JPA, Entity Beans and UI

Binding DB Data To UI

- Next we need to design the UI. We need to consider i.e. following things
  - What data user wants to see in our application?
  - How we display that information?
  - What kind of operations the user needs to make to achieve some goal?

Binding DB Data To UI

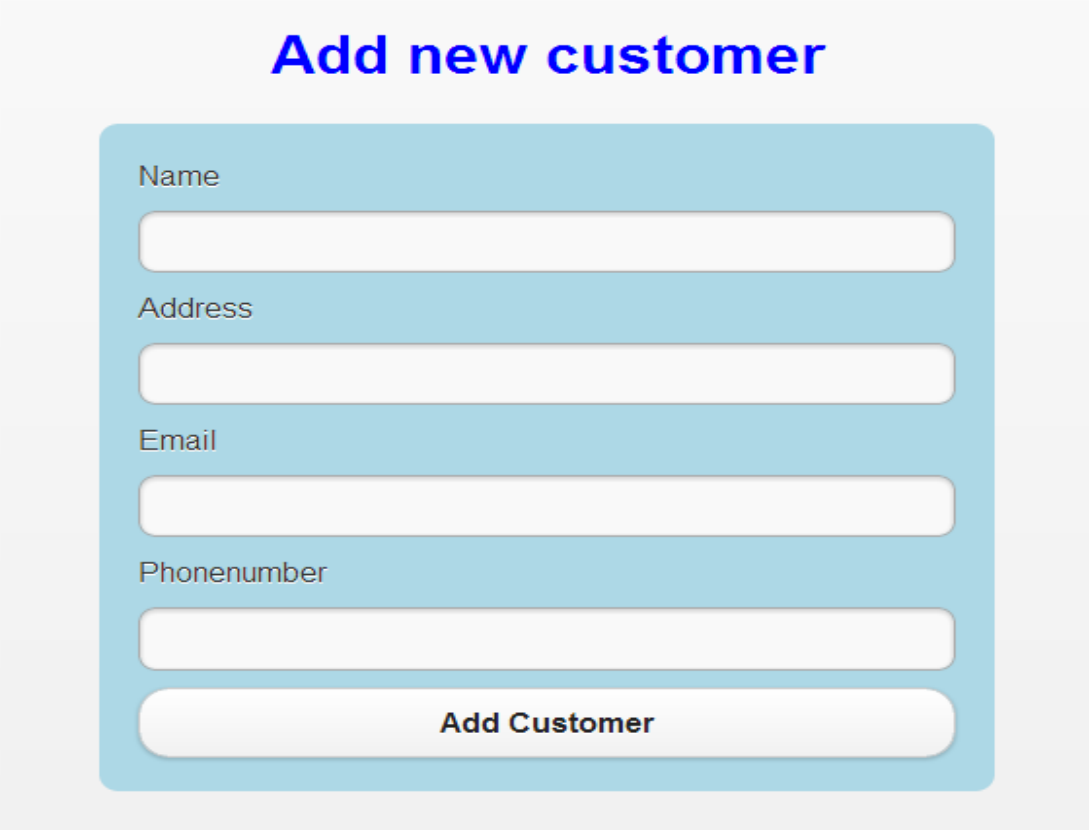
- Lets start from the customer table. User should be able to make next operations in our example:
  - Add new customer
  - View all customers
  - Find customer by name
- Let's make one operation at time. We could make own page for all of these operations, but for saving time we make these all in one page.

Add new customer

- To add new customer user have to give next information’s:name, address, email and phonenumber. So we need a user interface containing textfield for each information and button that will save the information to database when user hits it.
- There should be also all kind of data validation, like checking that all required information is given before saving the data, but this I will let you to handle.

Add new customer

- The goal for our UI is next...



