PROJECT ON ECOMMERCE PORTAL FOE ELECTRICALS

Developed by

Name:K.A.Pavithran Reg.No:R20117460057

NIIT

NIIT

CERTIFICATE

This is to certify that the report titled, ***EShopping ELECTRICALS****, embodies the original work done by* ***PAVITHRAN K.A.(R201174600578)*** *in partial fulfillment of his course requirement at NIIT.*

PROJECT TITLE

Ecommerce portal for ELECTRICALS

NAME : K.A.Pavithran

STUDENT ID : S201174600211

REG NO : *R201174600578*

COORDINATOR : MRS.Indumathi

Start Date :07/07/2019

End Date :23/02/2020

Git hub link : https://github.com/pavithrankumar1999/ecccc

ACKNOWLEDGEMENT

I would like to express my thanks of gratitude to my mentor “Mrs.Indumathi” For their able guidance and support in completing my project.

TABLE OF CONTENTS

Abstract

Retail via internet has changed the way internet is being used over the past decade. Buying and selling stuff, sitting right in front of the computer screen has made even the laziest of shoppers to sit upright and get involved in the activity. All things needed to fulfil day to day requirements are now just a click away. But is the process as easy as it sounds? This mini project aims at creating a seamless, user friendly interface for an online interior décor store. Shopping for products to spruce up the environments at one's home or work place was never easy. Finding them online is even more of a task. This project that uses Java Server Faces as the front end and H2 Database as the backend hopes to create a web application that has templates of work or home spaces from where various products could be brought. For those who are thinking of renovating their abode, rooms with colour themes will be available as choice. All details of the product are provided on just hovering of the mouse and ordering the stuff is also not as pervasive as the current model.

CHAPTER 1

INTRODUCTION

E-commerce is the most visible business use of the World Wide Web. The primary goal of an e-commerce site is to sell goods and services online.

Online shopping is the process whereby consumers directly buy goods, services, etc. from a seller interactively in real-time without an intermediary service over the internet.

Since the emergence of the World Wide Web, merchants have sought to sell their products to people who surf on the internet. Shoppers can visit web stores from the comfort of their homes and shop by sitting in front of the computer. Consumers buy a variety of items from online stores. In fact, people can purchase just about anything from companies that provide their products online. Books, clothing, household appliances, toys, hardware, software and health insurance are some of the hundreds of products consumers can buy from online shopping stores.

Many people choose to shop online for convenience. For example when a person wants to shop at a brick-and-mortar store, she has to drive there, find a parking place and walk throughout the store to locate the product she needs. After finding the items she wants to purchase, she may often need to stand in long lines at the cash register.

This project tries to eliminate even the search for a product in the endless possibilities. The design of the application is in such a way that the user has a more clearer idea of what he wants to purchase from the store. The color coded themes provide the perfect platform for those in the lookout for renovation ideas for their home or workspace.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

In the existing system, the shopping procedure is kind of tedious and does not help users select that perfect set of crockery for the kitchen or the exact type of furniture they would like for the living room, keeping in regard the color and environment of the space they wish to decorate.

2.1.1DISADVANTAGES:

The existing system makes use of a grid architecture for showcasing the products available which makes it difficult for the consumer to decide if a particular product will go well with the surroundings of their renovation space.

2.2 PROPOSED SYSTEM

The proposed system, aims at filling that gap specified earlier and provide the user with an interactive interface that would make things even better. The system would have fully designed color themes for renovation spaces and the usual grid system for displaying products for sale has been done away with. The consumer can hover over any product in any color theme and buy the product which will prove to be more interactive and easier in-decision and buying matters.

