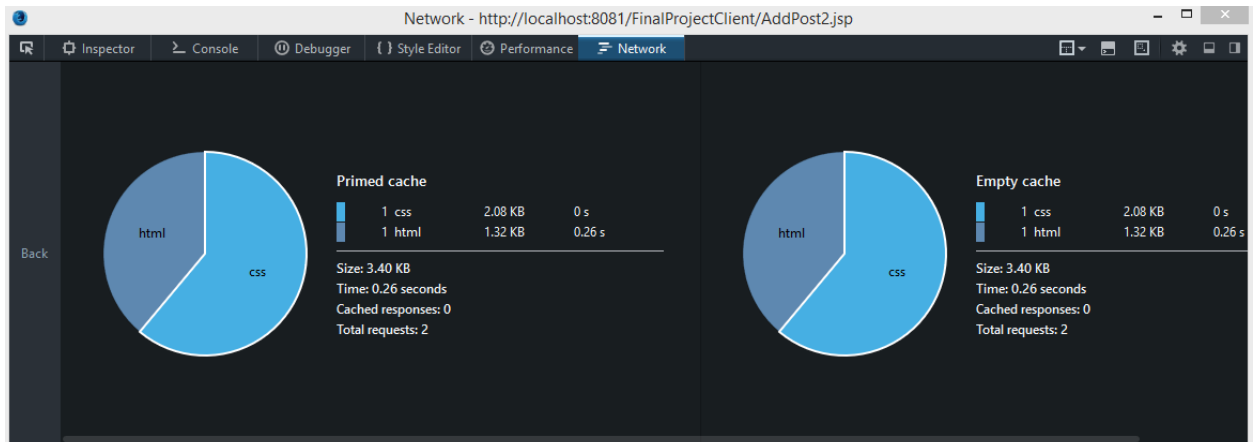


Figure 51: Time taken to add a post

Observations: The above pie chart shows the total time taken to add a post in to database and to load the corresponding page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take 0.26ms to load the page.

6. To Send Message.

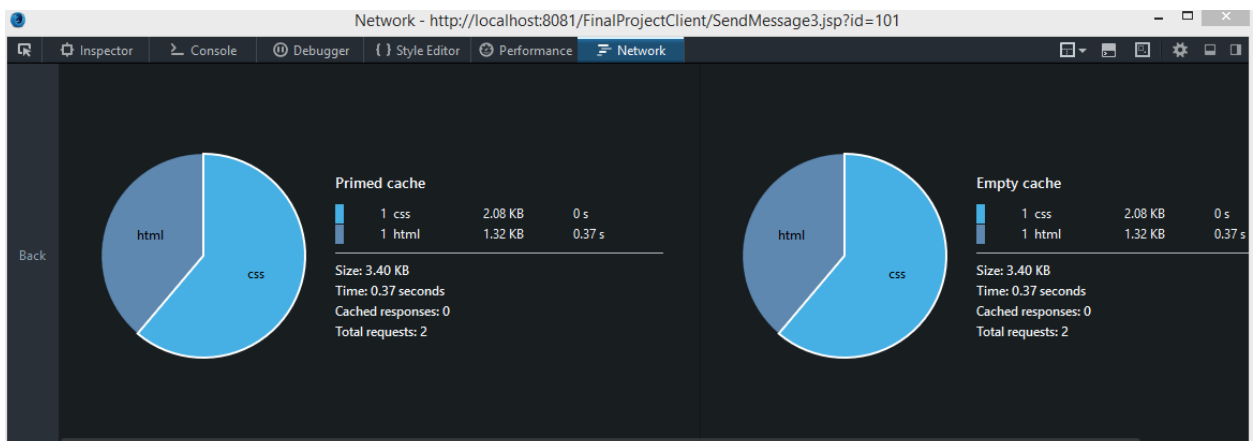


Figure 52: Time taken to Send Message

Observations: The above pie chart shows the total time taken to send a message and to load the corresponding page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take 0.37ms to load the page.

