M30COM Report

# Introduction

This document is to support and document my journey while creating the electronics product store application.

In it I hope to demonstrate my understanding of Web Development using Java EE.

This has been an interesting experience as I have a back ground with Asp.Net and C# and I like having the opportunity to experience a different but very similar development experience.

# Overview

The task was to create a web site where users could purchase electronics products. The application was required to have a list of products from 5 different categories that customers could add or remove from there shopping cart.

The technologies I was limited to use were and attempted to incorporate in the project were:

* Java servlets
* JavaBeans
* JFS Pages (Java Faces)
* HTML/CSS
* Jarvascript

# Design

## Foreword

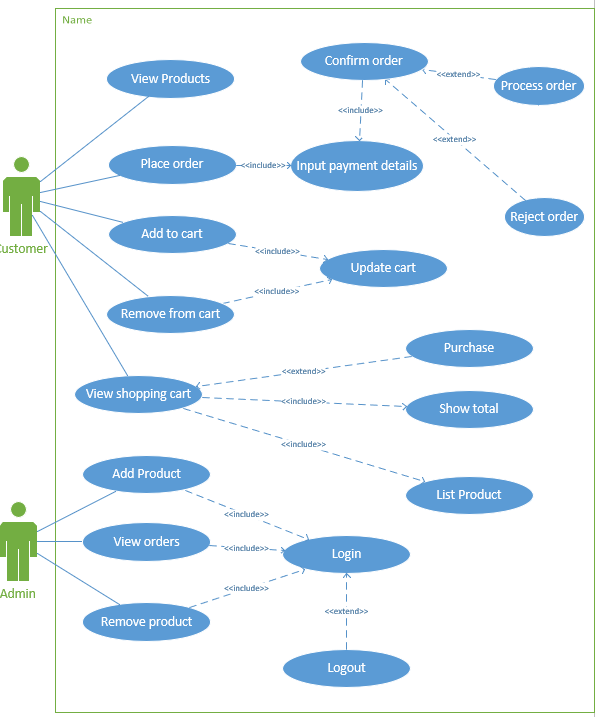
To Begin with I broke down the following paragraph:

“The application should enable **customers** to **buy** electronics **products** online and also provide an **administrative** interface which allows members of staff to keep **track** of customers’ **orders** and enable them to perform various updates functions on database”

From the paragraph above I broke the design into two stages. High-Level(High scope, Low Detail) and Lower-level (High Detail, Low Scope)

## Step 1: High-Level

### Use case diagram



### Use Case Descriptions

#### UC1: Display a list of available products for customers

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Display a list of available products for customers. |
| **Subject area** | Electronics product store |
| **Business event** | Potential order |
| **Actors** | Customer |
| **Scenario** | User visits website and is displayed with a list of products that are currently available. |
| **Pre-Conditions** | Allows the completion of use case:   * UC2 * UC3 * UC6   Products available to show |
| **Post-Conditions** | None |
| **User - System sequence** | |
| **User** | **System** |
| 1. **User arrives on index page.** | 1. **Returns a list of available products grouped by category.** |

#### UC2: Allow customers to add products to a shopping cart

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Allow customers to add products to a shopping cart |
| **Subject area** | Electronics product store |
| **Business event** | Potential order |
| **Actors** | Customer |
| **Scenario** | User visits website and is displayed with a list of products that are currently available. |
| **Pre-Conditions** | None |
| **Post-Conditions** | Allows the completion of use case:   * UC6   Item added to shopping cart.  Allows the placing of an order. |
| **User – System sequence** | |
| **User** | **System** |
| 1. **User pushes Add Item** | 1. **The selected product is added to the shopping cart.** 2. **Item quantities updated.** 3. **Shopping cart total updated.** |
| **Reference Notes** | Viso UseCase page |