CHAPTER 3

SYSTEM REQUIREMENTS

- Processor: Intel i5 processor or more
- 256 GB hard disk space or more
- 4GB RAM or more
- Keyboard
- Mouse

- Operating System:

  - Windows 10 Operating systems

- Software:

  - Eclipse Mars: Eclipse Java EE IDE for Web Developers.Version:Mars.2 Release(4.5.2)
  - Apache Tomcat 9.0.30
  - Oracle Java SDK 8: JDK 1.8.0_73
  - Notepad++ or Atom Text editor.
  - Maven 3 : Embedded 3.3.3
  - H2 Database installer.(Version 1.4.199)
  - Pdf Reader
  - Chrome Browser.(Version 70.0. or above)

ECLIPSE IDE:

- **Eclipse** is an integrated development environment (IDE) used in computer programming.[6] It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins,
  including Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, JavaScript, Julia,[7] Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Rust, Scala, and Scheme. It can also be used to

develop documents with LaTeX (via a TeXlipse plug-in) and packages for the software Mathematica.

- Eclipse software development kit (SDK) is free and open-source software, released under the terms of the Eclipse Public License, although it is incompatible with the GNU General Public License.[10] It was one of the first IDEs to run under GNU Classpath and it runs without problems under IcedTea.

APACHE TOMCAT:

- **Apache Tomcat** (sometimes simply "Tomcat") is an open-source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and WebSocket technologies.[3] Tomcat provides a "pure Java" HTTP web server environment in which Java code can run.
- Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under the Apache License 2.0 license.

ORACLE Java SDK:

- The **Java Development Kit** (**JDK**) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms[1] released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, macOS or Windows. The JDK includes a private JVM and a few other resources to finish the development of a Java Application.[2] Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that they would release it under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the OpenJDK

MAVEN:

- Maven is a build automation tool used primarily for Java projects. 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 addresses two aspects of building software: how software is built, and its dependencies. Unlike earlier 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.[2] This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

H2DATABASE:

- H2 is a relational database management system written in Java. It can be embedded in Java applications or run in client-server mode.[1]

