

Student Registration System using Java Server Faces

Prashanth Kumar Daram ¹, Jeevan Reddy Gopu ², and Jinan Fiaidhi ²

¹Lakehead University

²Affiliation not available

October 30, 2023

Abstract

Java Server Faces is a framework to build a web application in java. We using Java Server Faces framework to build Student Registration System. We using MVC architecture in our model. In this paper, we given detail explanation about all the components which are helpful to build the application.

Student Registration System using Java Server Faces

Prashanth Kumar Daram
Student ID: 111650
pdaram@lakeheadu.ca
Department of Computer Science
Lakehead University, Canada

Jeevan Reddy Gopu
Student ID: 1105340
jgopu@lakeheadu.ca
Department of Computer Science
Lakehead University, Canada

Dr.J.Fiaidhi
Instructor
jfiaidhi@lakeheadu.ca
Department of Computer Science
Lakehead University, Canada

Abstract—This paper describes the implementation of a web-based application for the student registration system using Java Server Faces (JSF). JSF is a standardized specification for building User Interfaces (UI) for a client and server-side based applications. We all know the struts framework which is popular for building JSP web-based application framework and Swing framework for desktop applications. JSF is a mixture of the two frameworks. The lifecycle of a web application is managed by JSF through a controller servlet. JSF provides an enhanced mechanism to handle events and to render the components. This paper describes briefly about introduction to JSF and what it means and how we use that to create several applications using JSF UI components.

keywords—*client, server, managed bean, framework, MVC framework, UI component.*

I. INTRODUCTION

Java Server Faces is a framework to build a web application in java. Java server faces is a framework but not technology. A framework is a platform to develop software applications and to accelerate application development. A framework provides a template, which helps to develop an application or customize an application. It can also be defined as a semi-implemented application that can use to design enterprise applications more simply. With the help of framework, developer work and time are reduced so automatically development cost is also reduced and productivity increases. Frameworks help to develop a complex application in less time. So, most of the software industries use frameworks to develop a web application. The architecture it uses and the components which are used to build an application is mentioned. The lifecycle of JSF is described below.

A. Different Types of Frameworks

There are two types of frameworks in use. They are web-based and application-based. Web framework provides a good environment to design and develop web applications and application framework provides a good environment to

develop the distributed application. The main difference between web frameworks and application framework is a web server responds to only web application request whereas the application server responds to both web application and mobile applications.

II. OVERVIEW OF JAVA SERVER FACES

There are different frameworks to build web applications such as struts, spring MVC, plain servlets and JSP. JSF simplifies the integration of application development and it acts as a standard display technology that was formalized through a java Community process. The JSF follows industry patterns such as it uses Model-View-Control (MVC) architecture. It simplifies the building of user interfaces (UI) for server-based applications using reusable components. It provides methods to connect the widgets of UI to the data sources and event handlers on the server-side.

A. JSF Benefits

- JSF allows reusable UI components and extendable.
- JSF helps in managing the application state for web requests.
- It performs data validation and conversion.
- Easy Data Transfer between UI Components.
- Managing UI state across multiple servers.

III. MVC ARCHITECTURE

JSF uses the Model View Controller (MVC) architecture for building the application. MVC architecture separates logic from the presentation which helps the developers to concentrate more on the actual content and helps in collaboration. In this architecture, the application is divided into three individual components termed as the model, the view, the controller. The task of a controller is to process the application by routing the request. The model provides access to the database; it can retrieve and insert data in the database. The data and logic are maintained by the model. The view renders the Html page to the user. MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. The block diagram of a MVC architecture has been shown in figure 1.

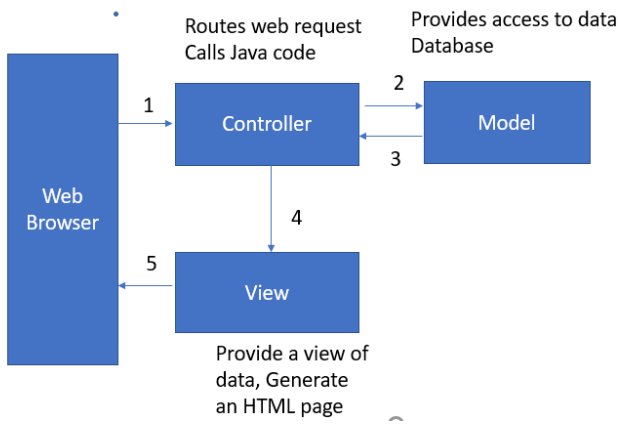


Fig. 1. Block Diagram of MVC Architecture [1].

