The Staff Development Committee (SPD)
of Information Technology Department
invites you to attend

Web Application Development using Jakarta Server Faces

Date : Thursday, 27th January, 2022
Time : 10:00 AM - 12:00 PM
Venue : Online via MS Teams

Resource Speaker:
Dr. Bala Dhandayuthapani V.
Lecturer- IT Department
UTAS-Shinas

University of Technology and Applied Sciences Shinas

ECLIPSE

Sultanate of Oman
University of Technology and Applied Sciences-Shinas

سلطنة عمان
جامعة التقنية والعلوم التطبيقية شناص

Web Application Development Using Jakarta Faces

Date & Time

Thursday 27 Jan 2021 - 10.00 AM to 12.00 PM

Presented by

Dr. Bala Dhandayuthapani V.

bala.veerasamy@shct.edu.om

JAKARTA EE

ECLIPSE FOUNDATION

Everything we know is always easy

Things we do not know is difficult “at first”.

Dr. Bala Dhandayuthapani V.

- **Required skills**
  - You are expected to have a good understanding of the **Java programming language**.
  - You are expected to understand the fundamentals of **web applications** and the HTTP protocol.

Software used:

- JSF 3, JDK 1.8, NetBeans 8.2 (or)
- **JSF 3.0 JDK 11, NetBeans 12.6,**
- **Glashfish** / Payara / Tomcat / WildFly / Jboss
- Jakarta EE 9
- Java DB Derby

Goals of the Session

The goal of this session is to present and demonstrate Jakarta Faces used in web app development.

- **At the end of the session, you will be able to**
  - Grasp the fundamentals of a web application framework.
  - Understand the Jakarta EE.
  - Understand the basics of Jakarta's faces technology.
  - Understand UI components and component suites.
  - Be familiar with CDI, converters, and validators.
  - Be familiar with connecting database.

Topics planned

The topics covered in this session are

1. Web Application Framework
2. Introduction to Jakarta EE
3. Introduction to Jakarta Faces Technology
4. Designing JSF pages using Facelets
5. Using Managed Beans / CDI
6. Validating and converting data
7. Working events handling and AJAX
8. Working with Hibernate

1. Web Application Framework (WAF)

- It is a software framework that is designed to support the **development of web applications**, including web services and web resources**.**
- Every framework has an **architecture.**

Example:

- Java own: JavaServer Faces by J2EE (transferred to Eclipse)
- Eclipse own: Jakarta Faces by Eclipse (new)
- Java based: Spring Web MVC, Struts (Apache)

Model View Controller (MVC)

- Many frameworks follow the MVC **architectural pattern** to separate the data model with business rules from the user interface.
- This is generally considered a good practice as it **modularizes code**, **promotes code reuse**, and **allows multiple interfaces** to be applied.

Push-based vs. pull-based

- Push-based also called "**action-based**".
  - these frameworks use actions that do the required processing, and then "push" the data to the view layer to render the results.
- Pull-based also called "**component-based**".
  - these frameworks start with the view layer, which can then "pull" results from multiple controllers as needed.

Jakarta faces - Architecture

- Jakarta faces technology is a web application framework for
  - developing, building server-side user interface components
  - and using them in a web application.
- Jakarta faces technology is based on the **Model View Controller (MVC)** architecture for separating logic from presentation.
  - **Model** - Connector for *view and controller*.
  - **View** - Shows User Interface. e.g. **web design**
  - **Controller** - Handles processing of an application, used to process **user actions**.