CHAPTER 4

SYSTEM DESIGN

4.1 ARCHITECTURE OF THE SYSTEM

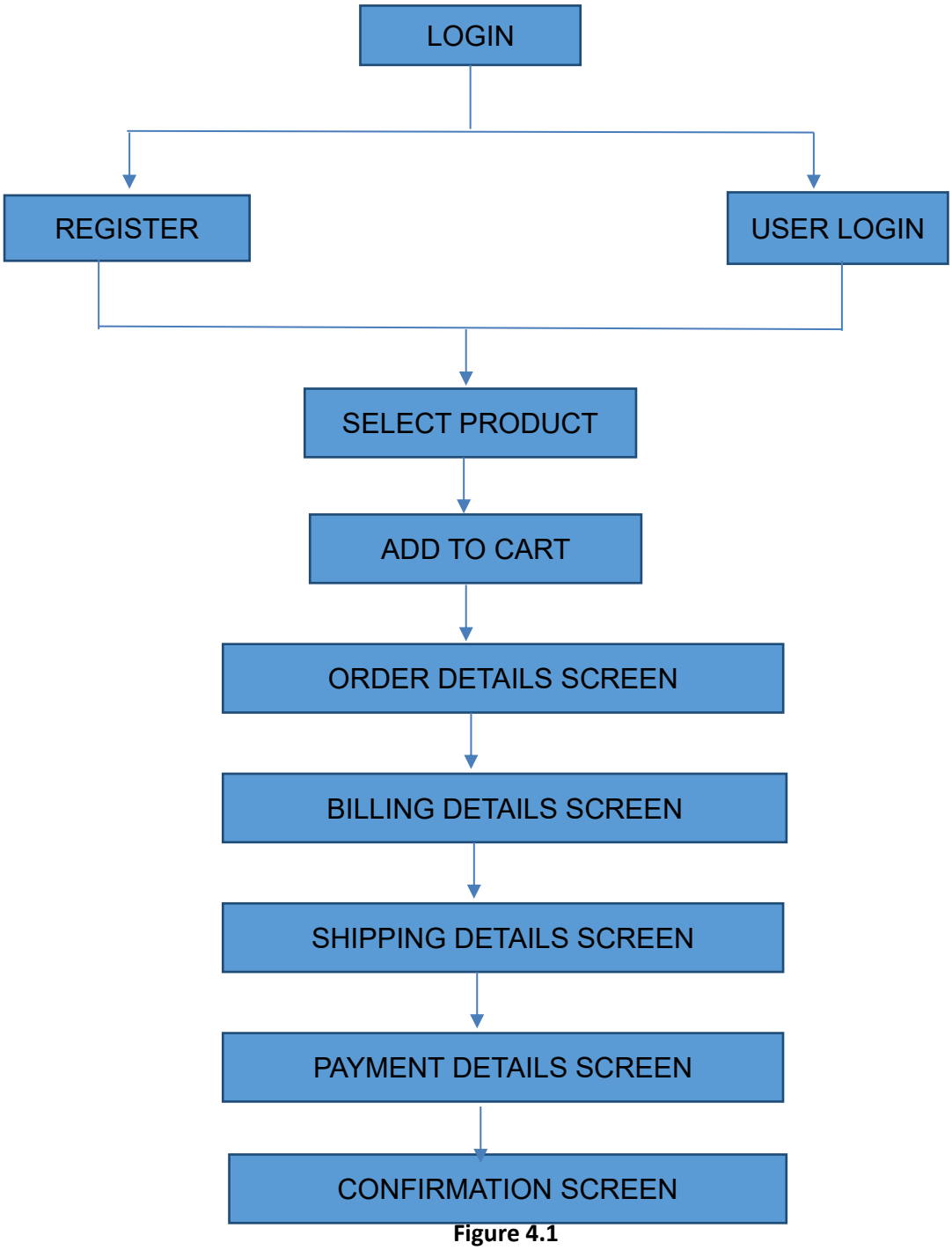


Figure 4.1

4.2 MODULE DESCRIPTION

In this project there are three modules, namely

1. Generating the login and other pages.
2. Generating the order summary screens.
3. Connecting the database and retrieve information.

Each of the modules is described in detail as follows.

The first module of the project involves generating the login and home screens using JSF and RichFaces Library. The Apache Tomcat server has to be setup before any of the above is done since the server is the only way the Java EE application can be run successfully. The various color themed pages are also designed along with navigation rules are set.

The following module involves the post order placement phase. The various screens such as Order summary, shipping, billing and confirmation details are brought about in the form of modal panels with proceed or shop further options.

The final phase involves the most important part of the web application development which is connecting the database to the application screens. Retrieval of data from the MySQL workbench using java is implemented by creating DAO and JavaBean classes. The JavaBean class is the one that is accessible by the screen of the web application. The DAO class is the one that connects the bean and the database.

The queries to fetch data from the database is specified in the DAO class and various commands to switch on the connection, etc., are specified in the classes.

4.3 SOFTWARE MODELING.

SEQUENCE DIAGRAM:

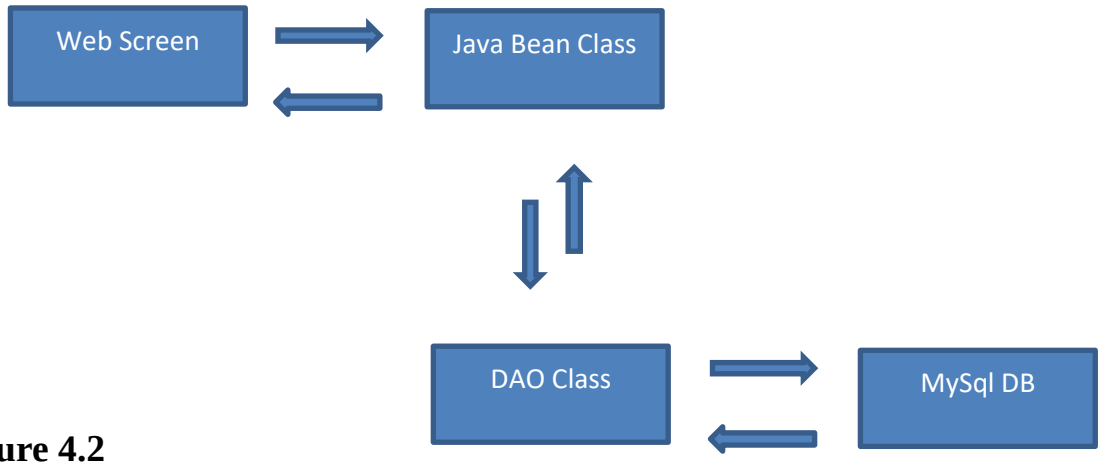


Figure 4.2

USE CASE DIAGRAM:

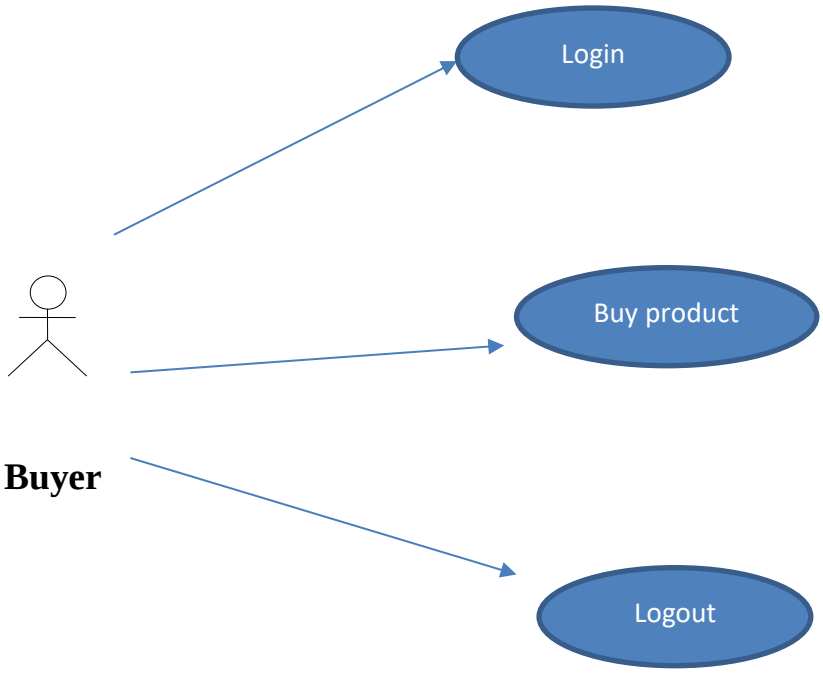


Figure 4.3

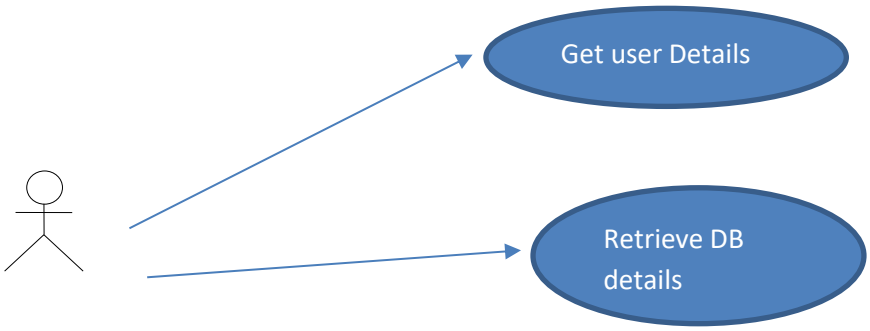


Figure 4.4

CHAPTER 5

SYSTEM IMPLEMENTATION

This project needs a java development kit (JaveSE1.8 and above). Also web application development environments such as Eclipse Luna or Galileo is required to generate the project. Apache Tomcat is a server on which the code can be run and RichFaces Library is required for smooth code development. MySQL connector BAT files and MySQL server is also required for data management.

