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:

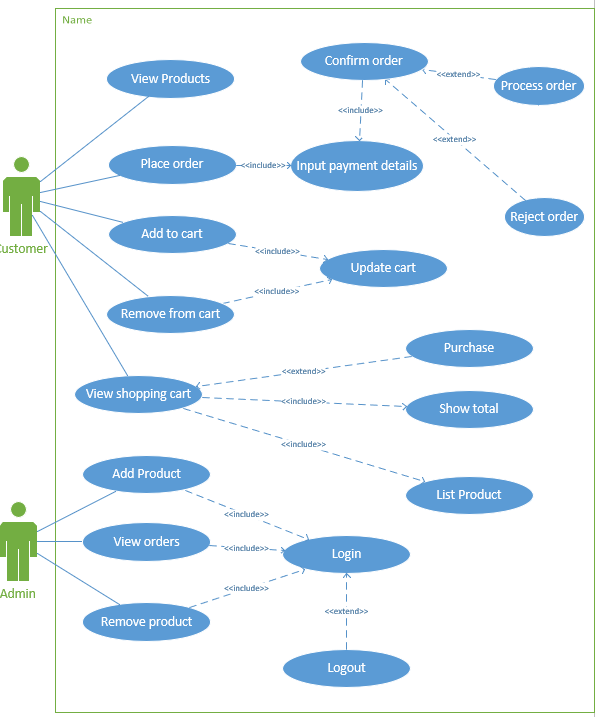
“The application should enable **customers** to **buy** electronics **products** online and also provide an **administrative** interface which allows members of staff to keep **track** of customers’ **orders** and enable them to perform various updates functions on database”

From the paragraph above I broke the design into two stages.

High-Level (High scope) and Low-level (High Detail)

## 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** |

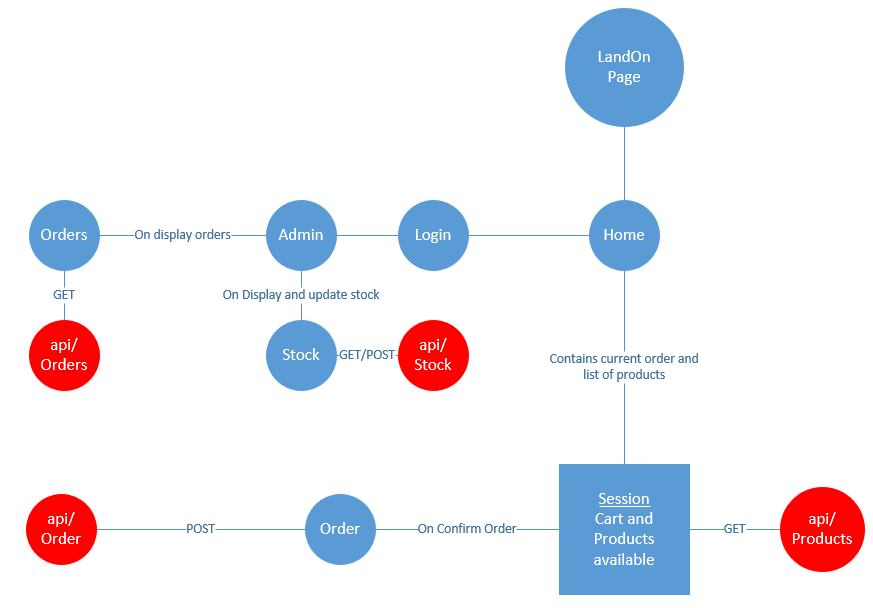
#### UC6: Administrative interface is required to allow staff to view and track customers order, and also allow staff to add new products.

|  |  |
| --- | --- |
| **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** |