#### UC3: Removal of items from the shopping cart

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Removal of items from the shopping cart |
| **Subject area** | Electronics product store |
| **Business event** | Potential order |
| **Actors** | Customer |
| **Scenario** | Use clicks remove on an item and the item is removed from the shopping cart. |
| **Pre-Conditions** | Item is in shopping cart |
| **Post-Conditions** | Item removed from shopping cart  Item quantities updated  Shopping cart total updated |
| **User - System sequence** | |
| **User** | **System** |
| 1. **User pushes remove item** | 1. **The item is removed from the shopping cart** 2. **Item quantities updated.** 3. **Shopping cart total updated.** |
| **Reference Notes** | Viso UseCase page |

#### UC4: Customers can view a summary of all the items and quantities in the shopping cart

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Customers can view a summary of all the items and quantities in the shopping cart |
| **Subject area** | Electronics product store |
| **Business event** | Potential order |
| **Actors** | Customer |
| **Scenario** | User can focus in on the shopping basket, seeing the total price and the items he has added to the basket |
| **Pre-Conditions** | There are items in the basket |
| **Post-Conditions** | The customer is able to order the products |
| **User - System sequence** | |
| **User** | **System** |
| **1 The customer selects view cart.** | 1. **The customers current shopping cart is displayed** 2. **The item subtotals are displayed** 3. **The shopping cart total is showed** |

#### UC5: Enable a customer to place an order and make payment by using fictious credit card details

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Enable a customer to place an order and make payment by using fictious credit card details |
| **Subject area** | Electronics product store |
| **Business event** | A shipping order is placed to the suppliers |
| **Actors** | Customer |
| **Scenario** | User visits website and is displayed with a list of products that are currently available. |
| **Pre-Conditions** | None |
| **Post-Conditions** | * User is presented with a confirmation message OR An unsuccessful message |
| **User - System sequence** | |
| **User** | **System** |
| 1. **User selects place order** 2. **The user Inputs credit car details and shipping address** 3. **The user selects confirm** | 1. **A page to enter the customers credit details is shown** 2. **Credit card details are checked and sent to the CreditCardService with a response returned.** 3. **A successful or unsuccessful message is shown** |

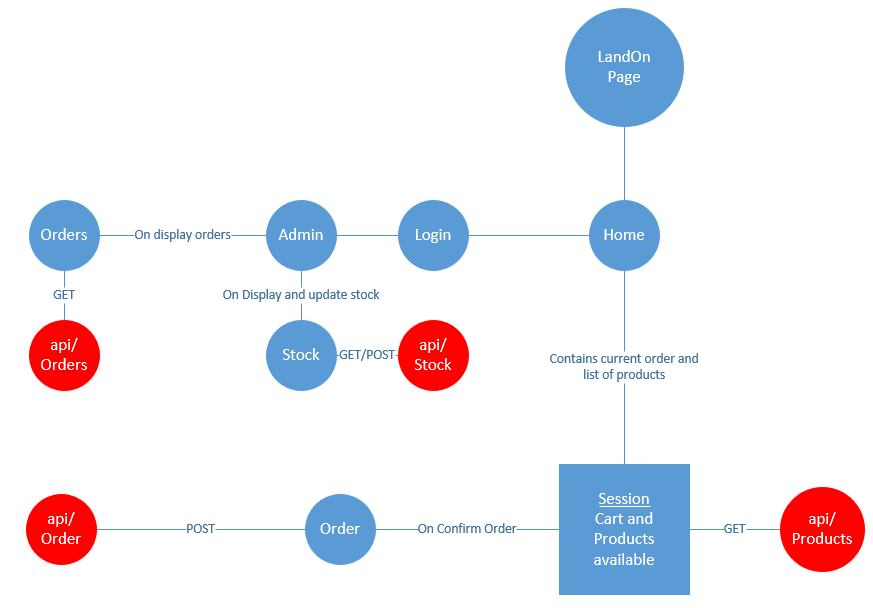
#### UC6: Administrative interface is required to allow staff to view and track customers order, and also allow staff to add new products.

