**ABSTRACT**

In this modern world one of most developing websites would be an e-commerce website. These websites make people to buy almost everything without stepping out of the door . Electronic Commerce is process of doing business through computer networks. A person sitting on his chair in front of a computer can access all the facilities of the Internet to buy or sell the products.

Unlike traditional commerce that is carried out physically with effort of a person to go & get products, ecommerce has made it easier for human to reduce physical work and to save time.    E-Commerce which was started in early 1990’s has taken a great leap in the world of computers, but the fact that has hindered the growth of e-commerce is security. Security is the challenge faced in e-commerce today & there is still a lot of advancement made in the field of security.

I have created an e-commerce portal with cosmetics as my selling product .Cosmetics are substances or products used to enhance the appearence of a person which helps them to present themselves with a charm.

I have used Eclipse IDK and have developed a maven project.

**TECHNOLOGIES AND SOFTWARES USED:**

Front-End:

* HTML5
* CSS3
* Bootstrap
* AngularJS

Back-End:

* H2(database)
* Hibernate
* DAO

Middleware:

* SpringMVC(framework)

**MAVEN:**

Maven is a build automation tool used primarily for Java projects. The word maven means "accumulator of knowledge" in Yiddish.

Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies. Contrary to preceding tools like Apache Ant, it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging.

Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache. This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project.

Maven is built using a plugin-based architecture that allows it to make use of any application controllable through standard input. Theoretically, this would allow anyone to write plugins to interface with build tools (compilers, unit test tools, etc.) for any other language. In reality, support and use for languages other than Java has been minimal. Currently a plugin for the .NET framework exists and is maintained, and a C/C++ native plugin is maintained for Maven 2.

Alternative technologies like Gradle and sbt as build tools do not rely on XML, but keep the key concepts Maven introduced. With Apache Ivy, a dedicated dependency manager was developed as well that also supports Maven repositories.

**FRONT-END**:

HTML5:

Hypertext Markup Language revision 5 (HTML5) is markup language for the structure and presentation of World Wide Web contents. HTML5 supports the traditional HTML and XHTML-style syntax and other new features in its markup, New APIs, XHTML and error handling.

There are three organizations that are currently in charge of the specification of HTML5:

1. Web Hypertext Application Technology Working Group (WHATWG) created the HTML5 specification and is in charge of the HTML5 development that provides open collaboration of browser vendors and other involved parties.
2. World Wide Web Consortium (W3C) is in charge with delivering the HTML5 specification.
3. Internet Engineering Task Force (IETF) is in charge of the development of HTML5 WebSocket API.

CSS3:

A cascading style sheet (CSS) is a Web page derived from multiple sources with a defined order of precedence where the definitions of any style element conflict. The Cascading Style Sheet, level 1,recommendation from the World Wide Web Consortium , which is implemented in the latest versions of the Netscape and Microsoft Web browsers, specifies the possible style sheets or statements that may determine how a given element is presented in a Web page.

CSS gives more control over the appearance of a Web page to the page creator than to the browser designer or the viewer. With CSS, the sources of style definition for a given document element are in this order of precedence:

1. The STYLE attribute on an individual element tag
2. The STYLE element that defines a specific style sheet containing style declarations or a LINK element that links to a separate document containing the STYLE element. In a Web page, the STYLE element is placed between the TITLE statement and the BODY statement.
3. An imported style sheet, using the CSS @import notation to automatically import and merge an external style sheet with the current style sheet
4. Style attributes specified by the viewer to the browser
5. The default style sheet assumed by the browser

In general, the Web page creator's style sheet takes precedence, but it's recommended that browsers provide ways for the viewer to override the style attributes in some respects. Since it's likely that different browsers will choose to implement CSS1 somewhat differently, the Web page creator must test the page with different browsers.

Bootstrap:

A bootstrap is a small strap or loop at the back of a leather boot that enables you to pull the entire boot on. In computers, to bootstrap (or "to [boot](http://searchwinit.techtarget.com/definition/boot)") is to load a program into a computer using a much smaller initial program to load in the desired program (which is usually an operating system). In general usage, bootstrapping is the leveraging of a small initial effort into something larger and more significant. There is also a common expression, "pulling yourself up by your own bootstraps," meaning to leverage yourself to success from a small beginning.

AngularJs:

AngularJS (commonly referred to as "Angular" or "Angular.js") is a complete JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. The JavaScript components complement Apache Cordova, the framework used for developing cross-platform mobile apps. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications.

The AngularJS framework works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources.

**BACK-END:**

H2(database):

H2 is a relational database management system written in Java. It can be embedded in Java applications or run in the client-server mode. The disk footprint (size of the jar file) is about 1.5 MB.