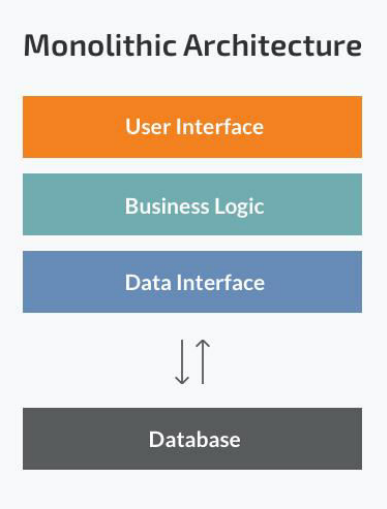
2. Introduction to Jakarta EE

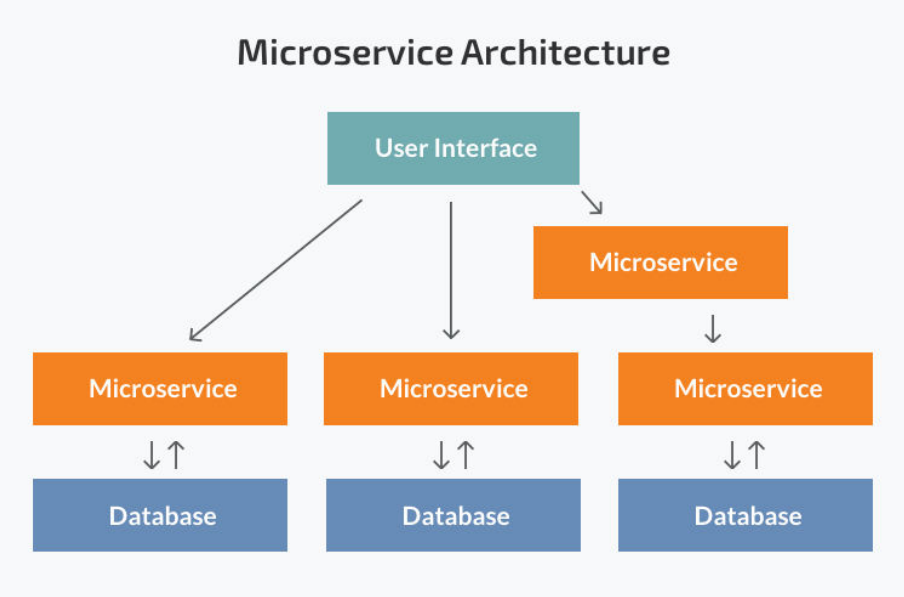
What is Jakarta EE?

- Jakarta EE is the future for **cloud-native**, light-weight, and traditional enterprise Java applications
- **Java EE** technologies contributed by **Oracle**
  - are being used to create the new **Jakarta EE** platform
- The Eclipse Foundation is the home of **Cloud Native Java** **open innovation**

What is Cloud Native?

- **Cloud Native** refers to an application that's **built for the cloud.**
- Cloud native microservices make best use of the cloud
- Cloud native computing is an approach in software development
  - utilizes cloud computing to "build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds".





Why Cloud Native Microservices?

- Companies are working towards cloud native microservices
  - in order to save cost and make the best use of cloud resources.
- When a microservice runs in the cloud, it only costs you money when it is running.
- Depending on the load, it can scale up or down. (**Pay as You Go model**.)
- **For existing monolith applications**, it is time to think about how to deploy them in the Cloud.
  - You can either containerise them or break them into microservices.
- **For newly-written applications**, the best choice is to build green-field cloud native microservices.

Transition from Java EE to Jakarta EE

Version	Date
J2EE 1.2	December 1999
J2EE 1.3	September 2001
J2EE 1.4	November 2003
Java EE 5	May 2006
Java EE 6	December 2009
Java EE 7	April 2013
Java EE 8	August 2017
Jakarta EE	February 2018*

JDK1.0	January 23, 1996	Sun Microsystems, Inc
JavaSE 7	January 27, 2010	Oracle Corporation
JavaSE 17	September 14, 2021	Oracle Corporation

Maven Coordinates	Certified as	Javadoc Terms	Built from	Java Package	TCK Level	API Level
javax.faces:javax.faces-api:2.3	Java EE 8	Java EE terms	github/javaee	javax.faces	JDK 8	JDK 8
jakarta.faces:jakarta.faces-api:2.3.1	Java EE 8	Java EE terms	github/eclipse-ee4j	javax.faces	JDK 8	JDK 8
jakarta.faces:jakarta.faces-api:2.3.2	Jakarta EE 8	Jakarta terms	github/eclipse-ee4j	javax.faces	JDK 8	JDK 8
jakarta.faces:jakarta.faces-api:3.0	Jakarta EE 9	Jakarta terms	github/eclipse-ee4j	jakarta.faces	JDK 11	JDK 8

Cloud Native Technologies

- **MicroProfile** is a **collection of specifications** designed to help developers build Enterprise Java cloud-native microservices.
  - Eclipse *MicroProfile* and Spring *Microservices*

Eclipse MicroProfile:

- Address microservice architectures for Jakarta EE and non-Jakarta EE technologies.
  - **Jakarta EE** (https://jakarta.ee)
  - **non-Jakarta EE technologies** (https://microprofile.io)
- Developers can ***mix and match*** Jakarta EE and MicroProfile APIs in the same application.
- This approach allows enterprises to built applications during the **pre-cloud, pre-container era** to take advantage of more efficient microservices.