Add new customer

- The JSF code head part is...

```
<h:head>
    <title>Customers</title>
    <link type="text/css"
          rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.css"/>
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js"></script>
    <h:outputStylesheet library="css" name="my_style.css"></h:outputStylesheet>
</h:head>
```

Add new customer

- And body part...

```
<h:body>
    <h1>Add new customer</h1>
    <h:form>
        <fieldset>
            <h:outputLabel for="username" value="Name"></h:outputLabel>
            <h:inputText id="username"></h:inputText>
            <h:outputLabel for="address" value="Address"></h:outputLabel>
            <h:inputText id="address"></h:inputText>
            <h:outputLabel for="email" value="Email"></h:outputLabel>
            <h:inputText id="email"></h:inputText>
            <h:outputLabel for="phone" value="Phonenumber"></h:outputLabel>
            <h:inputText id="phone"></h:inputText>
            <h:commandButton type="submit" value="Add Customer"></h:commandButton>
        </fieldset>
    </h:form>
</h:body>
```

Add new customer

- And my_style.css

```
html,body{
    margin: 0;
    padding: 0;
    min-width: 100%;
    width: 100%;
    height: 100%;
}

form{
    margin: auto;
    width: 480px;
}

fieldset{
    background-color: lightblue;
}
```

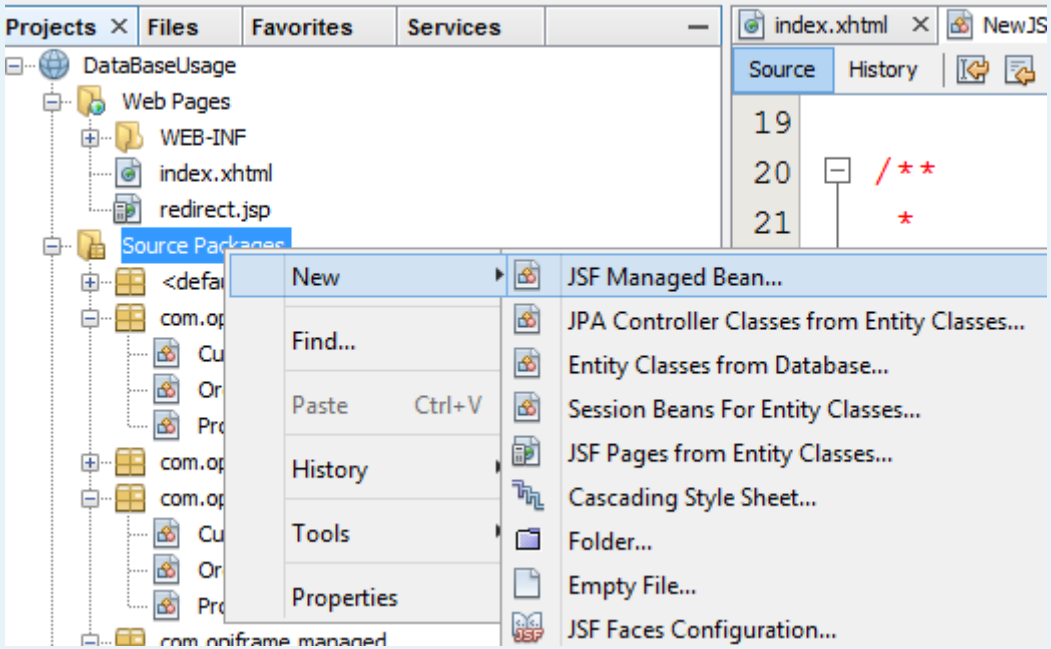
```
.ui-mobile fieldset{
    margin: 10px;
    padding: 10px;
    border-radius: 10px;
}

div.ui-input-text{
    margin: 10px;
}
label.ui-input-text{
    margin: 10px;
}

.ui-btn{
    margin: 10px;
}
h1{
    text-align: center;
    color: blue;
}
```

Storing the user in DB

- First of all we need an JSF Managed Bean
- The managed bean will contain a function that then will append a new user into database.
- We will bind this function to our Add Customer button action property.