In many cases H2 is faster than other (open source and not open source) database engines. Please note this is mostly a single connection benchmark run on one computer, with many very simple operations running against the database. This benchmark does not include very complex queries. The embedded mode of H2 is faster than the client-server mode because the per-statement overhead is greatly reduced.

Hibernate:

**Object/Relational Mapping**

Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC).

**JPA Provider**

In addition to its own "native" API, Hibernate is also an implementation of the Java Persistence API (JPA) specification. As such, it can be easily used in any environment supporting JPA including Java SE applications, Java EE application servers, Enterprise OSGi containers, etc.

**Idiomatic persistence**

Hibernate enables you to develop persistent classes following natural Object-oriented idioms including inheritance, polymorphism, association, composition, and the Java collections framework. Hibernate requires no interfaces or base classes for persistent classes and enables any class or data structure to be persistent.

**High Performance**

Hibernate supports lazy initialization, numerous fetching strategies and optimistic locking with automatic versioning and time stamping. Hibernate requires no special database tables or fields and generates much of the SQL at system initialization time instead of at runtime.

Hibernate consistently offers superior performance over straight JDBC code, both in terms of developer productivity and runtime performance.

**Scalability**

Hibernate was designed to work in an application server cluster and deliver a highly scalable architecture. Hibernate scales well in any environment: Use it to drive your in-house Intranet that serves hundreds of users or for mission-critical applications that serve hundreds of thousands.

**Reliable**

Hibernate is well known for its excellent stability and quality, proven by the acceptance and use by tens of thousands of Java developers.

**Extensibility**

Hibernate is highly configurable and extensible.

**DATA ACCESS OBJECTS**:

Data Access Objects (or DAOs for short) are used as a direct line of connection and communication with our database. DAOs are used when the actual CRUD (CRUD = Create, Read, Update, Delete) operations are needed and invoked in our Java code. These data access objects also represent the “data layer” of our application.

These objects are still just plain old Java objects that incorporate the use of some Hibernate annotations to give them the functionality we need from them. Again, that functionality being the communication with the database.

Also, believe it or not, the concept of creating a file specifically for accessing the database is a design pattern , it’s called the Data Access Object Pattern.

## WHAT HIBERNSTE ANNOTATIONS SHOULD WE USE?

To integrate these plain old Java objects with our Hibernate framework (so they actually do the work we need them to do),

There are two main annotations that you need to be familiar with:

* @Repository
* @Transactional

The first of the two annotations @Repository is one from Spring and it’s pretty straight-forward. It’s used to mark the Java file as something Spring calls a “Component”, which enables it to be scanned and incorporated into Spring’s code. You really just need to remember to put this annotation in all of your DAOs (on the class level) and you’ll be good to go.

The second (and more complicated) annotation is the @Transactional annotation. This annotation is used as a means to enable transaction management within your Java DAO file.

Transaction management is all about data integrity , which means it’s all about keeping your data valid, non-corrupt and error free . Let’s think of the opposite scenario, having our data be invalid, corrupt and full of errors.

## WHAT IS A TRANSACTION?

A transaction (as it pertains to databases) is the trick we use to be able to roll back (or revert) a database operation. Consider the following database operation:1.PNG

This database operation has to do a few things:

1. Insert a new row of data into the users table
2. Assign a primary key to this new row
3. Populate the new row with the provided data

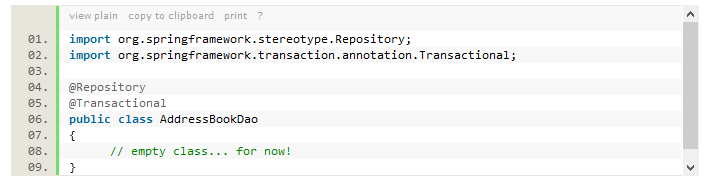
What happens if your database server crashed after step #1? Or what happens if someone yanks the power cord out of the server by accident after step #1? What happens if the disk drive becomes full after step #2?

All of these scenarios would leave the database operation only “partly committed”. If we were to leave the data as is, in its partially committed state, then our data would be corrupt and could lead to errors in our application.

Its job is to “roll back” the database operation if it doesn’t complete correctly. So let’s assume that we open a transaction, start our database operation, get to step #2 and then there’s a failure. The database keeps track of the “state” its in and (once restored) sees that there’s a transaction that was not fully committed, so it will roll back to the previously known “stable state” , thus eliminating the partially committed (and invalid) data.