Developers can create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, **without requiring any changes in service code.**

Docker : (https://www.docker.com)

It is a **containers**, give developers to

- Create different application environments and test different scenarios
- Port applications from one cloud or environment to another.

Kubernetes : (https://kubernetes.io)

- It is an **open source container orchestration system**
- for automating application deployment, scaling, and management.

Istio : (https://istio.io)

- It is an **open source service mesh** for connecting, monitoring, and securing microservices.

Web Applications

A web application is a dynamic extension of a web or application server.

Presentation-oriented

- An Interactive web pages such as HTML, XHTML, XML
  - Dynamic content in response to requests.
- Jakarta Faces Technology
- Jakarta Servlet Technology

Service-oriented

- Implements the **endpoint** (clients) of a web service.
  - Presentation-oriented applications are often clients
- **Web Services** with Jakarta XML Web Services
- **RESTful Web Services** with Jakarta REST

Jakarta EE Platform 10
March 31, 2022
Jakarta Server Faces 4.0

Jakarta EE 9 (December 8, 2020)

- Jakarta Servlet 5.0
- Jakarta Server Pages 3.0
- Jakarta Expression Language 4.0
- Jakarta Debugging Support for Other Languages 2.0
- Jakarta Standard Tag Library 2.0
- Jakarta Server Faces 3.0
- Jakarta RESTful Web Services 3.0
- Jakarta WebSocket 2.0
- Jakarta JSON Processing 2.0
- Jakarta JSON Binding 2.0
- Jakarta Annotations 2.0
- Jakarta Enterprise Beans 4.0 Lite
- Jakarta Transactions 2.0
- Jakarta Persistence 3.0
- Jakarta Bean Validation 3.0
- Jakarta Managed Beans 2.0
- Jakarta Interceptors 2.0
- Jakarta Contexts and Dependency Injection 3.0
- Jakarta Dependency Injection 2.0
- Jakarta Security 2.0
- Jakarta Authentication 2.0

3. Introduction to Jakarta Faces Technology

- Jakarta Faces technology is a **server-side component framework** for building Java technology based web applications.

Jakarta Faces technology consists of the following:

- **An API** for representing components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features
- **Tag libraries** for adding components to web pages and for connecting components to server-side objects

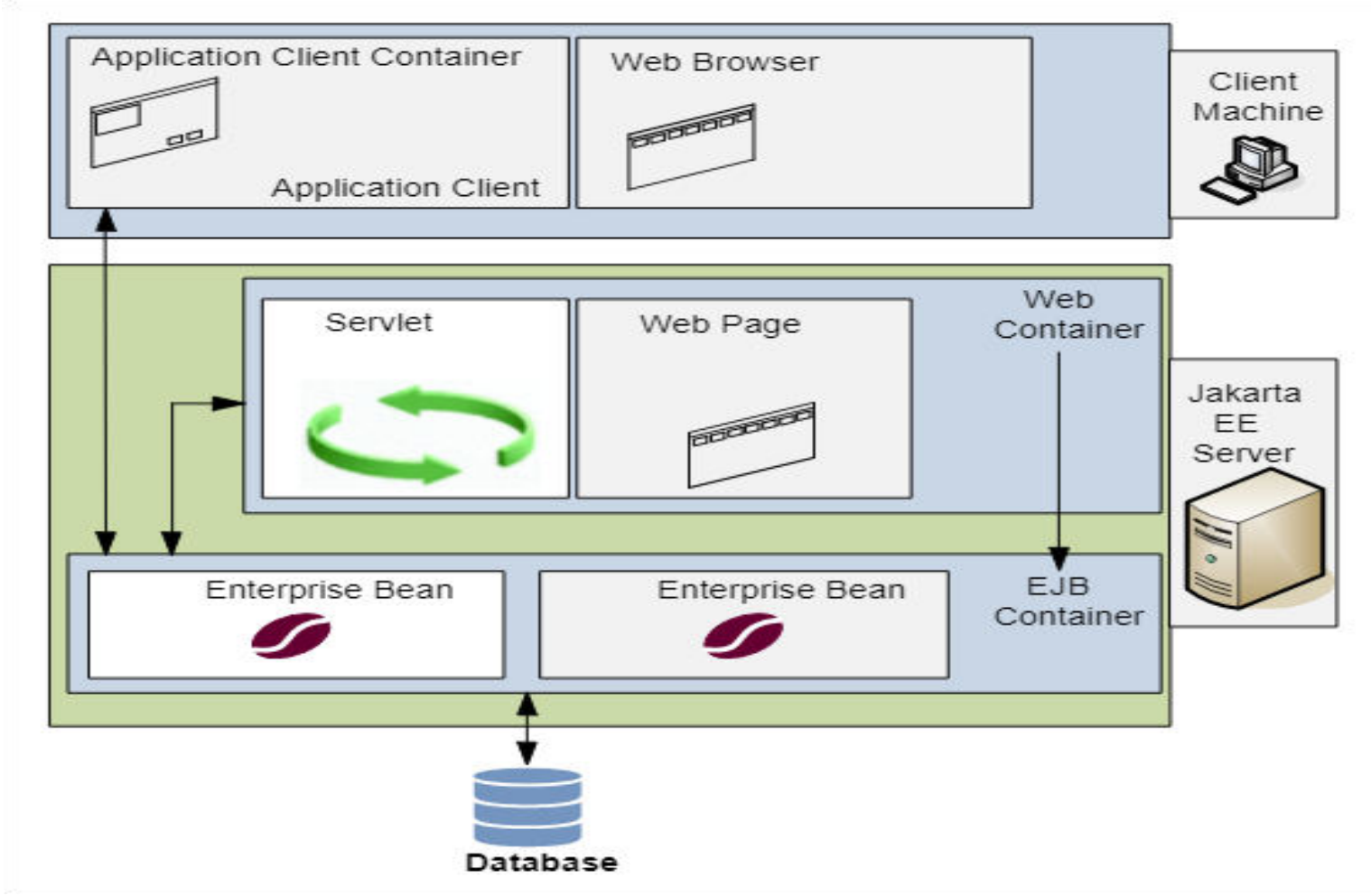
What is a Jakarta Faces Application?