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Administrative interface is required to allow staff to view and track customers order, and also allow staff to add new products. |
| **Subject area** | Electronics product store |
| **Business event** | Potential order |
| **Actors** | Customer |
| **Scenario** | A user has logged in as administrator and now has access to   * Viewing orders placed * Adding products to the available products list * Removing products from the available products list |
| **Pre-Conditions** | The user is Authenticated  The user is Authorized to   * View orders placed * Add products to the available products list   Remove products from the available products list |
| **Post-Conditions** | Database has been updated if an operation has been performed |
| **User - System sequence** | |
| **User** | **System** |
| 1. **User logs in as an admin**   **3A. User selects view orders**  **3B User Select Add Product**  **5B User inputs details for a**  **new product and submits**  **3C User Select Remove Product**  **5C User selects multiple**  **Products from the list of**  **available products and submits** | 1. **Admin dashboard is shown**   **4A A list of orders is fetched from**  **the database and returned to the user**  **4B Interface to input a product**  **is returned**  **6B The product is added to**  **the database**  **4C A list of available products**  **is returned**  **6C The products are removed from the database** |

#### UC7: Backend database should be update correctly in relation to order placed and products delivered

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | Backend database should be update correctly in relation to order placed and products delivered |
| **Subject area** | Electronics product store database |
| **Business event** | Database updated |
| **Actors** | Customer |
| **Scenario** | User visits website and is displayed with a list of products that are currently available. |
| **Pre-Conditions** | User is Authenticated  User is Authorized |
| **Post-Conditions** | Database updated |
| **User - System sequence** | |
| **User** | **System** |
| 1. **User inputs database command** | 1. **Command is sent to the database server** 2. **Response is returned to the user** |

### Site Flow diagram



## Step 2: Low-level

### Class diagrams



### ER Diagram



## Reflection:

This part of the application process can bee seen as the most important part. Most application rely on a solid design which follow best practices etc. I tried to put my knowledge of ER diagrams and SQL database to use but my main background is NoSQL, particularly MongoDB which allows for a much looser design flow, where SQL needs to have a well-defined schema to avoid database anomalies. I always find this stage the hardest as it’s a more abstract way of thinking and I tend to over complicate systems… this time I think I went over board and did the opposite. I could be commended for KIS however (Keep It Simple).

# Implementation

## Foreword

To Implement the design above I used the following tools:

* Java Net-beans IDE
* SQL Local-DB

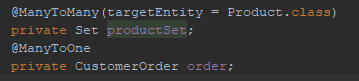
And the following libraries

* JDBC Driver Library (Library for communicating with an SQL database using the JDBC notation
* Eclipse JPA Library (ORM, Similar to entity framework in Microsoft realms)
* JDK Libraries
* Glassfish server library
* Java SDK

I opted to use JSF Pages for my presentation layer. JSF pages work in a MVC (Model View Controller) way which is an industry standard now for separating the concerns of Presentation (**View**), business logic (**Controller**) and persistence (**Model**)

## Step-1: Create package and java models (Entity models)

The first step I took was to take create the package to contain my models. Then I took care of the data models. I began buy using my ER and class diagrams to write the Java classes that will be mapped to an SQL database. For the relationships between object, using the JPA library annotations you inform the JPA driver that a certain property relates to another relation. For example in Purchases.java:



The purchase class has a Many to One relation ship with a CustomerOrders (a purchase can have only one Order but an order can have many purchases) using this annotation the JPA driver will create an extra table to map this relationship.

## Step-2: Create JSF pages

Once I had the Entity models made I could auto generate the JSF pages based. Using the auto generation feature for Product, Purchase, Category, CustomerOrder and selecting “Create Persistence Unit” from the list of options, Net beans generated the following components that were generated for each model are as follows:

* Package
  + Controller
  + Façade (Data Repository)
* Web-Pages
  + Component
    - View
    - Edit
    - List
    - Create

The auto generated classes greatly helped speed up my understanding of jsf pages, with most of the required functionality being somewhere in the generated code.