## EXAMPLE OF REAL DATA ACCESS OBJECT:

Let’s take a look at an example DAO class, let’s take a look at what the beginning of our Address Book DAO would look like:



As you can see above, we have put both of the annotations that we’ve just learned about at the class level for the DAO.

Now let’s talk about the SessionFactory.

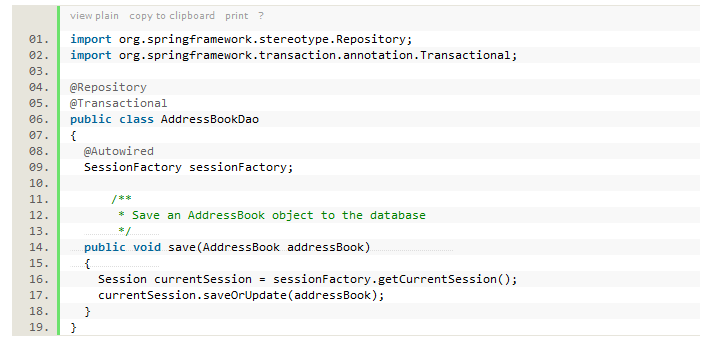
## SESSION FACTORY:

A SessionFactory is what we use to grab individual Sessions so that we can talk to our database. It uses the Factory design pattern, which essentially means it’s sole purpose is to dish our Sessions whenever you ask for them.

**Session**

A Session is what we use to create open our initial transaction, perform some kind of database operation, and then commit the transaction (or roll it back if there was a failure).

Let’s take a look at how we get a Session in Java:

 @Autowired annotation is used to automatically populate the sessionFactory with an actual real instantiated instance. Spring can do this because we’ve already configured the SessionFactory in our PersistenceConfig class.

Once we have a real instance of the SessionFactory we can use it to grab a real Session. We do so by invoking the getCurrentSession() method and assign it to a variable.

Now that we have a Session we can invoke one of our CRUD operations. In the example above, I am invoking a saveOrUpdate(), which will either create a new row of data or update an existing row.

## HIBERNATE’S saveOrUpdate METHOD:

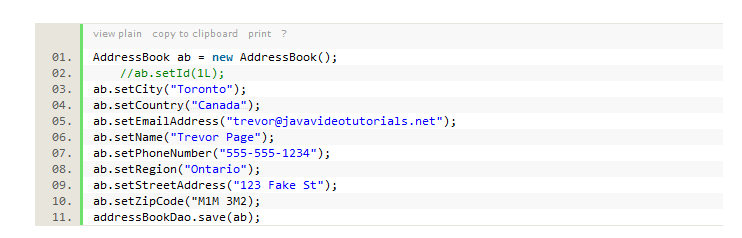
It incorporates one of the fundamental rules by which Hibernate operates.

How the is Hibernate supposed to know wheather or not you wish to create a new row of data with the object you’ve given to it, or if it should update an existing row of data?

Let’s assume you’ve created a brand new AddressBook object and populated it with some relevant information. Now you wish to save it to the database ,you know that it’s a new piece of data, and therefore it should have a new row created in the database , but how does Hibernate know?

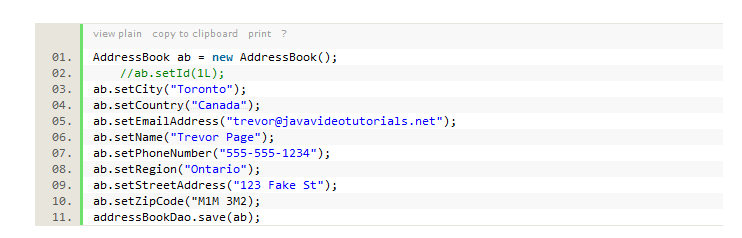
Hibernate looks at the @Id variable that you’ve defined in the AddressBook class! If there’s an actual ID value assigned to your object, then that means it must already exist in the database and therefore this operation should be an UPDATE , if there’s no ID valued populated in the AddressBook object (i.e. it’s null), then it should be a new row and it performs an INSERT.

Let’s say we have the following AddressBook scenario:

 What do you think will happen when we invoke the save(ab) method?

If you look carefully, you’ll see that I’ve assigned a value to the ID field , so given my explanation above, that means that Hibernate will perform an UPDATE.

Now, what happens if we get rid of the line of code that assigns the ID?

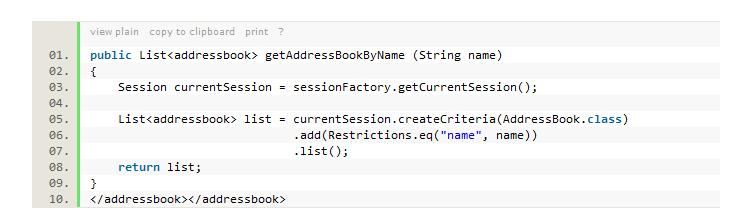


Now when we run the save(ab) method, hibernate sees that there’s no ID set, so it will perform an INSERT.

