**ERP Simulator – Design Documentation of 4th phase**

**Introduction:**

ERP Simulator to integrate all the functional departments in an enterprise. A user first has to sign-up himself as a consumer. Once registered, a consumer can log-in to place an order from the products provided by the organization. A consumer can create, modify or cancel an order update the order status. Producer views the orders and update order status and can also view and update the inventory. Admin can view the producers list or consumers list and the orders based on consumer name or producer name.

**Overall Design:**

The following were the objects implemented in our application:

1. Product

* ProductEntity : Used to represent the product table.
* ProductService : Contains services that modify the product data.
* ProductModel : Used to represent product data displayed on the website and for passing data to different pages.

1. Order

* OrderEntity : Used to represent the order table.
* OrderService : Contains services that modify the order data.
* OrderModel : Used to represent order data displayed on the website and for passing data to different pages.

1. Inventory

* InventoryEntity : Used to represent the inventory table.
* InventoryService : Contains services that modify the inventory data.
* InventoryModel : Used to represent inventory data displayed on the website and for passing data to different pages.

**Front End Design**

In the front end of the application we had XHTML pages supported by their backing beans. Each web page of the application had a separate XHTML page along with its corresponding bean to store the information.

For data associated pages (user, producer, inventory), pages that load information from /the database, we decided to populate a model with data corresponding to the required bean. Thus using the models we were able to access and display only values that were necessary for a page instead of querying all the entities. Also, we used the managed-property feature to pass data along the managed beans. Using this, we were able to persist the changes occurring in a particular bean to the entire application.

**Back End Design**

We designed the database tables for the application based on the design decisions made in parts 1 and 2 of the project. All the tables’ relationships with other tables by design of the application.

The services relating to the entities implement the CRUD functions required for application on each entity in the database.

The straightforward approach to passing data between the backend and the front end (data passing the beans and services) would have been to pass the resulting bean/beans to the requesting bean.

**ENTITY BEANS**

They map the service classes to the database.

We created different entity beans for the three main tables of ERP Simulator which are USERS, ORDERS and INVENTORY named HelloServiceEntity, OrderEntity and InventoryEntity respectively. This makes sure that the operations which would be implemented on the three tables in our database are compartmentalized.

The Entity beans have @Entity tags which declares the class as an entity (i.e. a persistent POJO class). They also have the table name tags which specify the database table to which it is mapped to. E.g. @Table(name="INVENTORY”).

Along with this the various column name attributes are set. They specify a column name in the database. E.g. @Column(name="PRODID").

There will be getter and setters functions for each variable corresponding to each column in the database to which it is mapped.

@Id sets the identifier property of this entity i.e. primary key. E.g. userid is the Identifier in the HelloServiceEntity class which maps to USERID, the Primary Key in the USERS Table.

**BACKING BEANS**

1. AdminBean – uses the service class HelloService to give functionalities to the admin of the portal relating to modifying the USERS table. Admin can list the consumers or the producers and can block a consumer. It uses an object helloService of the class HelloService to call the functions listpro(), listcon() and delconsumer().
2. OrderBean – uses the service class OrderService to give functionalities to a consumer, producer and admin relating to modifying the ORDERS table. A consumer can place an order, check the status of his order, update his order or cancel his order. It uses an object orderservice of the class OrderService to call the functions oset(),checkstatus(),oupdate() and cancel(). A producer can view orders and update the status of order. It uses an object orderservice of the class OrderService to call the functions viewporder(), ostatusupdate(). An admin can view the orders based on a producer name or a consumer name . It uses an object orderservice of the class OrderService to call the functions conorderdisplay() and proorderdisplay().
3. ProducerBean – uses the service class ProducerService to give functionalities to a producer relating to modifying the INVENTORY table. A producer can view and update contents of the inventory. It uses an object producerservice to call the functions viewpinventory and updateinventory().
4. RegisterBean – uses the service class HelloService for registering a new user and for logging in an existing user. It uses an object helloService of the class HelloService to call the functions userName() and loginUser().