- FRAMEWORKS AND TECHNOLOGY USED:
  - HIBERNATE
  - SPRING FRAMEWORK
  - BOOTSRAP
  - GITHUB
  - SERVLET
- 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). Unfamiliar with the notion of ORM?

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

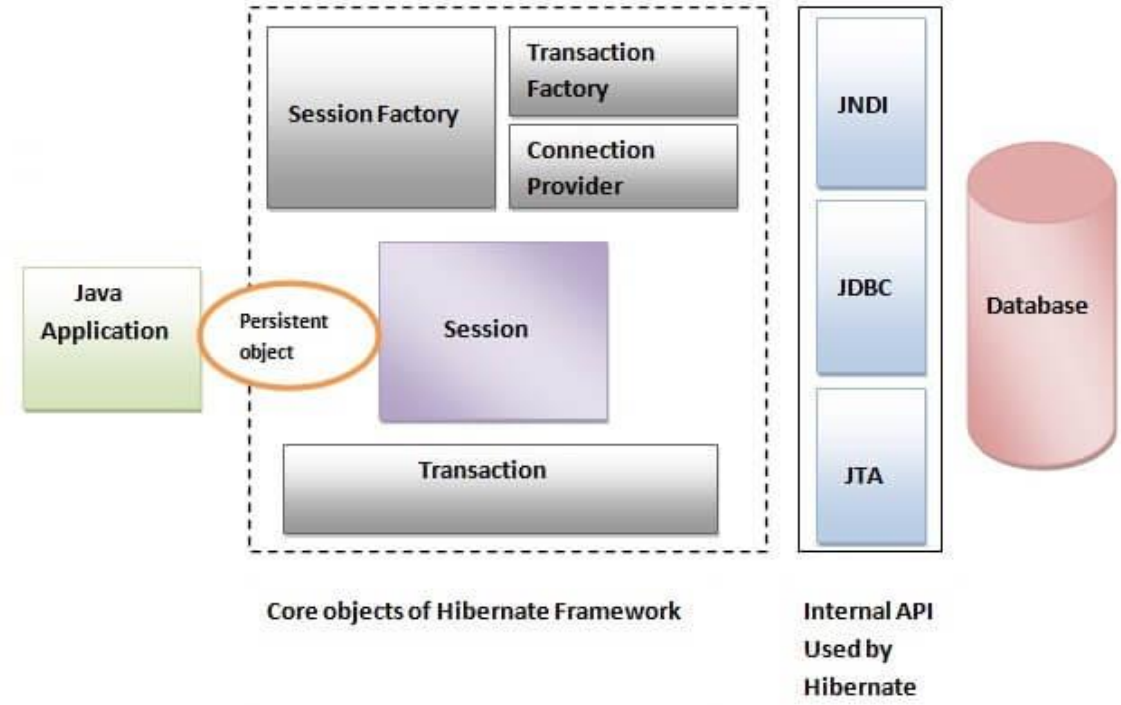
Hibernate was designed to work in an application server cluster and deliver a highly scalable architecture. Hibernate scales well in any environment: Use it to drive your in-house Intranet that serves hundreds of users or for mission-critical applications that serve hundreds of thousands.