The user uses a web browser as an interface to submit form data then the request comes to the controller. The controller specifies which part of the code should be executed to serve the request. The controller can access a model, the model provides access to backend data. It retrieves and updates data for the user from the back end. Once the controller gets the data then it passes to view. View renders the result of the query to the user.

IV. JSF APPLICATION LIFE CYCLE

There are six phases involved in a JSF application. The six phases are:

A. Restore View Phase

This phase gets activated as soon as the user initiates or clicks the link or the button and the JSF receives the request. During this, the JSF builds the view and wires event handlers and validators to UI components and saves the view in the face's context instance. The faces context instance will now contain all the information required to process a request.

B. Apply request Phase

In this phase after the creation and restoration of the component tree each component in it uses the decode method to extract its new values from request parameters. Component stores this value. If a conversation is failed an error message is generated and queued on faces context. The message gets rendered on the render response page as well as validation errors are displayed. If the event listeners of the decode methods call render responses on the current Faces Context instance the JSF moves to render response page.

C. Process validation Phase

All validators are processed in this phase that is on the component tree. The attribute rules are checked for validating and compares these values with the local values that are stored for a component. If the local values come invalid, then an error message is generated to faces context instance and it is shifted to render response page and displays the same page with an error message.

D. Update Model values phase

When the data is proved valid the component tree is taken into consideration and the server-side object properties are set to the local values of the components. The bean properties are updated corresponding to input components value attributes. If the update model method calls render response page on the present faces context instance the process gets shifted to render response page.

E. Invoke Application phase

The application-level events are handled and some of the examples of these are linking the related web pages to each other, form submission and verification, etc.

F. Render response Page

On this page, the application server is asked to render the page if that server is using JSF pages. The components on the page are added to the tree for the initial request purpose as the page gets executed. This step occurs if it is an initial request or else, they are not added again. In both cases, the components will be rendered as the JSF application server renders the phase. Now the content that appears in the view phase is rendered and the response state is saved so that the requests will be able to access it and restores the view phase.

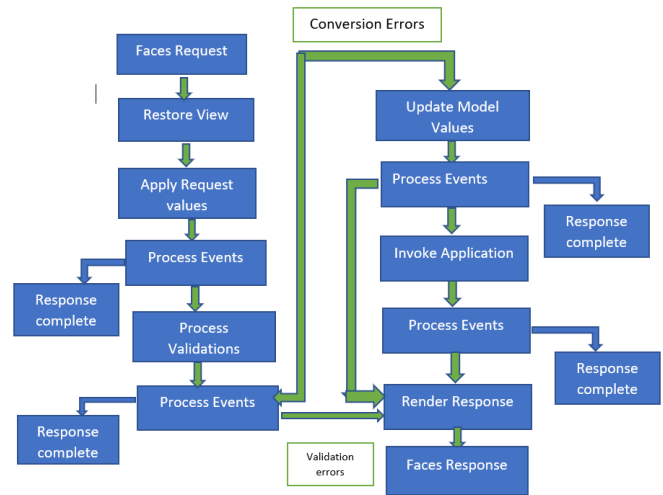


Fig. 2. Life Cycle of JSF [1].

V. JSF COMPONENTS

A. Components of JSF application

- A set of web pages to layout UI components.
- A set of managed beans.
- A web deployment descriptor (web.xml file).
- Optionally you can drop some custom objects, custom tags, components and validators.

B. Third-Party components

JSF allows third party UI components to include in the application. There are many third-party UI components on the internet. We can download it from the internet and can include it in our application.