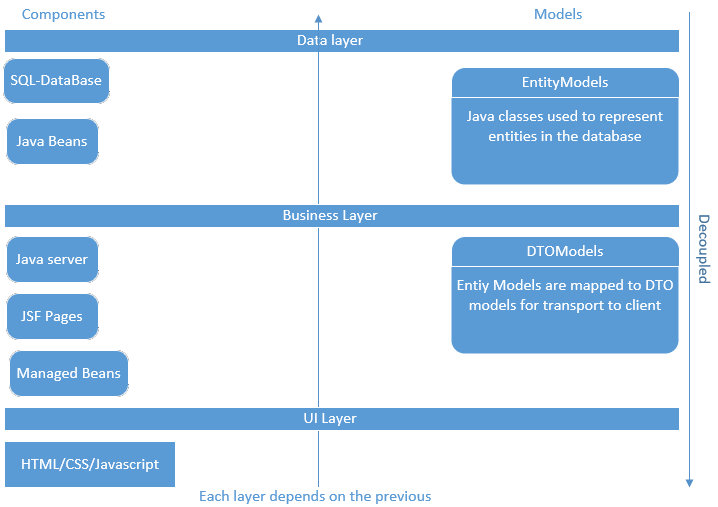
#### UC7: Backend database should be update correctly in relation to order placed and products delivered

|  |  |
| --- | --- |
| **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



### Application layering



## Step 2: Low-level

### Class diagrams



### ER Diagram



## Reflection:

This part of the application process can be seen as the most important part.

Most applications rely on a solid design which follows best practices.

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.

SQL needs to have a well-defined schema to avoid database anomalies, which is a benefit. 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 have learnt a lot and I’m finding the design stage more enjoyable than a chore.

Visualizing how you use your data helps guide the direction of the system.

# 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 use MVC (Model View Controller) design flow which is an industry standard for separating the concerns of Presentation (**View**), business logic (**Controller**) and persistence (**Model**).

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

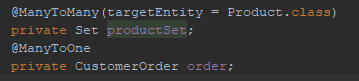
The first step I took was to create the packages to contain my main models:

They are as follows:

1. EntityModels (Contains the Java beans (Entities) as well as the Managed Beans (Controllers) and Data Repository (Façade)
2. Models (Various component which are not to be persisted)
3. Servlets (Folder to contain the Api end points not associated with a particular Model)

I then 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 using the JPA.

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 relationship with a CustomerOrders (a purchase can have only one Order but an order can contain many purchases) using the annotations the JPA driver will create an extra table to map this relationship Called PRODUCTCATEGORY.

## Step-2: Create JSF pages

Once I had the Entity models, I could auto generate the JSF pages. 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 for each model:

* In EntityModels 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.

Inside the XHTML File you can see the reference to the controllers and the use of the Beans to display the data.

The server container looks at the XHTML file and creates a JSF Servlet with end point that match the public methods etc referenced in the file.

## Step-3: Setup Admin Authentication

In this step I needed to gain access to the glass fish server and add an account with a set of groups assigned to it. I then had to place a **ServletSecurity** annotation above the protected routes (java servlets) that I wanted the authentication check to kick in. Once the user logged in, the session cookie is then updated and will be valid up until its expiry time.

The user and groups assigned are as follows:

**User**: admin

**Password**: 1234 (Unsecure)

**Groups**:

1. **CanEditProductList**
2. **CanViewOrders**

Using the annotation:



## Reflection

This part took longer than normal, I found the NetBeans IDE clunky and the java documentation saturated with too much useless information in parts.

Once I got into the flow, I did find the JSF pages very similar to the Razor pages in ASP.Net and they became a joy to work with.

I don’t like having to use custom elements however provided by providers as I find it obscures what’s going on under the hood, for example the creation of the standard html elements and attributes, but some are useful in certain instances and are needed to simplify development. A noticeable example is the form control which, along with command Links and Command Buttons, Hooks everything up on the client to communicate back with the server.

The authentication process is not the best and feels clunky. Having to add users manually in the Glash-Fish server is very rigid and would be difficult for IT administrators to maintain.

I’m sure there is a way to create a backing store of users, or incorporate a third-party identity provider such as Azure AD, but due to time constraints I could not.