7. To delete a profile

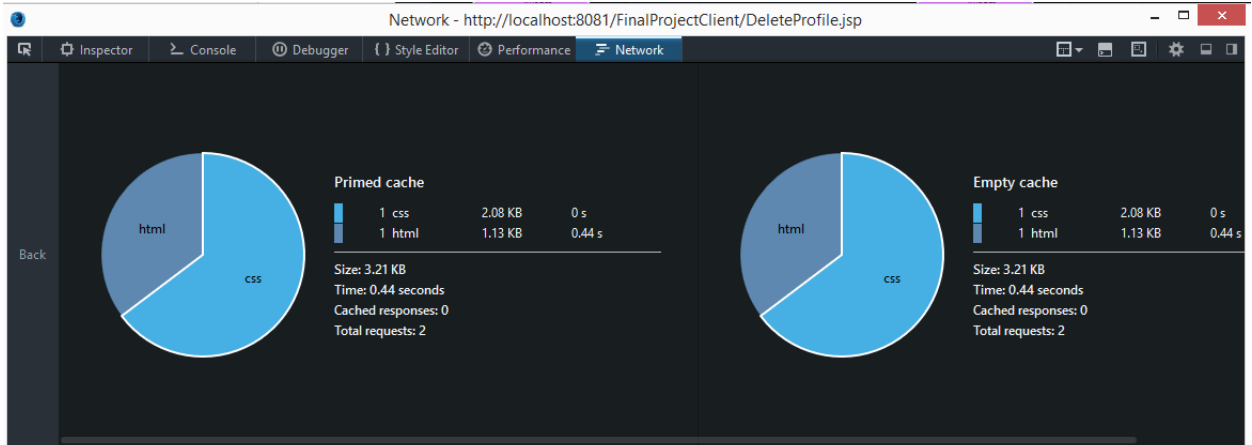


Figure 53: Time taken to delete a profile

Observations: The above pie chart shows the total time taken to delete a profile and to load the corresponding page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take 0.44ms to perform this operation.

8. To create a profile

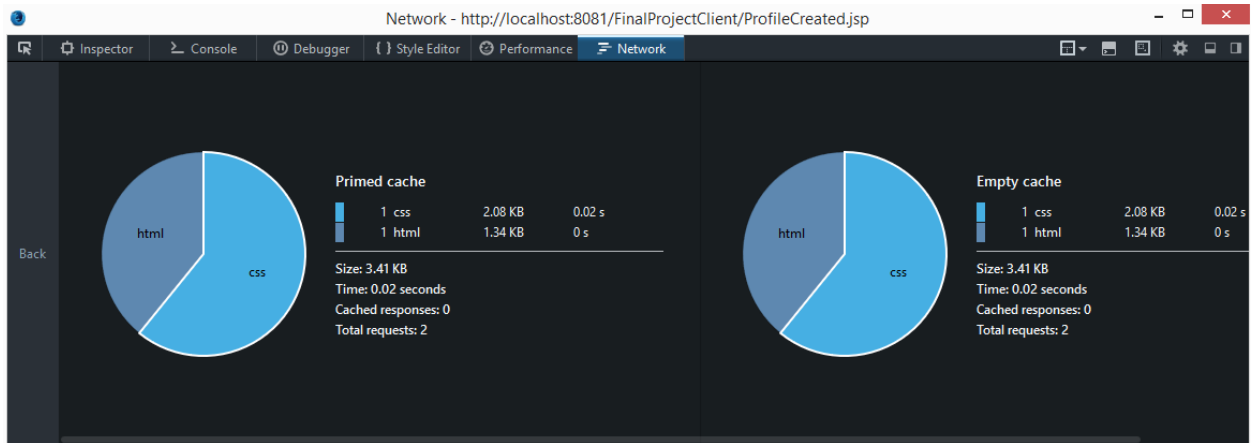


Figure 54: Time taken to create a profile

Observations: The above pie chart shows the total time taken to create a profile and to load the corresponding page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take no time to perform this operation.

9. Time taken to logout.

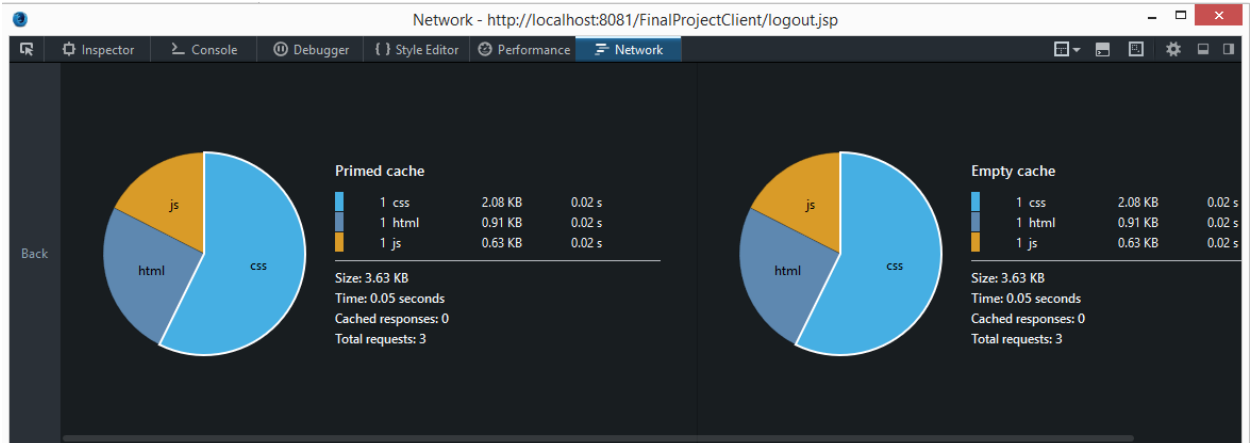


Figure 55: Time taken to logout.