C. JSF managed bean

It is nothing but a simple java class and it contains all the properties involved and it has getter and setter methods. It is sometimes lazy, and it is instantiated only when the request is made from the application. A bean can be forcefully placed in the application scope as soon as the process is started. It acts as a framework for JSF models. There will be certain functions which a managed bean will be performing, and they are

- It handles events that are created by components.
- Component data validation.
- It helps in navigating between the pages.

```
1 public class User {
2     private String UserName;
3     private String Password;
4     public String getUsername() {
5         return UserName;
6     }
7     public void setUsername(String UserName) {
8         this.UserName = UserName;
9     }
10    public String password() {
11        return Password;
12    }
13    public void setPassword(String Password) {this.
14        Password = Password;
15    }
```

Listing 1. Managed bean example code

D. Servlet

The capabilities of a server are extended by a servlet. Web containers are mostly implemented by the servlet even though many types of requests come. Servlets are mainly used to add the dynamic or memory content to the webserver. It stores a java class.

E. FacesServlet

It is a servlet that manages the request processing lifecycle for web applications that are utilizing Java Server Faces to construct the user interface.

F. Deployment Descriptor

A web application's deployment descriptor describes the classes, resources, and configuration of the application and how the web server uses them to serve web requests. When the webserver receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code to serve the request. The deployment descriptor is a file named web.xml.

VI. JSF UI COMPONENT TAGS

A. JSF Basic Tags

JSF provides a various HTML tag library. To implement these tags in the JSF application, we need to use namespaces of URI in the Html code.

```
1 <html
2     xmlns= http ://www.w3.org/1999/xhtml
3     xmlns:h= http ://java.sun.com/jsf/html >
```

Listing 2. JSF Basic Tags syntax

Some of the JSF basic tags are

- **h:inputText** :Exhibits a HTML input of type="text", text box.
- **h:inputText** :Exhibits a HTML input of type="password", text box.
- **h:inputTextarea** :Exhibits a HTML text area field.
- **h:inputHidden** :Exhibits a HTML input of type="hidden".
- **h:selectBooleanCheckbox**:Exhibits a single HTML check box.
- **h:selectManyCheckbox**:Exhibits a group of HTML check boxes.
- **h:selectOneRadio** :Exhibits a single HTML radio button.
- **h:selectOneListbox**:Exhibits a HTML single list box.
- **h:selectManyListbox** : Exhibits a HTML multiple list box.
- **h:outputText** :Exhibits a HTML text.
- **h:outputStylesheet** :It inserts a CSS style sheet in HTML output.
- **h:message** :Displays message for a JSF UI Component.
- **h:graphicimage** :Renders an image.

B. JSF Page Navigation

JSF provides navigation rules that describe which page is to be shown when a button or link is clicked. The basic tags used in page navigations are

- **h:commandButton** : Displays a HTML input of type="submit" button.
- **h:Link** : Displays a HTML anchor.
- **h:outputLink** : Displays a HTML anchor.
- **h:commandLink** :Displays a HTML anchor.

C. JSF Facelet Tags

Some special tags are provided by JSF to create a common layout for a web application called Facelet Tags. These tags help in managing common parts of multiple pages in one place. To implement these tags in JSF application, we need to use name spaces of URI in html code.

```
1 <html
2     xmlns = http ://www.w3.org/1999/xhtml"
3     xmlns:ui = http ://java.sun.com/jsf/facelets >
```

Listing 3. JSF Facelet Tag Syntax

We can use following tags while creating a template.

- `ui:insert`
- `ui:define`
- `ui:include`
- `ui:composition`

VII. LITERATURE REVIEW

With the rapid growth of technology, the usage of online web pages has been increased to perform daily works such as online shopping websites, travel booking websites and grocery purchase websites. The demand for developing websites has been increased. With this increased demand many technologies have been developed to create an online web application. Most of the people used HTML and servlets to develop web pages but it has some disadvantages. This technology does not allow to reuse the components and connecting to the database is a tough task with this technology. To overcome this problem, Sun microsystem developed java technology to build a web application. The technology is called Java Server Faces (JSF), it has many advantages compared to other technologies such as struts and spring. The author of [4] developed a hotel reservation system using JSF technology. It minimizes the development cost and time and generates the final output according to our requirements. In this paper student registration system has been developed based on JSF technology.

VIII. PROPOSED MODEL

A. Environment Setup

To build JSF applications we need the following

Java Application Server- There are different Java Application servers such as Tomcat, Glassfish etc. We are using Tomcat as our server.

Java Integrated Development Environment- The IDE may be anything like Text pad, note pad, NetBeans, Eclipse. Here We are using Eclipse.