- A set of web pages in which components are laid out.
- A set of tags to add components to the web page.
- A set of managed beans, which are lightweight, container-managed objects (POJOs). It serve as backing beans, which define properties and functions for UI components on a page.
- A web deployment descriptor (web.xml file)
- Optional
  - **faces-config.xml file**, which can be used to define page navigation rules and configure beans and other custom objects, such as custom components.
  - **custom objects**, which can include custom components, validators, converters, or listeners, created by the application developer.
  - **custom tags**, for representing custom objects on the page.

Jakarta Faces Technology Benefits

- Separation between behaviour or login and presentation for web applications.
  - development team to focus on a single piece of the development process
- It can map HTTP requests to component-specific event handling and manage components as stateful objects on the server
- It is to leverage familiar component and web-tier concepts without limiting you to a particular scripting technology or markup language.

Jakarta EE Server and Containers



Jakarta faces - Life Cycle

- The lifecycle of a Jakarta Faces application
  - **begins** when the **client makes an HTTP request** for a page
  - **ends** when the server responds with the page.

The lifecycle is divided into **two main phases**:

1. Execute Phase
2. Render Phase

Facelets

It is a lightweight **page declaration language** which is used to build **Jakarta Faces views** using HTML style.

- It uses **XHTML** that supports **Facelets tag** libraries for creating web pages.
- It supports the Expression Language (**EL**).

Advantages:

- Faster compilation time, High-performance rendering.
- It validates expression language at compile-time.
- It supports code reusability through templating and composite components.
- It provides functional extensibility of components and other server-side objects through customization.

Tag Library	URI	Prefix	Example	Contents
Jakarta Faces HTML Tag Library	http://xmlns.jcp.org/jsf/html	h:	h:head h:body h:outputText h:inputText	JSF component tags for all UIComponent objects
Jakarta Faces Core Tag Library	http://xmlns.jcp.org/jsf/core	f:	f:convertNumber f:attribute	Tags for Validation & conversion internationalization overall application development.
Composite Component Tag Library	http://xmlns.jcp.org/jsf/composite	cc:	cc:interface	Tags to support composite components
Jakarta Faces Facelets Tag Library	http://xmlns.jcp.org/jsf/facelets	ui:	ui:component ui:insert	Tags for templating

The package is jakarta.faces

4. Designing Jakarta Faces pages using Facelets

- Rich set of classes for specifying the **state and behaviour** of UI components.
  - A **rendering model** that defines how to render the components in various ways.
  - A **conversion model** that defines how to register data converters onto a component.
  - A **validation model** that defines how to register validators onto a component.
  - An **event and listener model** that defines how to handle component events.