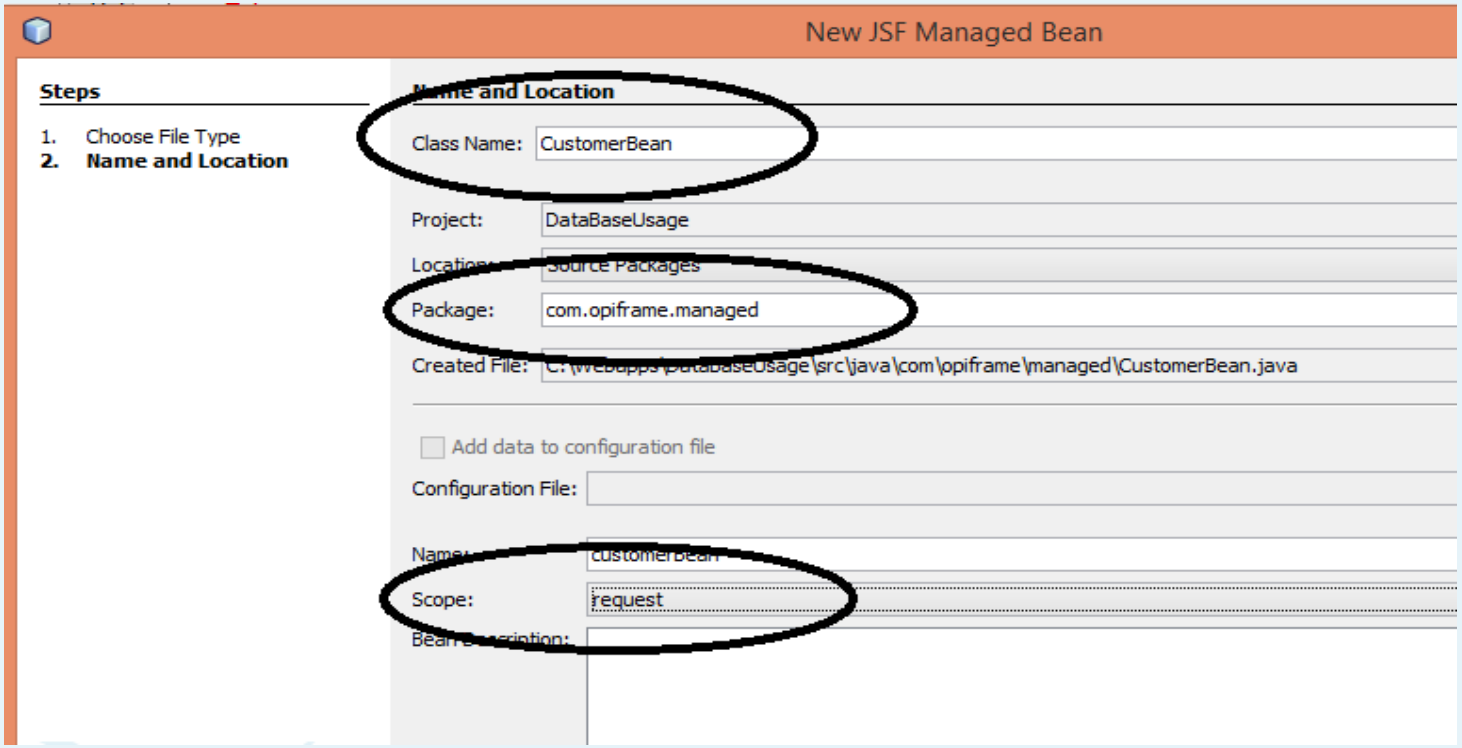
Storing the user in DB

- To append Managed Bean to your project, right click over Source Packages, select New->JSF Managed bean



Storing the user in DB

- Give name, package and scope for the bean and press Finsih button



Storing the user in DB

- Next we need to define Injections. If you see the CustomerJpaController class you notice that it constructor takes two arguments...

```
public CustomersJpaController(UserTransaction utx, EntityManagerFactory emf) {
    this.utx = utx;
    this.emf = emf;
}
```

Storing the user in DB

- So when we create a new CustomerJpaController we need to pass two object for it **UserTransaction** and **EntityManagerFactory**.
- Both of these classes are controlled by GlassFish web container, so there is an instance of these already present
- So we can inject these instances in our managed bean.