JSF Jar Files- JSF is an API. It has number of classes, Interfaces that we must use. We can do it by downloading jar files and we need to include them in a project. Eclipse will automatically download the jsf jar files and includes them in project.

B. Installing Tomcat on windows

- Step-1: Download the Tomcat. Website for downloading tomcat is <http://tomcat.apache.org>.
 - Step-2: Setup the environment.
 - Step-3: Verify the installation.
- Just go to the browser and type this link then it shows Tomcat server: <http://localhost:8080>.

C. Installing Eclipse on windows

- Step-1: Download the Eclipse. Website for downloading Eclipse is <https://www.eclipse.org/downloads/packages/release/neon/3/eclipse-ide-java-ee-developers>.

- Step-2: Unzip the Eclipse
Extract the file and place the file in C folder.
- Step -3: Run the eclipse.

D. Connecting Eclipse and Tomcat

Here Tomcat and Eclipse both are separate applications, by connecting Tomcat and eclipse we can access or start tomcat from eclipse. We can easily deploy JSF applications directly to Tomcat.

E. Developing Student registration Form with JSF

Step 1: Create the login.xhtml page

Let's create a login page using JSF UI components. On this page, we are using an input text field and password components.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
  transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5 xmlns:ui="http://java.sun.com/jsf/facelets"
6 xmlns:h="http://java.sun.com/jsf/html"
7 xmlns:f="http://java.sun.com/jsf/core">
8
9 <ui:composition template="/basictemplate.xhtml">
10 <ui:define name="content">
11 <h:form>
12 <h:panelGrid columns="2">
13 <h:outputText value="name"></h:outputText>
14 <h:inputText value="#{loginbean.name}"></h:
  inputText>
15 <h:outputText value="password"></h:
  outputText>
16 <h:inputSecret value="#{loginbean.
  password}"></h:inputSecret>
17 </h:panelGrid>
18 <h:commandButton value="login" action="login"
19 ></h:commandButton>
20 </h:form>
21 </ui:define>
22 </ui:composition>
23 </html>
```

Listing 4. Example of login.xhtml file from student registration system.

step 2: Create the Response.xhtml

The response page is created using JSF UI components. It displays the form data to the user.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
  transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5 xmlns:ui="http://java.sun.com/jsf/facelets"
6 xmlns:h="http://java.sun.com/jsf/html"
7 xmlns:f="http://java.sun.com/jsf/core">
8
9 <ui:composition template="/WEB-INF/template/
  basictemplate.xhtml">
10 <ui:define name="content">
11 <h:outputLabel value="Welcome #{loginbean.
  name}"></h:outputLabel>
12 <br/><br/>
13 <h:link value="Home Page" outcome="
  homepage"/>
14 </ui:define>
```

```

15 </ui:composition>
16 </html>

```

Listing 5. Example of response.xhtml file from student registration system.

Step-3: Creation of loginbean.java.

The JSF mostly recommends the Post method to send input data from an online form to an application server but, sometimes there is a strong need for the use of the Getting method instead. In the student registration form, the Get method is used during the registration process for getting activating student account. The set method is used to set the values. In eclipse, we don't need to create getter and setter methods. Eclipse automatically generates getter and setter methods activating student account. Set method is used to set the values.

```

1 package com.tutorial;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.PreparedStatement;
6
7 public class loginbean {
8
9     private String name;
10    private String password;
11    public String getName() {
12        return name;
13    }
14    public void setName(String name) {
15        this.name = name;
16    }
17    public String getPassword() {
18        return password;
19    }
20    public void setPassword(String password) {
21        this.password = password;
22    }
23 }

```

Listing 6. Managed bean file of student registration system.

Step 4: Connecting to JDBC server

```

1 public boolean save(){
2     int result = 0;
3     try{
4         Class.forName("com.mysql.jdbc.Driver");
5         Connection con = DriverManager.getConnection("
6             jdbc:mysql://localhost:3306/Emp","prashu","");
7         PreparedStatement stmt = con.prepareStatement("
8             insert into user(name,password) values(?,?)");
9         stmt.setString(1, name);
10        stmt.setString(2, password);
11        result = stmt.executeUpdate();
12    }catch(Exception e){
13        System.out.println(e);
14    }
15    if(result == 1){
16        return true;
17    }else return false;
18 }
19 public String submit(){
20     if(this.save()){
21         return "index_response.xhtml";
22     }else return "index.xhtml";
23 }

```

Listing 7. Connecting to JDBC Server.