Using the annotations to apply authentication to the request pipeline is a common method. When the Container gets a request for a servlet it will use some kind of reflection to look at the attributes assigned to it and execute them.

I must have spent nearly an hour wondering why it wasn’t working and it turned out I needed to restart the GlashFish server to see the changes.

# Testing

## Foreword

Testing is an important part of a product life cycle.

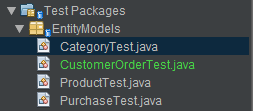
A common approach to development is TDD (Test Driven Development).

This approach requires that you write a test for the component your testing, this test initially fails but as you write code and add functionality to the component your testing the test should eventually pass.

## Step1: Unit Tests

Units tests are an important process to create and maintain quality software. The idea is to break your system in to atomic parts and test them in an isolated environment, providing there inputs and asserting their outputs.

In the Test Packages package, I have crated unit tests to cover as much functionality as possible testing the public methods on the Entity Models.

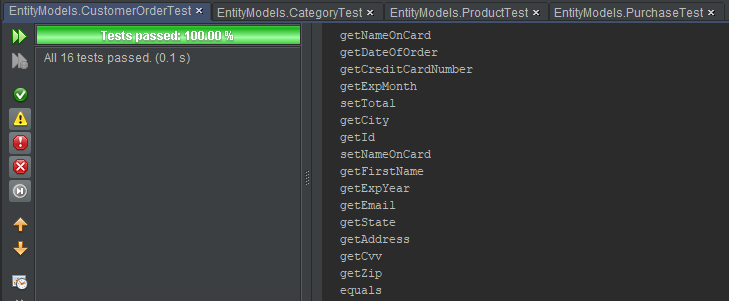


Although the tests a simple they have never the less protected me against any unwanted changes that could be applied to my classes. If for example someone decides to add functionality by appending characters to a model’s name, my tests will find this before it has been sent out into production.

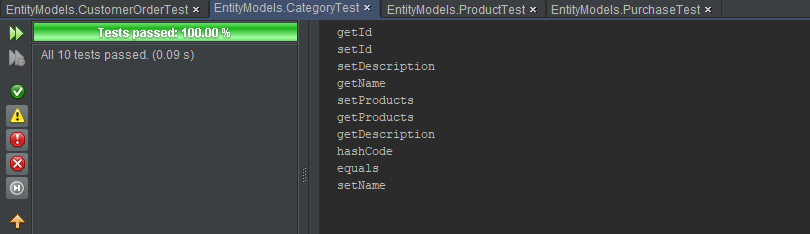
I wanted to use a mocking framework but I couldn’t find sufficient documentation with my allowed time frame.