## Reflection

# Testing

**Include testing results. Show unit tests output**

## Foreword

Testing is an important part of a product life cycle

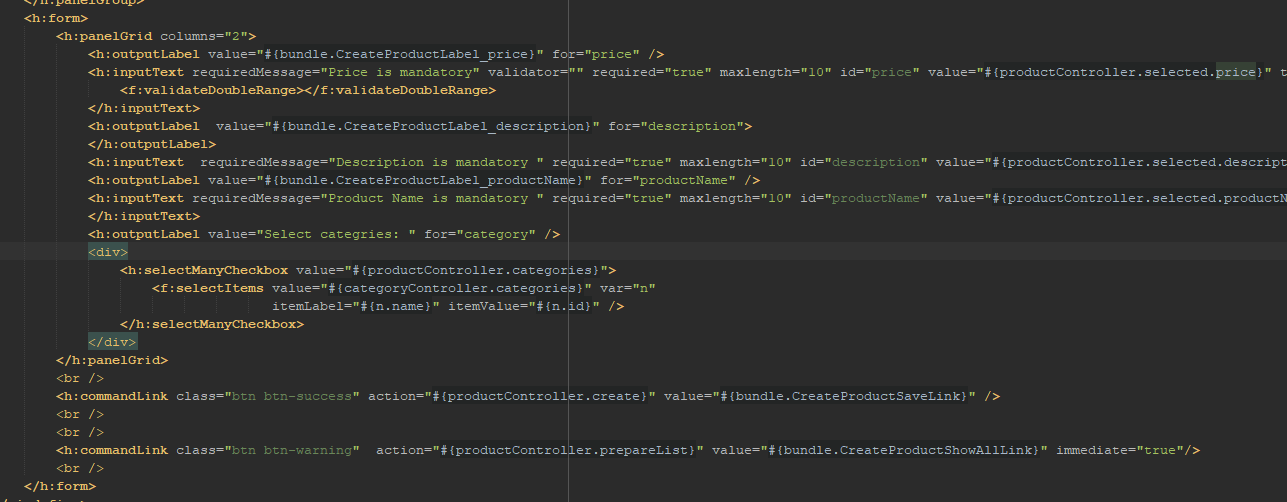
## Step 1: Unit tests

## Reflection

# Validation

## Client side

I have used html attributes for the client side attributes.

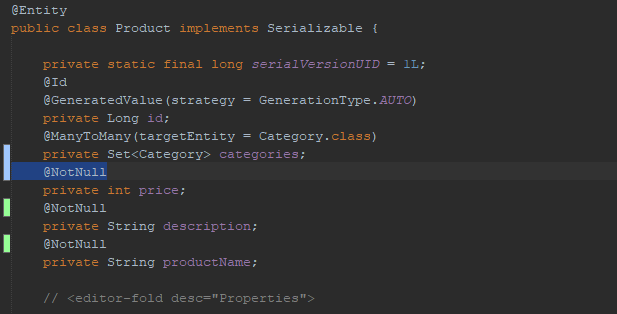


Here you can see for both the name and description inputs I have used the required attribute to make sure that when you create a category you must give it name and a description.

I have also used the max-length attribute to apply some further constraints to ensure for sensible inputs.

For the price I apply the validateDoubleRange validatior from the core jsf namespace.

## Server side validation



I have opted for using the java beans annotations for the majority of the server side validation. The annotiations are then applied to the

The @NotNUll attribute will instruct the SQL DBMS to create the column for that property with the constraint NOTNULL.

# Source Control

For source control I used git, which in my opinion is the easiest source control system to use and in my experience the easiest to recover from mistakes.

# Appendix:

## Template:

|  |  |
| --- | --- |
| **Use case Desription** | |
| **Use-case field** | **Description** |
| **Use case name** | An active verb phrase that describes a particular task. |
| **Subject area** | A use role or other grouping mechanism that can be used to group use cases. |
| **Business event** | A trigger that stimulates activity within the business. Many business events occur at the interface point between the business and one of the external entities with which it interacts. Business events must be observable. |
| **Actors** | The actor that initiates this use case and all users who participate in this use case. |
| A description of the overview of the use case |
| **Pre-Conditions** | Constraints that must be met for the use case to be taken by the solution developer and used to create a workflow. This might include a required sequencing of use cases. For example, one or more other use cases might need to be performed successfully for this use case to begin. |
| **Post-Conditions** | Constraints that must be met after the sequence of the use case has completed. |

# Notes

## Java Platform, Enterprise Edition (Java EE)

Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software.