JSF View

- The <h:form> tag represents an input form.
- It includes child UI components
- It contain data which is either presented to the user or submitted with the form.

```
<f:view>
    <h:form>
            <!-- form elements -->
    </h:form>
</f:view>
```

Input UI Components

Tag	Functions
h:inputText	It allows a user to input a **single line string.**
h:inputTextarea	It allows a user to enter a **multiline string.**
h:inputHidden	It allows a page author to include a **hidden variable** in a page.
h:inputSecret	It allows a user to input a string without the actual string appearing in the field. It is for **password field**.
h:inputFile	It allows a user to **upload a file**.

Example:

```
<h:inputText
    id="username"
    value="#{user.name}"
    label="username"
    maxlength="10"
    size="15"
    alt="username"
    readonly="false"
    required="true"
    requiredMessage="Username is required“
    validatorMessage=“Username allowed with rules…"
    style="color:red" accesskey="q">

</h:inputText>
```

Output UI Components

Tag	Functions
h:outputLabel	It displays a nested component as a **label for a specified input field**.
h:outputText	It **displays a line of text**.
h:outputStylesheet	It renders a **<style>** element.
h:message	It displays a localized **message**.
h:messages	It displays localized **messages**.
h:graphicImage	It displays an **image**.

Button/Link UI Components

Tag	Functions
h:commandButton	It **submits a form** to the application. **Action based**.
h:commandLink	It **links to another page** or location on a page. **Action based.**
h:outputLink	It **links to another page** or location on a page **without** generating an **action event**. (Internal/External link)

Select Input UI Components

Tag	Functions
h:selectManyListbox	**select multiple items** from a set of items all displayed at once.
h:selectOneListbox	**select one item** from a set of items all displayed at once.
h:selectManyMenu	**select multiple items** from a set of items.
h:selectOneMenu	**select one item** from a set of items.
h:selectManyCheckbox	**select multiple values.**
h:selectBooleanCheckbox	boolean choice.
h:selectOneRadio	**select one item** from a set of items.

Panel UI Components

Tag	Functions
h:panelGrid	It displays UI components in a **table**.
h:panelGroup	It **groups** a set of UI components under one parent.

Data Table UI Components

Tag	Functions
h:dataTable	It represents a **data wrapper.**
h:column	It represents a **column of data** in a data component.

5. Contexts and Dependency Injection (CDI)

- Mainly used for programmatic **annotations** rather than configuration.

Common functions:

- validating a component's data,
- handling an event,
- processing data,
- navigations.

- **Using objects from expression language (EL)**
- **@Named annotation** enables interaction with **CDI managed beans** using the bean name starting with a lowercase letter.
  - **Example: @Named("mybean")**
- It includes one or more Managed beans (POJOs) each of which can be associated with the components/fields used in a particular page (XHTML).

Dependency injection

- objects accessed across **various classes** within an application.
  - For instance, if an object is populated with values, it may make sense to use that object's values in their current state from within another class for some process.
- Assigning scope to contextual objects

Assigning scope to contextual objects

Few scopes for a CDI bean class:

- **Application (@ApplicationScoped):** persists across **all user's interactions** with a web application.
- **Session (@SessionScoped):** persists across **multiple HTTP requests** in a web application.
- **View (@ViewScoped):** persists during a **user's interaction with a single page** (view)
- **Request (@RequestScoped):** persists during a **single HTTP request** in a web application.

6. Validating and Converting Data

Convertors:

- The Jakarta Faces provides a set of Converters.
- You can use that to convert component data.
- The **jakarta.faces.convert** package contains all the standard converters.
- You can also access these converters by **converter ID.**

Convertor Double

```
<f:converter converterId="jakarta.faces.Double"/>
```

Convertor Integer

```
<f:converter converterId="jakarta.faces.Integer"/>
```

Percentage

```
<f:convertNumber type = "percent" />
```

Currency

```
<f:convertNumber currencySymbol = "$" type = "currency" />
```

Pattern

```
<f:convertNumber pattern = "#000.000" />
```

Min Fraction

```
<f:convertNumber minFractionDigits = "2" />
```

Date Convertor

```
<f:convertDateTime pattern = "dd-mm-yyyy" />
```

JSF Validation

JavaServer Faces technology provides a **set of standard classes and associated tags** that you can use to validate elements data.