## HOW DO WE READ FROM THE DATABASE WITH HIBERNATE:

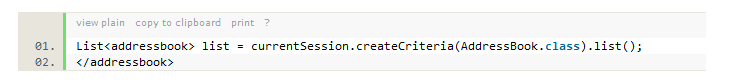
Criteria is the object that we will use to create SQL queries in Java , if you like you can also create native SQL queries with Hibernate, but it’s actually a LOT easier to use Criterias.

In the example below, you will see how we create the Criteria to search the address\_book table for the name that we will pass in:



We use a Session to create the Criteria and we add Restrictions to filter our results.

To illustrate how this works, let’s assume we just use this Criteria:



This will create the following SQL query:

 This is a very broad search, and thus we need to add a filter on it to make it more narrow. Remember that our current mission is to retrieve a particular person’s address information. We do this by adding Restrictions, and in our example we restrict the broad results by searching for a name.

We then invoke the list() method which will actually execute the query and return a list of results. If there are no matching entries then it will be an empty list, if there are one or more entries, then they will be populated in our list.

There is lots more we can do with the Criteria object in Hibernate, such as grouping and ordering our data .

## HOW DO WE DELETE ROWS FROM DATABASE:

You need to LOAD the row from the database before you can delete it. This is something built into the Hibernate framework and it’s another concept that is important to understand. If you simply just create your own object manually and then pass it in to a delete method, hibernate won’t be able to carry out the operation as the object you’ve asked to be deleted is “detached”.

There’s some great information on the “state” of objects as they pertain to the Hibernate framework. It’s a bit of a heavy read, but it’s definitely good to understand how this stuff works.

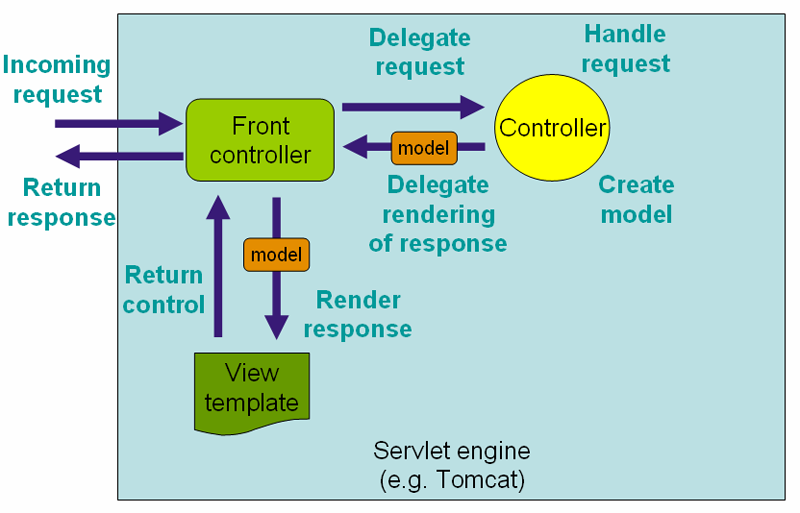
You need to make sure the object that you wish to delete is the actual persistent version of the object. So this means you’ll need to first load the object from the database before invoking the delete. Here’s how to do it:

 Notice the use of the uniqueResult() method. Previously when we read data from the database, we used the list() method to invoke our query, but here we have a slightly different scenario. We are loading the object by the ID, so we know there is only going to be ONE result back from the database… since we know we are only getting one object back, we can ask for a uniqueResult as opposed to getting list.