Observations: The above pie chart shows the total time taken to logout from the application to load the corresponding page. It also shows the various elements of the pages, data and the time taken to load those elements. It almost take no time to perform this operation.

Chapter 9 - Learning and Experience

9.1 Learning

Hibernate and Web Services are the new technologies I have learnt and implemented for this project this project.

➤ Hibernate

- Automatically creating tables from classes, imposing integrity constraints and defining relations between them.
- Automatically generating sequences and customized sequences for columns values.
- Performing CRUD operations as required.
- Working with hibernate with and without using hibernate annotations.
- Hibernate Query Language (HQL) to perform crud operations on the underlying database.

➤ JAX-RS RESTful Web Services

- Implementing JERSEY library to develop Web APIs.
- Jersey annotations to develop web services.
- Learnt about Entity/ Model classes and their uses.
- Exception handling in web services.
- Securing Web API.
- Maintaining MVC architecture while implementing.

- Client Application
 - Marshalling and Un-marshalling objects in to XML classes and vice versa.
 - Creating a client which can call GET, POST, PUT and DELETE methods of HTTP and make a request to web services.
 - Sending and receiving images as byte streams embedded in XML.
 - Maintaining MVC architecture of the application.
- Tools and Configuration
 - JMeter for testing the response time and throughput of the application.
 - Mozilla developer tool to analyze the response times of various requests.
 - Postman rest client to send requests and check the correctness of web services.
 - OpenSSL to generate self-signed SSL certificate.
- Configurations
 - Configuring hibernate.xml file according to the requirements.
 - Configure class.hbm.xml to define constraints on generating tables
 - Server.xml to handle SSL security.
 - Web.xml of server to configure user roles and their passwords.
 - Web.xml of server application for security and authentication.

9.2 Project Development Experience

It took around time for me to learn the concepts of hibernate and web services from the basics. I have wrote many test programs in the process of learning each concept and implementing them. Once, I learnt all the concepts required to implement my idea I started the design and implemented it. I first started with developing the hibernate code which manipulate the database. Then I have developed Web services on top of it which completes the development of fully functioning Web API.

Then, I have started developing the Client application which sends requests to the Web API and also handles responses from it. I initially developed the client application with minimal UI which is enough to serve the purpose. Once the full functionality is achieved I wrote code to improve the look and feel.

Chapter 10 - Future Work

The following can be done project to still improve the performance and the usability of the system.

- Use third party services to maintain the connection pool which makes large number of people to use the application simultaneously.
- Implement cache memory at the client side to store some basic data which increases the response time and decreases the number of requests to the API.
- Improve the look and feel of the application by using Ajax, jQuery and related technologies.
- Completely replace JSPs with other frame works to improve the performance.
- Extend the functionality to support various media in posts.
- Implement share option and can think of implementing any other feature provided by Facebook.

Chapter 11 - Conclusion

This project works fine which can be easily used by any organization as an internal social networking application. They can customize the look and feel of the application according to their need and requirements. The organization need not worry about maintaining the database and background functionality. The data is always secured with the use of SSL connections and authentications. The client application developed is very to use even for the fist-time user. They can customize the GUI as they need which is really a cool option.

Chapter 12 - References

The following are the references I have used.

- [1] Oracle Corporation, you can find the documentation and tutorials of web services.

<https://docs.oracle.com/javase/6/tutorial/doc/bnayk.html>

- [2] Chemaxon, for architecture of web services.

<https://www.chemaxon.com/products/jchem-web-services/>

- [3] Java tutorial point, for hibernate architecture

<http://www.javatpoint.com/images/hibernate/architecture.jpg>

- [4] Java tutorial point, for MVC architecture

http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm

- [5] Jersey website for information about RESTful Web Services

<https://jersey.java.net/>

- [6] Jersey website for documentation, more information about annotations.

<https://jersey.java.net/documentation/latest/jaxrs-resources.html>

- [7] Open SSL, details on generating self-signed SSL certificate.

<https://www.openssl.org/>

- [8] Tomcat, website to more about configuring server xml files.

<https://tomcat.apache.org/tomcat-7.0-doc/config/>