Reliable

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

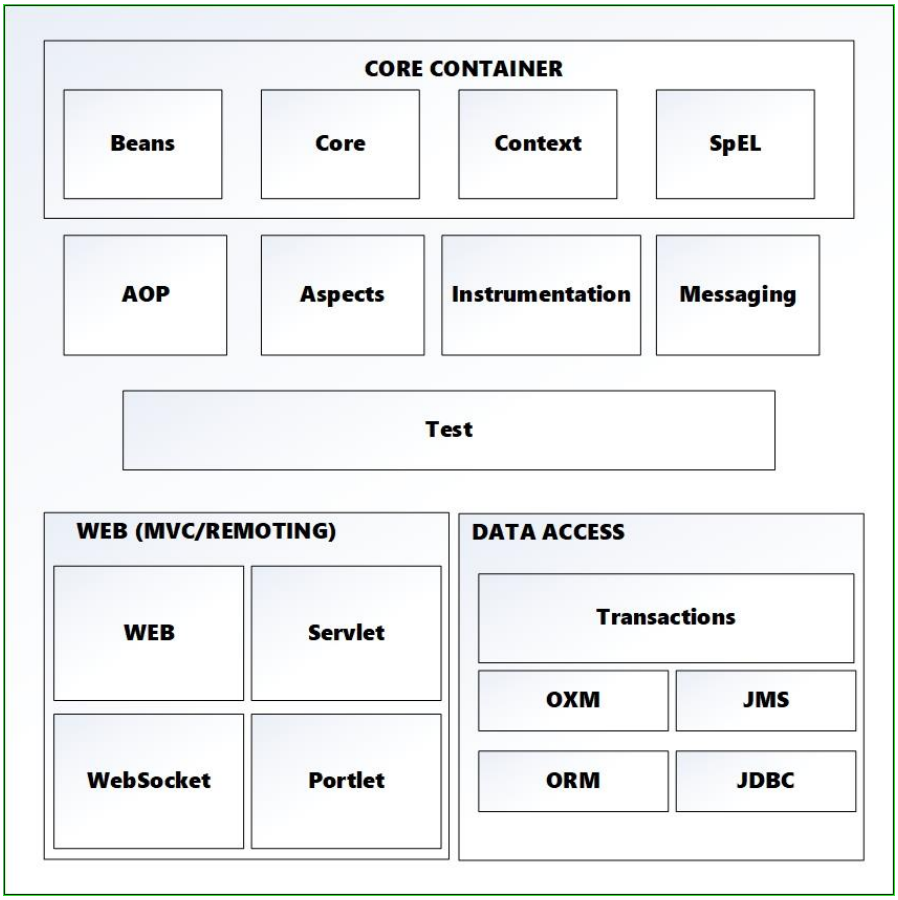
Extensibility

Hibernate is highly configurable and extensible.



- SPRING :

  The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.



- GITHUB:

- Git is an open-source version control system that was started by Linus Torvalds—the same person who created Linux. Git is similar to other version control systems—Subversion, CVS, and Mercurial to name a few.

- So, Git is a version control system, but what does that mean? When developers create something (an app, for example), they make constant changes to the code, releasing new versions up to and after the first official (non-beta) release.

- Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

- SERVLET(dispatcher-servlet):
    - Dispatcher-servlet is a xml(Xentensible Markup Language) which handles the request and response.
    - CODE:

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xmlns:webflow-config="http://www.springframework.org/schema/webflow-config"
	xsi:schemaLocation="http://www.springframework.org/schema/beans

	http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
                                        http://www.springframework.org/schema/mvc
                                http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
                                http://www.springframework.org/schema/tx
                                        http://www.springframework.org/schema/tx/spring-
tx-4.0.xsd
                                         http://www.springframework.org/schema/webflow-
config
                http://www.springframework.org/schema/webflow-config/spring-webflow-config.xsd
                                        http://www.springframework.org/schema/context

	http://www.springframework.org/schema/context/spring-context-4.0.xsd">

	<bean id="multipartResolver"
		class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
		<property name="maxUploadSize" value="4200000" />
	</bean>

	<context:component-scan base-package="com.niit.*" />
	<tx:annotation-driven />

	<bean
		class="org.springframework.web.servlet.view.InternalResourceViewResolver">

		<property name="prefix">
			<value>/WEB-INF/views/</value>
```

```
            </property>
            <property name="suffix">
                  <value>.jsp</value>
            </property>
      </bean>
      <mvc:resources mapping="/resources/**" location="/WEB-INF/resources/"
            cache-period="31556926" />
      <mvc:annotation-driven />
      <mvc:default-servlet-handler />