**Spring MVC:**

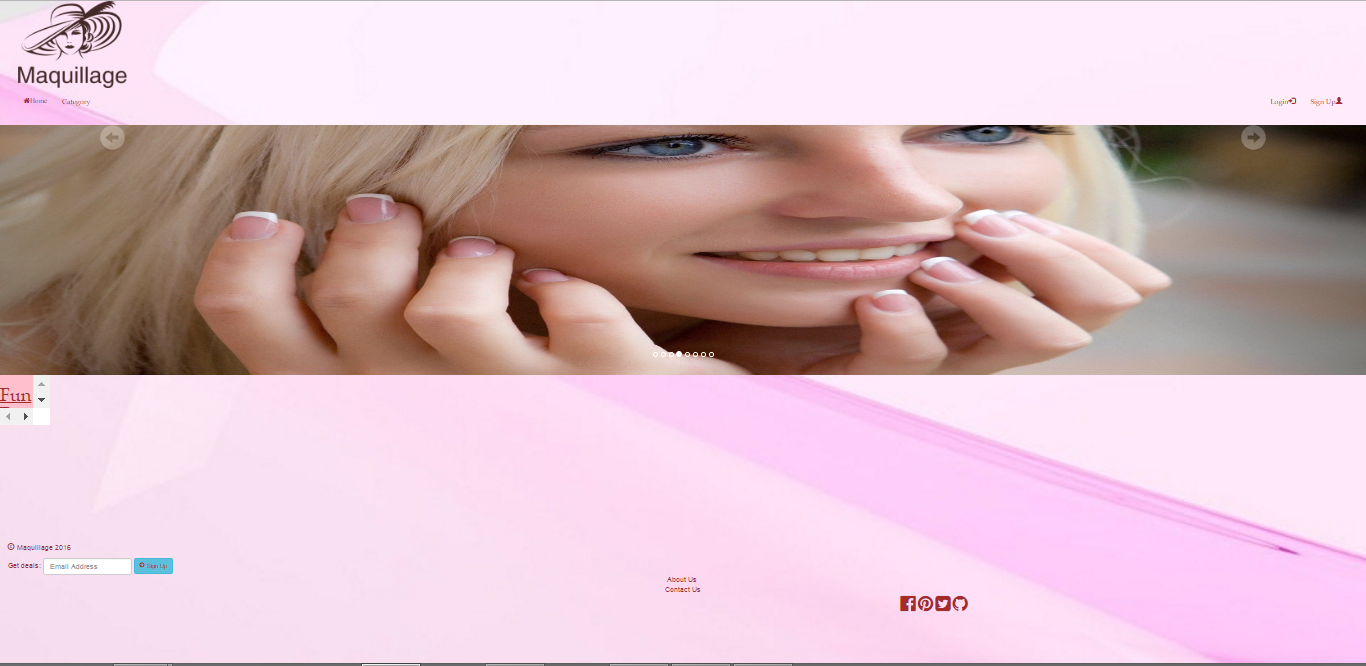
The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

* The Model encapsulates the application data and in general they will consist of POJO.
* The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The Controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