Step 5: faces-config.xml

It is not automatically created but needs to be created by us and

it is mandatory. It should be placed in WEB-INF. Every JSF application needs this file because it describes the properties of the application. The navigation rules and the default values of some variables, the bundles of messages are described by this. It is very helpful in checking the conditions which lead one page to navigate to another page. The default instances are defined.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config
3     xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-
5         instance"
6     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
7         javaee
8         http://xmlns.jcp.org/xml/ns/javaee/web-
9         facesconfig_2_2.xsd"
10    version="2.2">
11    <managed-bean>
12        <managed-bean-name>loginbean</managed-bean-name>
13        <managed-bean-class>com.tutorial.loginbean</
14            managed-bean-class>
15        <managed-bean-scope>session</managed-bean-scope>
16    </managed-bean>
17    <navigation-rule>
18        <display-name>login.xhtml</display-name>
19        <from-view-id>/login.xhtml</from-view-id>
20        <navigation-case>
21            <from-outcome>login</from-outcome>
22            <to-view-id>/welcome.xhtml</to-view-id>
23        </navigation-case>
24    </navigation-rule>
25 </faces-config>

```

Listing 8. Example of faces-config.xml file of student registration system.

Step6: Creating a Web.xml file

This is the final and most important step to execute our project. If we want to use the JSF framework in our web application the FacesServlet needs to be defined and it should be mapped in the descriptor file named as web.xml. The servlet plays a key role in processing all the requests and it plays a vital role in the front end of the application. The following code is shown in the below figure, it is an example code to create a web.xml file. Suppose we want to save the state on the server then we need to add a server as a parameter, not the client. If the client is used as a parameter, it gets hidden in the page.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema
3     -instance"
4     xmlns="http://xmlns.jcp.org/xml/ns/javaee"
5     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
6         javaee
7         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
8     version="3.1">
9    <display-name>jsftutorial</display-name>
10    <servlet>
11        <servlet-name>Faces Servlet</servlet-name>
12        <servlet-class>javax.faces.webapp.FacesServlet</
13            servlet-class>
14        <load-on-startup>1</load-on-startup>
15    </servlet>
16    <servlet-mapping>
17        <servlet-name>Faces Servlet</servlet-name>
18        <url-pattern>/faces/*</url-pattern>
19    </servlet-mapping>
20 </web-app>

```

Listing 9. Example of web.xml file of student registration system.

Step7: Output Screens

After running the application on the server, we get the final output which asks to enter the input details

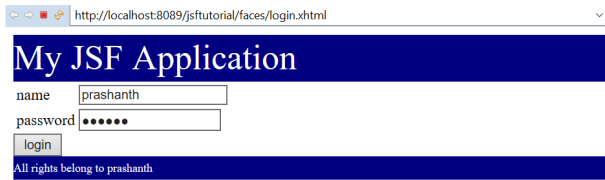


Fig. 3. Output of login.xhtml page [1].

After clicking on the login button, the following output screen is displayed. It validates the user details by using a servlet and displays the final output.



Fig. 4. Output of response.xhtml page [1].

Let's create a signup page using JSF UI components. In this page we are using input text field, radio button, select one button components.

Step 8: Creating the signup.xhtml page

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
  transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5 xmlns:ui="http://java.sun.com/jsf/facelets"
6 xmlns:h="http://java.sun.com/jsf/html"
7 xmlns:f="http://java.sun.com/jsf/core">
8
9 <ui:composition template="/basictemplate.xhtml">
10 <ui:define name="content">
11 <h:form>
12 <br/><br/>
13 First name: <h:inputText value="#{
  signup.firstname}" />
14 <br/><br/>
15 last name: <h:inputText value="#{
  signup.lastname}" />
16 <br/><br/>
17 Email id: <h:inputText value="#{
  signup.email}" />
18 <br/><br/>
19 age: <h:inputText value="#{signup.
  age}" />
20 <br/><br/>
21 Password:<h:inputSecret value="#{
  signup.password}" />
22 <br/><br/>
23
24 Country:
25 <h:selectOneMenu value="#{signup.
  country}">
26 <f:selectItems value="#{signup.
  countryOptions}" />
27 </h:selectOneMenu>
28 <br/><br/>
```

```
29 <h:commandButton value="submit"
  action="signup_response" />
30 </h:form>
31 </ui:define>
32 </ui:composition>
33 </html>
```

Listing 10. Example of signup.xhtml page

Step 9:Creation of signup-response.xhtml page

The response page is created using JSF UI components. It displays the form data to the user.