</beans>
```

The program contains only the user side where the user can select any product that he intends to buy which lead to the generation of post order summary screens.

CHAPTER 6

SYSTEM TESTING

The testing of a conventional software system involves some of the following phases. They are

- Unit Testing

Unit Testing:

A software module can be created by building up of many small parts into a single module. This small part is called as a unit. A unit is a piece of code that will perform a specific task. At the end of this testing all units will be tested so that we can get the correct result. By using unit testing we can easily identify the errors.

CHAPTER 7

CONCLUSION

This the project provides a user interactive mechanism for a simpler online shopping experience using Java Server Pages Library as the main softwares being used. Thus the user experience of using the net to buy products for sprucing up their home or workspace is made much more easier and friendly.

CHAPTER 8

APPENDIX

PROJECT CODE:

HOME:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<!--   <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
-->
  <style>
  .container {
  padding-left:0px;
  padding-right:0px;
  }
  .carousel-inner{
  width:100%;
  max-height: 450px !important;
}
  .carousel-inner > .item > img,
  .carousel-inner > .item > a > img {
      width: 100%;
            width:100%;
       max-height: 450px;
      margin: auto;
  }

  .fnt{
   font-family: "Times New Roman", Times, serif;
  }
  </style>
</head>
<body>

<div class="container" style="width:100%; height:450px">
  <br>
  <div id="myCarousel" class="carousel slide" data-ride="carousel">
```

```
    <!-- Indicators -->
    <ol class="carousel-indicators">
      <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
      <li data-target="#myCarousel" data-slide-to="1"></li>
      <li data-target="#myCarousel" data-slide-to="2"></li>
    </ol>

    <!-- Wrapper for slides -->
    <div class="carousel-inner" role="listbox">
      <div class="item active">
        <img src="resources/carousel/apple.jpg" style="width:100%; height:450px" >
     <div class="fnt"> <h2><font color="#ce3175"></font></h2>
            <h3><font color="#ce3175"></font></h3>
         </div>
            </div>

      <div class="item">
        <img src="resources/carousel/oneplus.png" style="width:100%; height:450px" >
                  <div class="fnt"><h2><font color="#ce3175">Collection of</font></h2>
                  <h3><font color="#ce3175">Luxury Watches</font></h3>
      </div>
      </div>

      <div class="item">
        <img src="resources/carousel/a5s.jpg" style="width:100%; height:450px">
      <div class="fnt"><h2><font color="#ce3175">Classic</font></h2>
            <h3><font color="#ce3175">Wall Clock</font></h3>
            </div>
            </div>


    <!-- Left and right controls -->
    <a class="left carousel-control" href="#myCarousel" role="button" data-slide="prev">
      <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
      <span class="sr-only">Previous</span>
    </a>
    <a class="right carousel-control" href="#myCarousel" role="button" data-slide="next">
      <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
      <span class="sr-only">Next</span>
    </a>
  </div>
  </div>
</div>

</body>
</html>
```

JAVA BEAN:

```
import java.util.List;

import org.hibernate.Criteria;

import org.hibernate.Query;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import com.niit.dao.CategoryDAO;

import com.niit.model.Category;

@Repository("CategoryDAO")

@Transactional

public class CategoryDAOImpl implements CategoryDAO {

        @Autowired

        private SessionFactory sessionFactory;

        public CategoryDAOImpl(SessionFactory sessionFactory) {
```

```
                this.sessionFactory = sessionFactory;}

public List<Category> list() {

                @SuppressWarnings({ "unchecked" })

                List<Category> listCategory = (List<Category>)

sessionFactory.getCurrentSession().createCriteria(Category.class)

        setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY).list();

        return listCategory;

}

public Category getByCategoryId(int categoryid) {

        String hql = "from Category where CategoryId ='" + categoryid + "'";

        Query query = (Query) sessionFactory.getCurrentSession().createQuery(hql);

        @SuppressWarnings("unchecked")

        List<Category> listCategory = (List<Category>) (query).list();

        if (listCategory != null && !listCategory.isEmpty()) {

                        return listCategory.get(0);

                }

                return null;
```

```
        }

public Category getByCategoryName(String categoryname) {

                String hql = "from Category where CategoryName ='" + categoryname + "'";

                Query query = (Query) sessionFactory.getCurrentSession().createQuery(hql);

                @SuppressWarnings("unchecked")

                List<Category> listCategory = (List<Category>) (query).list();

        if (listCategory != null && !listCategory.isEmpty()) {

                        return listCategory.get(0);

                }

                return null;

        }

        public void saveOrUpdate(Category category) {

                sessionFactory.getCurrentSession().saveOrUpdate(category);

        }

        public void delete(int categoryId) {

                Category categoryToDelete = new Category();

                categoryToDelete.setCategoryId(categoryId);
```

```
                sessionFactory.getCurrentSession().delete(categoryToDelete);
        }
}
```