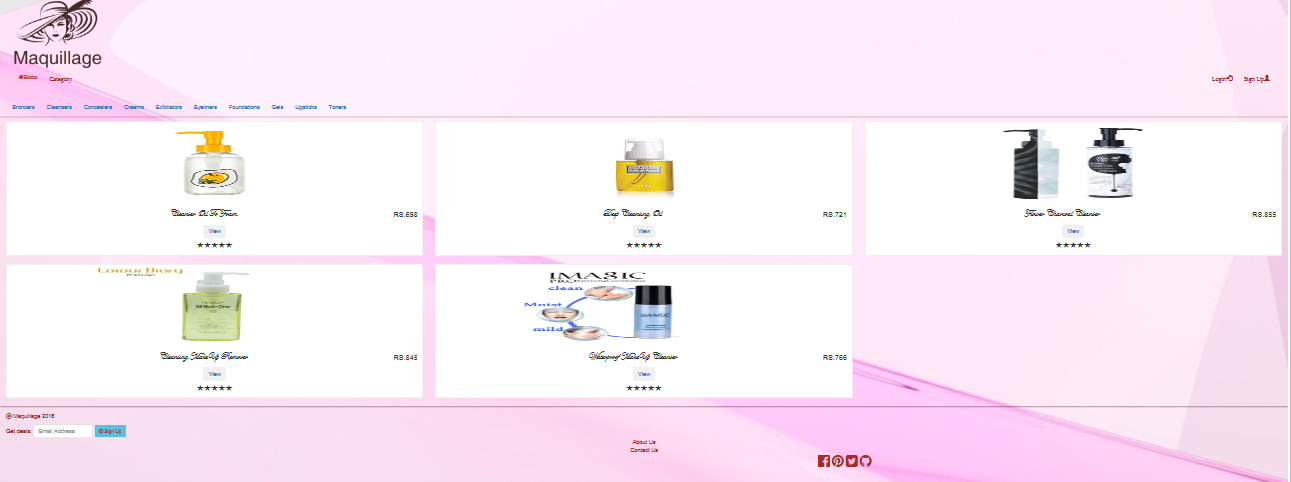


**MY PROJECT PAGES:**

**BEFORE LOGIN**

**Index page :** 

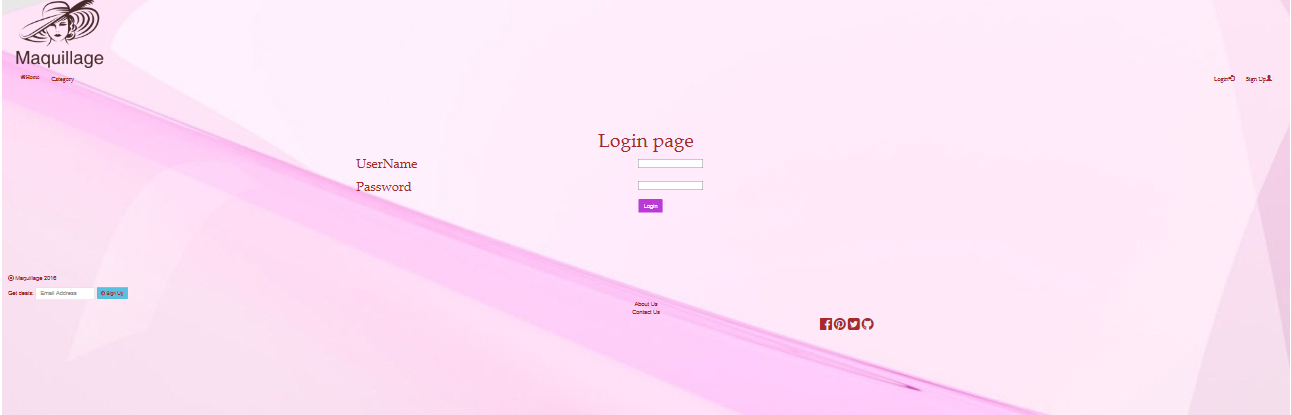
**Category page:**



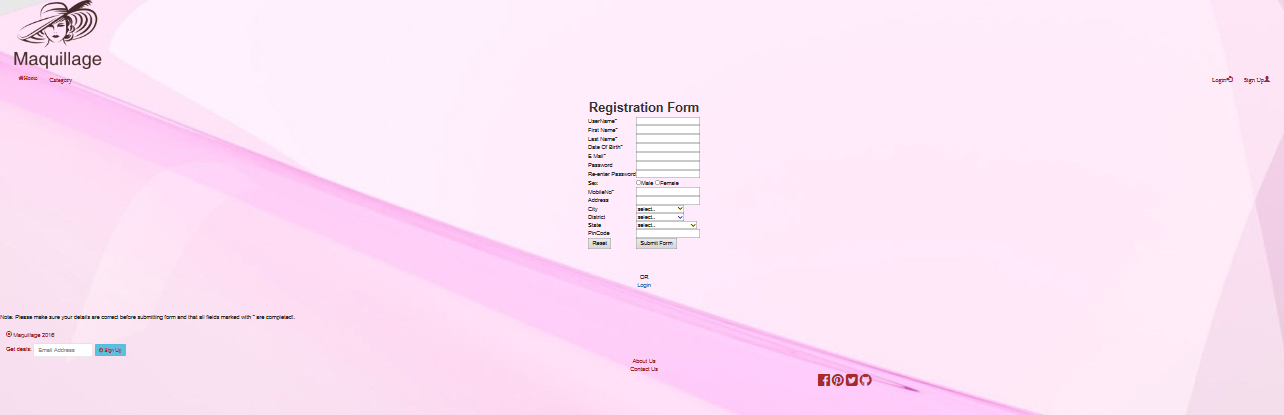