```
1 <!DOCTYPE html>
2 <html lang="en"
3 xmlns="http://www.w3.org/1999/xhtml"
4 xmlns:h="http://xmlns.jcp.org/jsf/html">
5
6 <h:head>
7 <title>user confirmation</title>
8 </h:head>
9 <h:body>
10 The user is confirmed: #{signup.firstname} #{
  signup.lastname} <br/>
11 user email id:#{signup.email} <br/>
12 user age:#{signup.age}
13 user password:#{signup.password}
14
15 <br/>
16 The user's country: #{signup.country}
17 <br/><br/>
18 <h:link value="Home Page" outcome="homepage" /
19 >
20 </h:body>
21 </html>
```

Listing 11. example of signup response page

step 10: Output Screens

After running the application on the server, we get the final output which asks user to enter the input details.

After filling the form details and by clicking the submit

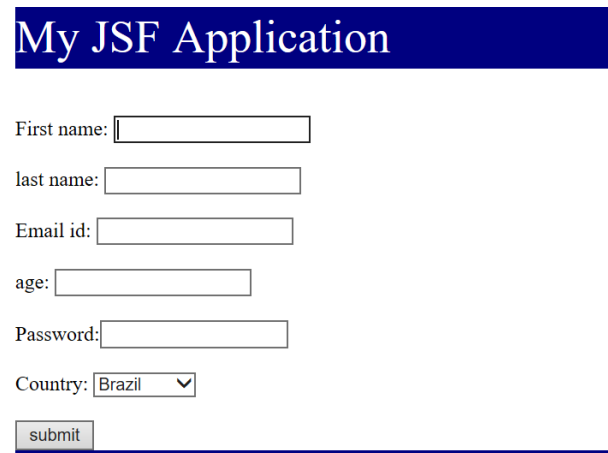



Fig. 5. output of signup page [1].

button it generates the following output.

 http://localhost:8089/jsftutorial/faces/signup.xhtml

The user is confirmed: prashanth kumar
user email id:pdaam@lakeheadu.ca
user age:23 user password:123456
The user's country: India

[Home Page](#)

Fig. 6. output of signup response page [1].

IX. CONCLUSION

The student registration form application is been developed with the help of some advanced and modern programming methods such as Java server faces and MVC architecture. The application which we created is more flexible and free from errors. The application can be modified at any according to their requirements in a simple manner. New functionalities can be added to the existing code to create similar applications. The problems that occur with other java web technologies are solved here.

X. ACKNOWLEDGEMENT

This reasearch paper is a part of the COMP5112 Research Methodology course at Computer Science, Lakehead University, Winter 2020, supervised by Professor. Dr.J.Fiaidhi.

REFERENCES

- [1] Java Server Faces tutorial point :<https://www.tutorialspoint.com/jsf/index.html>
- [2] Xu JunWu., Liang JunLing., "Developing CRM System of Web Application Based on Java Server Faces", IEEE, 2010.
- [3] Antonio García, Roberto Barchino, "Tool for Generation IMS-QTI v2.1 Files with Java Server Faces", IEEE, 2010.
- [4] Karolina Czekalska, Bartosz Sakowicz, " Hotel reservation system based on the JavaServer Faces technology", IEEE, 2008.
- [5] Mann K. D., "Java Server Faces in Action", Manning Publications Co., 2005.
- [6] Dudley B., Lehr J., Willis B., Mattingly L. "Mastering Java Server Faces", Wiley Publishing, 2004.
- [7] Java Server Faces technology specification:<http://java.sun.com/javaee/javaserverfaces/>
- [8] Wan Zhengjing, Tao Yizheng., "Web application development based on JSF technology", Modern electronic technology, 2007.