**SERVICE CLASSES (SESSION BEANS)**

Methods called in the backing beans would have their implementation in the service classes. The methods are implemented and then an object of entity beans type is created and passed to the entity beans where it gets persisted into the database.

1. HelloService – links the HelloServiceEntity (which maps to USERS table) to the RegisterBean and AdminBean.
2. OrderService – links the OrderEntity (which maps to ORDERS table) to the OrderBean.
3. ProducerService – links the InventoryEntity (which maps to INVENTORY table) to the ProducerBean.

**FLOW**

A consumer first need to register in the ERPS portal. The first view is home.xtml where you can log in as a consumer, producer or admin, or register as a new consumer. When the user clicks to register as a new consumer, newconsumer.xhtml is displayed. The user enters his preferred username and password and clicks on ‘Register’. The auto generated userid along with the specified username and password and usertype = ‘Consumer’ is entered into the USERS table in the database.

Now after log-in, consumer can either of 1. Place an order 2. Check an order status 3. View order details 4. Update order 5. Cancel an order.

Suppose he clicks on ‘Place an order’. This loads the placeorder.xhtml page. Here the user selects the product name from a drop down list and producer name from another drop down list and also enters the quantity of the products and order date. When he clicks ‘Submit ’ flow goes to the orderSet() function of OrderBean class. orderSet() function uses an object orderservice of the OrderService class and uses it to call the function oset() and passes as parameters the username of the consumer, product name, quantity, order date and producer name.

newEntity is an object of the OrderEntity class. From the arguments received from the backing bean (OrderBean) it sets the values in the OrderEntity class through the object newEntity. OrderEntity class then connects to the h2 database and inserts a row in the ORDERS table. Once the order is entered in the database, the orderdetails along with the order id which is auto generated is retrieved and passed as a return object orderdetails of type two dimensional list. This is received by the orderset() function of the OrderBean class. If it is found to be empty it will be directed to placeorderfail.xhtml page where failure message is displayed. Else it is directed to placedorder.html where the order details are displayed to the consumer to confirm.

The main linkage happens at faces-config.xml where the definition of the managed beans would be present and the h2 database link would be completed through the datasource entry at standalone.xml of Jboss wildfly.

**Lifetime of each object:**

**\* Admin**: An object of admin class is created as soon as the admin logs into the ERPS System. This object is destroyed only when the admin logs out from the system. During the time the admin is active , the admin object can view list of consumers, producers and orders and even can delete a consumer.

**\* Consumer**: An object of consumer class is created as soon as a consumer logs into the ERPS System. This object is destroyed only when a consumer logs out from the system. During the time the consumer is active , the consumer object can place an order, check an existing order/orders, modify the existing order and can even cancel an existing order.

**\* Producer**: An object of producer class is created as soon as a producer logs into the ERPS System. This object is destroyed only when a producer logs out from the system. During the time the producer is active , the producer object can manage the inventory, view the orders placed against him and can update the order status.

**Database Schema:**

**Users\_Orders:** Many to many relation between Users and Orders

**User\_Inventory:** Many to many relation between Users and Inventory

**Orders\_Inventory:** Many to one relation between Orders and Inventory

**Product\_Inventory:** One to one relation between Product and Inventory

**Users :** This corresponds to Users ORM.

**Orders :** This corresponds to Orders ORM.

**Inventory** : This corresponds to Inventory ORM.

**Product :** This corresponds to Product ORM.

**Ajax Implementation**

AJAX is used to:

1. To display the list of consumers by Admin.
2. To display the list of producers by Admin.
3. To display the order id of the order cancelled by the consumer.

In each of these three instances the display (consumer list, producer list, deleted order id) is provided without needing to reload the page.

**Non-Functional Requirements**  
  
We will be focusing mainly on two non-functional requirements in our project.

• One is **Maintainability**. Our goal is to have all the coding and design procedures stick to professional standards and enable easy modifications in the code and code reusability in the future. The Architectural standards are also to be strictly followed as per requirements without any deviations.  
  
• The second non-functional aspect of the application will be **Usability**.  
For usability this application will have intuitive user interaction on all of the pages in the application. We will use simple user interface enabling easy use of the application and also provide user guidance wherever necessary.