Storing the user in DB

- Injection is done with annotations. One thing to remember! Injections are done AFTER object construction which means you are not able to make REFERENCE to injected objects in managed bean constructor (if you need to do so you can annotate constructor with @PostCosntruct).
-

Storing the user in DB

```
@ManagedBean(name = "newJSFManagedBean")
@RequestScoped
public class NewJSFManagedBean implements Serializable{

    //Injections
    @PersistenceContext
    EntityManager emf;
    @Resource
    UserTransaction utx;

    public NewJSFManagedBean() {
    }
}
```

Storing the user in DB

- Next we need to read the values from the incoming form and save that information to database.
- To get a reference to client request you need to use FacesContext object.
- Next I append a new function to managed bean called saveUser. This function reads the values from incoming form and uses CustomerJpaController for storing the data to database.

Storing the user in DB

```
public void saveUser(){
    HttpServletRequest request =
            (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest();
    Map<String,String[]> map = request.getParameterMap();

    Customers cust = new Customers();
    cust.setName(map.get("j_idt7:username")[0]);
    cust.setAddress(map.get("j_idt7:address")[0]);
    cust.setEmail(map.get("j_idt7:email")[0]);
    cust.setPhone(map.get("j_idt7:phone")[0]);

    CustomersJpaController jpaCon = new CustomersJpaController(utx,emf.getEntityManagerFactory());
    try {
        jpaCon.create(cust);
    } catch (RollbackFailureException ex) {
        Logger.getLogger(NewJSFManagedBean.class.getName()).log(Level.SEVERE, null, ex);
    } catch (Exception ex) {
        Logger.getLogger(NewJSFManagedBean.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Storing the user in DB

- And finally we need to bind the addUser function to our command button action property.

```
<h:outputLabel for="phone" value="Phonenumber"></h:outputLabel>
<h:inputText id="phone"></h:inputText>
<h:commandButton type="submit" value="Add Customer" action="#{newJSFManagedBean.saveUser()}"></h:commandButton>
ieldset>
m>
```

Show the list of customers

- Next thing we want to do is show the list of all customers in our system.
- This can be done in several ways, this time we use <ui:repeat> to build a list of links of our customers in database...
- What I need to do is to append a new function to our managed bean that return a List of Customer objects.

Show the list of customers

- Managed Bean new function