**Product page:**



**Login page:**

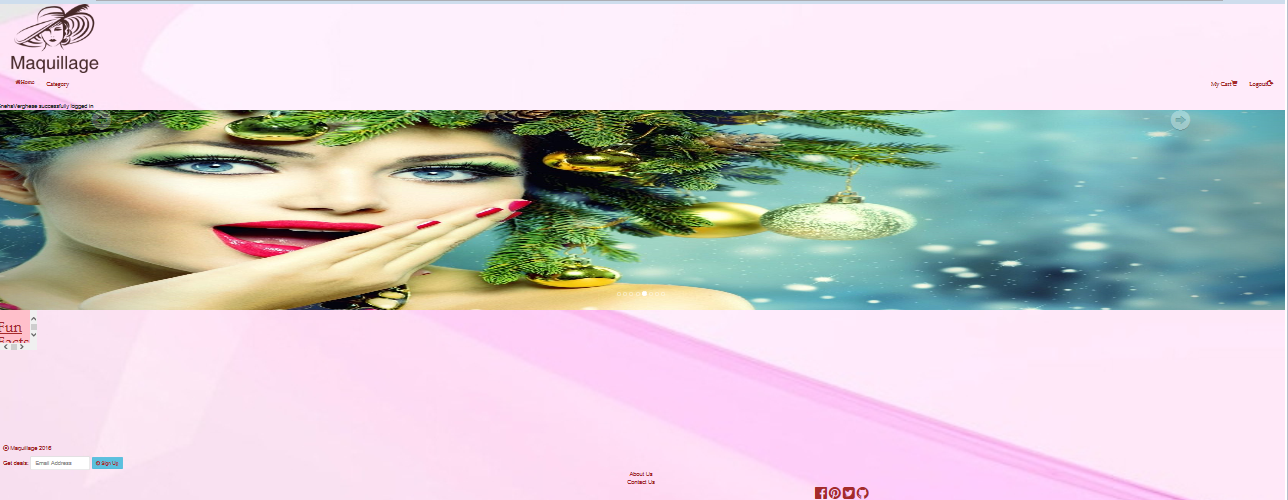


**Register page:**

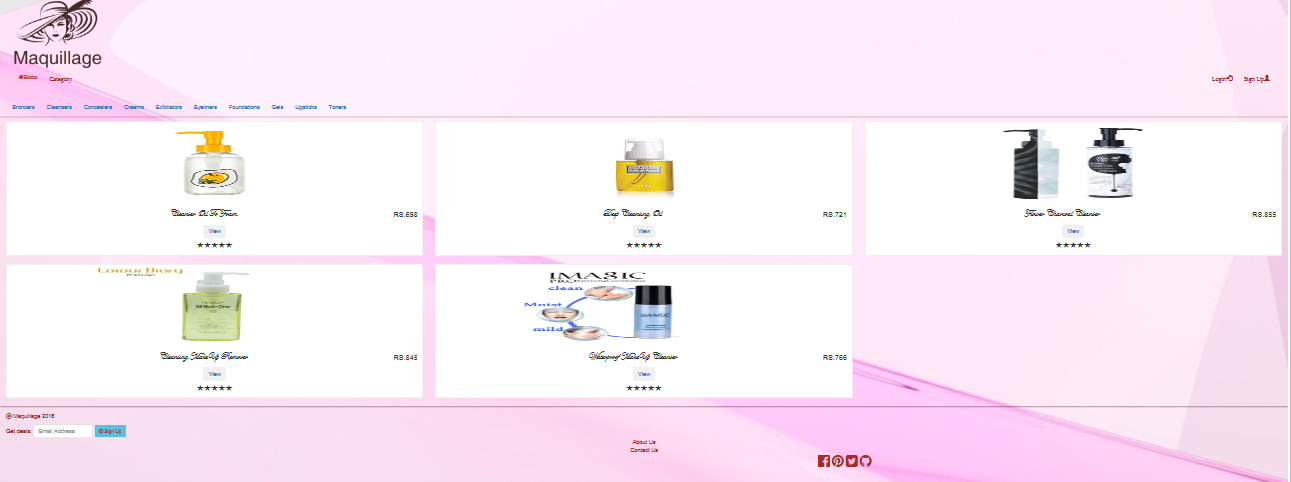


**AFTER LOGIN:**

**Index page:**



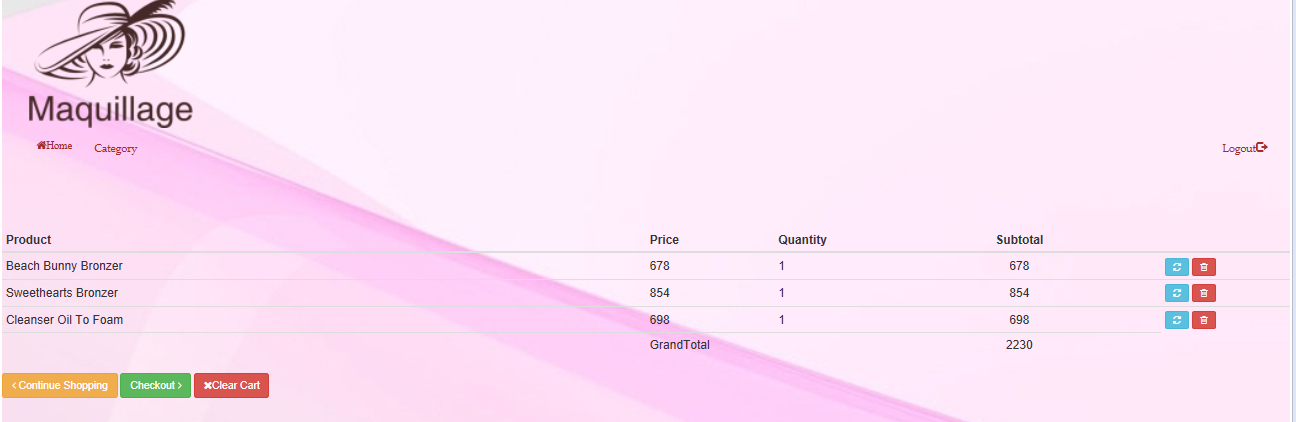