## Step2: Test Results

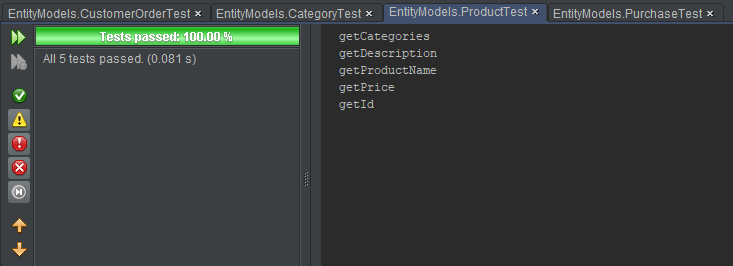
### CustomerOrder-Tests



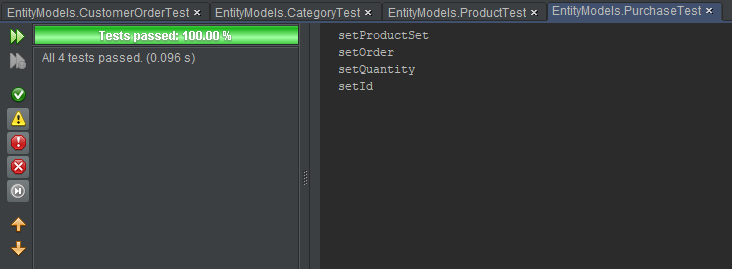
### Category-Tests



### Product-Tests



### Purchase-Tests



## Reflection

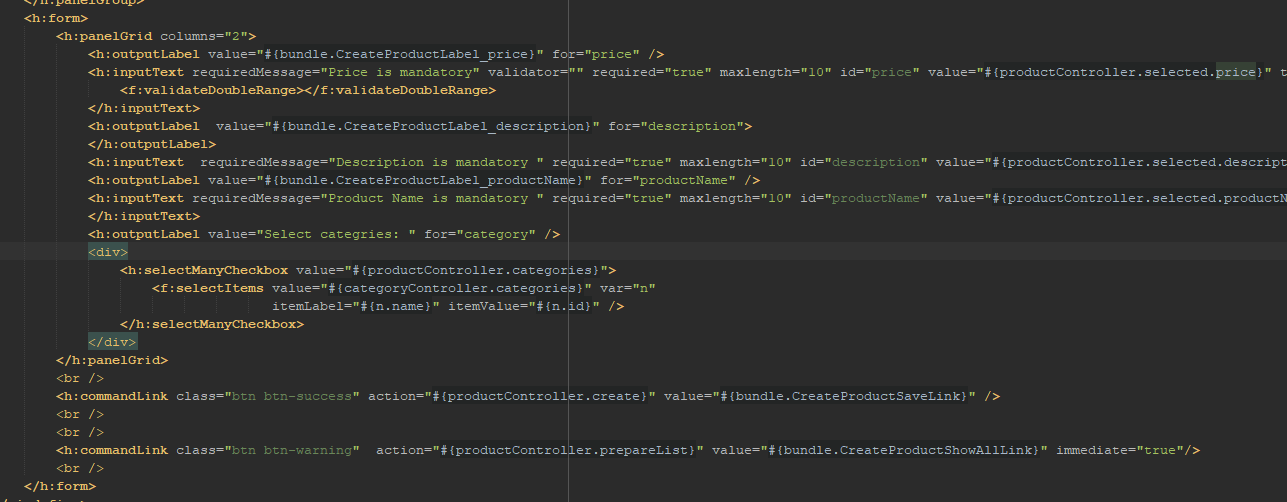
I tried my hardest to find a mocking framework that I could easily incorporate into the project but I couldn’t find one as simple as pulling in a library and using it.

With a mocking framework I could have tested a lot more functionality but without the ability to inject dependencies and check the various calls on the dependency I felt that I couldn’t test the rest of the functionality.

# Validation

## Client-side

I have used html attributes for the client-side validation.

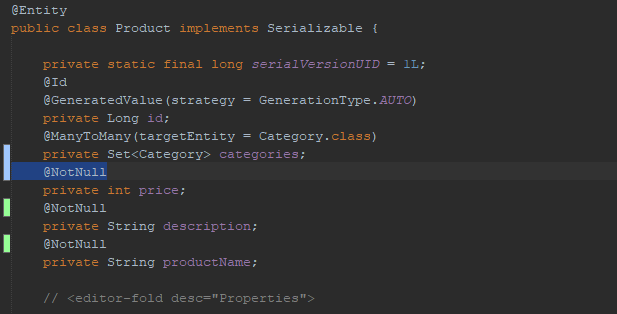


Here you can see for both the name and description inputs I have used the required attribute to make sure that when you create a category you must give it name and a description.

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 validator from the core JSF namespace. This makes sure that the user can only input numbers;

## Server-side validation



I have opted for using the java beans annotations for the majority of the server-side validation.

The annotations are then applied to the SQL tables and an exception will be thrown if the user attempts to input invalid data.

For example the @NotNull attribute will instruct the JPA Driver to create the column for that property with the constraint NOTNULL.

# Source Control

For source control I used git. Git makes it easy to work on multiple machines, track changes, revert and can get you out of most Difficult situations.

# 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. |