<f:validateRequired/>

<f:validate**Length** minimum = "5" maximum = "8" />

<f:validate**LongRange** minimum = "1" maximum = "100" />

<f:validate**DoubleRange** minimum = "1" maximum = "1000" />

<f:validate**Regex** pattern = "((?=.*[a-z,0-9]).{6,})"/>

<f:validateBean/>

JSF <h:message> Tag

- It is used to display a **single message** for a particular component. You can display your custom message by passing id of that component into the for attribute.

```
<h:inputText id="t1" value="#{user.name}"/>
<h:message for="t1" style="color: red"/>
```

JSF <h:messages> Tag

- It is used to **displays all messages** that were stored in the faces context during the course of the JSF life cycle.

```
<h:inputText id="name-id" value="#{user.name}"/>
<h:inputText id="mobile-id" value="#{user.mobile}"/>
<h:messages style="color: red"></h:messages>
```

7. Working Events Handling and AJAX

Event Handling

- When a user clicks a JSF button or link or changes any value in the text field, JSF UI component **fires an event**, which will be handled by the application code.
- To handle such an event, an event handler is to be registered in the application code or **managed bean.**
- There are **three types of events**
  - actionListener (button pressed…)
  - valueChangeListener (list value changed..)
  - Application Events (lifecycle events)

Example

```
public void valueChanged(ValueChangeEvent e){
        depatName=e.getNewValue().toString();
    }

public void actionDone(ActionEvent e){
        msg="Welcome to Webinar";
    }
```

JSF - Ajax

- AJAX stands for **Asynchronous JavaScript and Xml.**
  - is a technique to use HTTPXMLObject of JavaScript to **send data to the server and receive data from the server asynchronously**.
  - JavaScript code exchanges data with the server, updates parts of the **web page without reloading the whole page**.
- JSF provides excellent support for making ajax call. It provides **f:ajax tag** to handle ajax calls.

JSF Tag

<f:ajax **execute** = "input-compo-name" **render** = "output-compo-name" />

Example:

```
<h:form>
        <h:inputText id = "name" value = "#{u.name}"/>
                <h:commandButton value = "Show Name“  type=“submit()”>
                        <f:ajax execute = "name" render = "msg" />
                </h:commandButton>
        <h:outputText id = "msg" value = "#{u.name}"/>
</h:form>
```

Example using AJAX without commandButton

```
<h:form>

        <h:inputText id = "name" value = "#{u.name}“/>

         <f:ajax event="keyup" execute = "name" render = "msg" />

       <h:outputText id = "msg" value = "#{u.name}"/>

</h:form>
```

8. Connecting with Database

Jakarta Faces application can be used to connect database application in various ways, they are

- **Java Database Connectivity (JDBC) API**
- **Java Data Objects (JDO)**
- **Java Persistence API (JPA)**
- **Hibernate**

```
//connect the Database
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    connection =
        DriverManager.getConnection("jdbc:derby://localhost:1527/MyDB","bala","bala");

//Create a record
    PreparedStatement stmt =
     connection.prepareStatement("insert into contact(cid,name,phone) values(?,?,?)");
        stmt.setInt(1, id);
        stmt.setString(2, name);
        stmt.setInt(3, phone);
        result = stmt.executeUpdate();
        connection.close();

//Delete a  Record
    PreparedStatement stmt = connection.prepareStatement("delete from contact where cid = "+id);
        stmt.executeUpdate();
```

//Edit/select a Record

```
 Statement stmt=connect
ion.createStatement();
        ResultSet rs=stmt.executeQuery("select * from contact where cid = "+id);
        rs.next();
        user = new UserData();
        user.setId(rs.getInt("cid"));
        user.setName(rs.getString("name"));
        user.setPhone(rs.getInt("phone"));
```

//Update a Record

```
PreparedStatement stmt=connection.prepareStatement("update contact set name=?,phone=? where
cid=?");
        stmt.setString(1,u.getName());
        stmt.setInt(6,u.getPhone());
        stmt.executeUpdate();
        connection.close();
```

Summary

In this session, you could understand

- the fundamentals of a web application framework
- the Jakarta EE
- the basics of Jakarta's faces technology
- UI components and component suites.
- CDI, converters, and validators
- how to connect database.

This will assist your professional and personal growth.

References

Tutorial Jakarta faces technology:

- https://eclipse-ee4j.github.io/jakartaee-tutorial
- https://jakarta.ee/specifications/faces/3.0/jakarta-faces-3.0.html
- https://jakartablogs.ee

Jakarta EE download:

- https://jakarta.ee/compatibility/download

NetBeans IDE

- http://netbeans.org

Discussions…