**Category page:**



**Product page:**



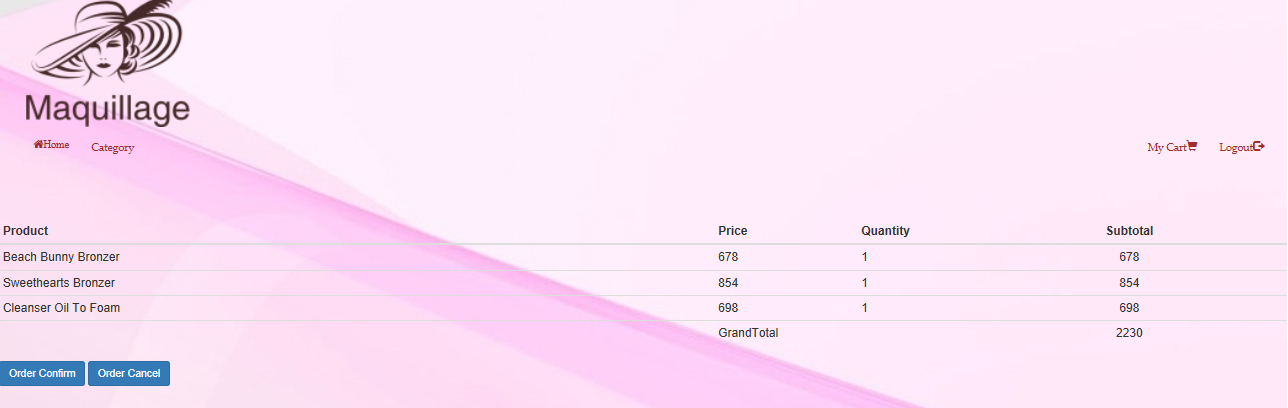
**My cart page:**



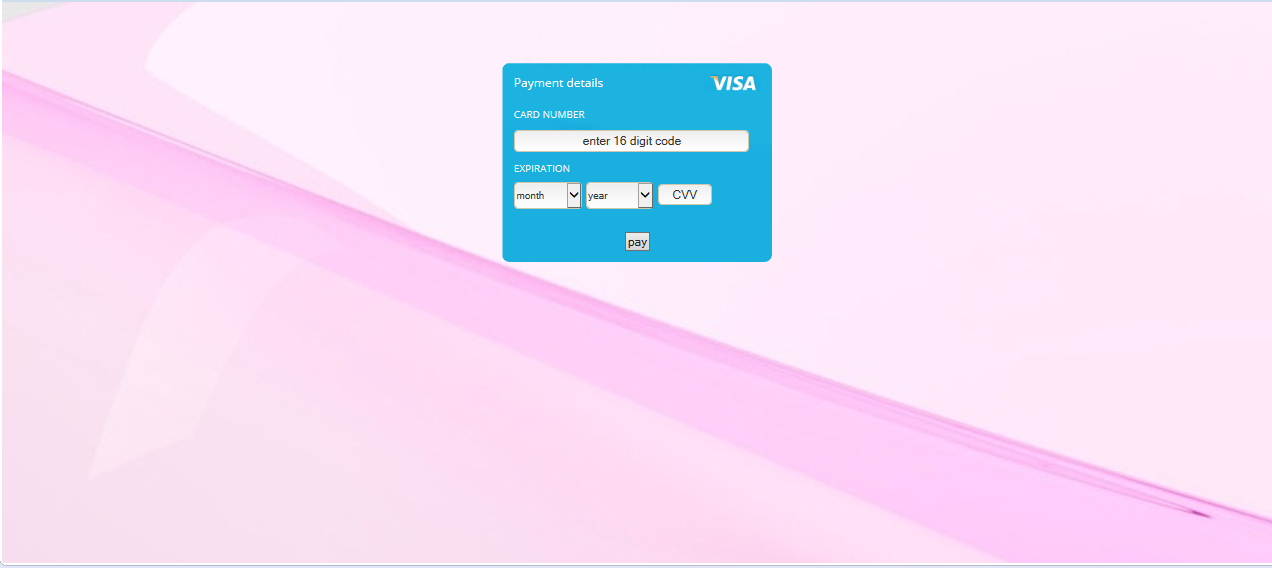
**Billing address page**:



**Order details page:**



**Payment page:**



**Success page**