Java EE is developed using the [Java Community Process](http://www.jcp.org/), with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity.

## Jave persistence API (JPA)

The **Java Persistence API** (**JPA**) is one possible approach to ORM (Object Relational Mapping). Via **JPA** the developer can map, store, update and retrieve data from relational databases to **Java** objects and vice versa. **JPA** can be used in **Java**-EE and **Java**-SE applications. **JPA** is a specification and several implementations are available.

## JDBC Resource

To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API.

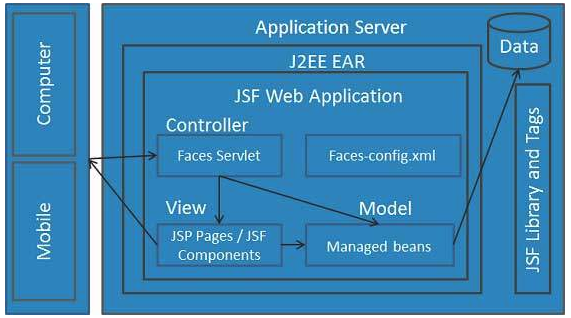
A JDBC resource (data source) provides applications with a means of connecting to a database. Typically, the administrator creates a JDBC resource for each database accessed by the applications deployed in a domain. (However, more than one JDBC resource can be created for a database.)

## JavaServer Faces

**JavaServer Faces** (JSF) is a MVC web framework that simplifies the construction of User Interfaces (UI) for server-based applications using reusable UI components in a page.

JSF provides a facility to connect UI widgets with data sources and to server-side event handlers.

The JSF specification defines a set of standard UI components and provides an Application Programming Interface (API) for developing components. JSF enables the reuse and extension of the existing standard UI components.



**Components:**

## Class FacesServlet

**FacesServlet** is a servlet that manages the request processing lifecycle for web applications that are utilizing JavaServer Faces to construct the user interface.

## Java Bean

Managed beans

L

A Java Bean is a java class that should follow following conventions:

* It should have a no-arg constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.

//Employee.java

Example Java Bean

**package** mypack;

**public** **class** Employee **implements** java.io.Serializable{

**private** **int** id;

**private** String name;

**public** Employee(){}

**public** **void** setId(**int** id){**this**.id=id;}

**public** **int** getId(){**return** id;}

**public** **void** setName(String name){**this**.name=name;}

**public** String getName(){**return** name;}

}

## Steps/Process:

Creating EntityModelsPackage

Created ModelPackage

Creating Entity Class with option “ create Persistence Unit” selected

PersistenceUnit Name is ElectronicProductStorePU with EclipseLink (JPA 2.1)(default) as Persistence Provider

Created 4 entities Customer, Order,Product,Purchase. Added there cardinatilites, onetomany many to one on the properies

Tutorial: <https://www.tutorialspoint.com/jpa/jpa_entity_relationships.htm>

* @ManyToOne Relation
  + Child:
    - @ManyToOne
    - private Department department;
  + Parent: nothing
* @OneToMany Relation
  + Child: Private Object objectname;
    - @OneToMany( targetEntity=Employee.class )
    - private List employeelist;
  + Parent: nothing
* @OneToOne Relation
* @ManyToMany Relation

Had Issue with Creating Order Entity Due to ORDER being a keyword in sql, the same with desc

Creat JSF pages from Entity classes

Create JDBC Resource for JPA Api to Connect to the database.

Setting:

JNDI name: org.apache.derby.jdbc.ClientDataSource

Creat a new JDBC Connectio Pool

DataSource class anemorg.apache.derby.jdbc.ClientDataSource

Database connection is extracted for excisting data base created earlier (Electronic product store)

URL: jdbc:derby://localhost:1527/ElectronicProduct

serverName: localhost

portNumber: 1527

DatabaseName:L ElectronicProduct

User: chris

Password: 1234

Configuerd User

Name:admin

Password:1234

Groups: CanEditProductList,CanViewOrders

I managed to get the authentication working, main issue was either not restarting the Glass server or not clean building