```
public List<Customers> getCustomers(){
    CustomersJpaController jpaCon = new CustomersJpaController(utx,emf.getEntityManagerFactory());
    return jpaCon.findCustomersEntities();
}
```

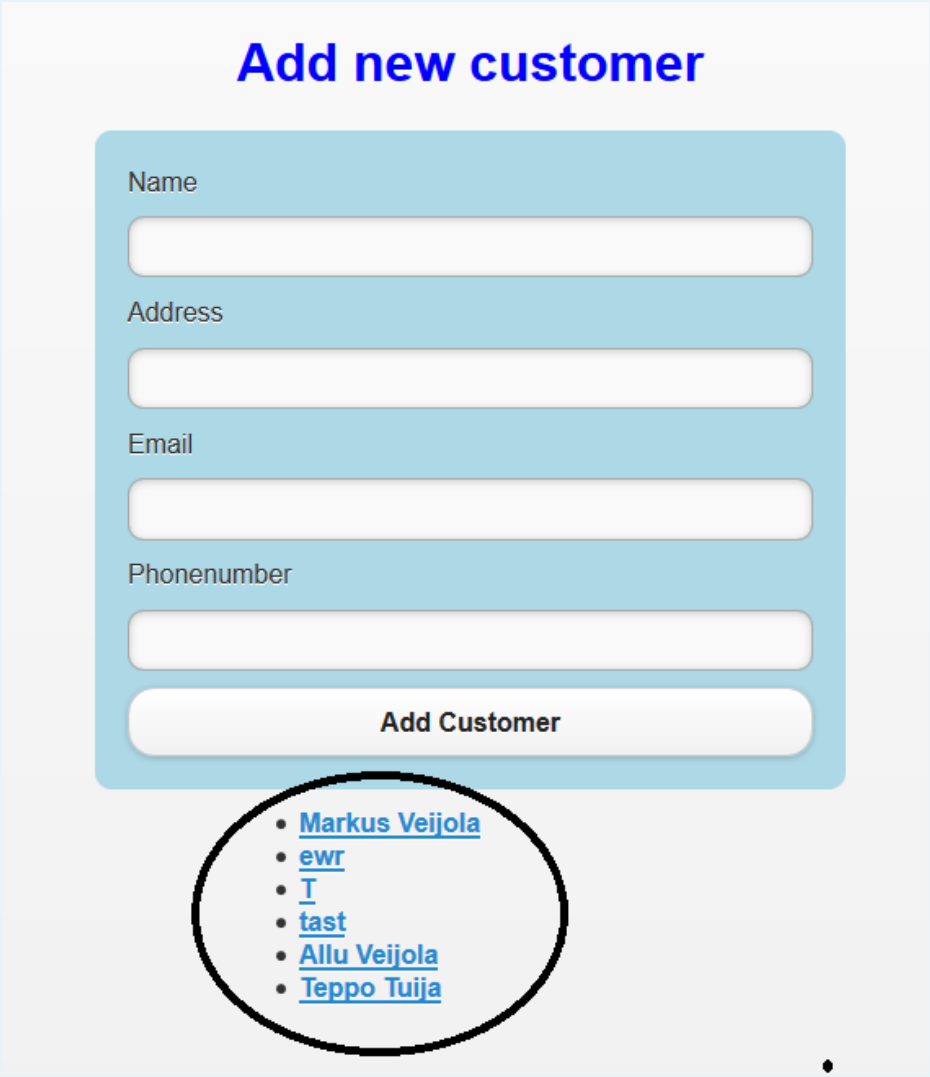
Show the list of customers

- Index.xhtml

```
<ul id='list' style='margin: auto;width: 250px;'>
    <ui:repeat value="#{newJSFManagedBean.customers}" var="cust">
        <li>
            <h:link value="#{cust.name}">
                <f:param name="id" value="#{cust.customerId}"></f:param>
            </h:link>
        </li>
    </ui:repeat>
</ul>
```

Show the list of customers

- Result



Handling h:link onclick action

- Next we want to do something if user click one name in our list.
- The something in this case is to retrieve all orders for that user.
- What I want to do is to read the customerId from the link attributes in my managed bean and get all orders with that id.
- Next example just gets the id of customer redirects the request to order.xhtml and passes the id as url argument

Handling h:link onclick action

```
public void nameLinkClicked(){
    ExternalContext context = FacesContext.getCurrentInstance().getExternalContext();
    String index = context.getRequestParameterMap().get("id");
    HttpServletResponse response = (HttpServletResponse)context.getResponse();
    try {
        response.sendRedirect("order.xhtml?id=" + index);
    } catch (IOException ex) {
        Logger.getLogger(NewJSFManagedBean.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Handling h:link onclick action

- The UI Code

```
<h:form>
<ul id='list' style='margin: auto;width: 250px;'>
    <ui:repeat value="#{newJSFManagedBean.customers}" var="cust">
        <li>
            <h:commandLink value="#{cust.name}" action="#{newJSFManagedBean.nameLinkClicked()}">
                <f:param name="id" value="#{cust.customerId}"></f:param>
            </h:commandLink>
        </li>
    </ui:repeat>
</ul>
</h:form>
```

What About the Rest?

- The rest of the application development is intended to be done by you.
- What is missing is handling the orders for user and of course adding the products to order.