JAVA DAO:

```
package com.niit.dao;

import java.util.List;

import com.niit.model.Category;

public interface CategoryDAO {

        public List<Category> list();

        public Category getByCategoryId(int categoryid);

        public Category getByCategoryName(String categoryname);

        public void saveOrUpdate(Category category);

        public void delete(int category_id);

}
```

JAVA MODEL CLASS:

```
package com.niit.model;

import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

import org.springframework.stereotype.Component;

@Table(name = "category")

@Entity

@Component

public class Category {

	@Id

	@GeneratedValue

	private int categoryId;

	private String categoryName;

	private String categoryDescription;

	public int getCategoryId() {

		return categoryId;

	}

	public void setCategoryId(int categoryId) {

		this.categoryId = categoryId;
```

```
    }

    public String getCategoryName() {

        return categoryName;

    }

    public void setCategoryName(String categoryName) {

        this.categoryName = categoryName;

    }

    public String getCategoryDescription() {

        return categoryDescription;

    }

    public void setCategoryDescription(String categoryDescription) {

        this.categoryDescription = categoryDescription;

    }

}
```

TEST CASE:

```
package com.niit.Test;

import org.springframework.context.annotation.
AnnotationConfigApplicationContext;

import com.niit.dao.BillingaddressDAO;

import com.niit.dao.CartDAO;

import com.niit.dao.CategoryDAO;

import com.niit.dao.ProductDAO;

import com.niit.dao.RoleDAO;

import com.niit.dao.ShippingaddressDAO;

import com.niit.dao.SupplierDAO;

import com.niit.dao.UserDAO;

import com.niit.model.Billingaddress;

import com.niit.model.Cart;

import com.niit.model.Category;

import com.niit.model.Product;

import com.niit.model.Role;

import com.niit.model.Shippingaddress;
```

```
import com.niit.model.Supplier;

import com.niit.model.User;

public class Test {

public static void main(String[] args) {

        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();

                context.scan("com.niit.*");

                context.refresh();

                CategoryDAO categoryDAO = (CategoryDAO)context.getBean("CategoryDAO");

                UserDAO UserDao = (UserDAO) context.getBean("UserDAO");

                ProductDAO ProductDao = (ProductDAO) context.getBean("ProductDAO");

                BillingaddressDAO BillingaddressDao = (BillingaddressDAO)
context.getBean("BillingaddressDAO");

                CartDAO CartDao = (CartDAO) context.getBean("CartDAO");

                SupplierDAO SupplierDao = (SupplierDAO) context.getBean("SupplierDAO");

                ShippingaddressDAO ShippingaddressDao = (ShippingaddressDAO)
context.getBean("ShippingaddressDAO");

                RoleDAO RoleDao = (RoleDAO) context.getBean("RoleDAO");

                Category category = (Category) context.getBean("category");
```

```
User user = (User) context.getBean("user");

Product product = (Product) context.getBean("product");

Billingaddress billingaddress = (Billingaddress) context.getBean("billingaddress");

Cart cart = (Cart) context.getBean("cart");

Supplier supplier = (Supplier) context.getBean("supplier");

Shippingaddress shippingaddress = (Shippingaddress) context.getBean("shippingaddress");

Role role = (Role) context.getBean("role");

category.setCategoryName("Watches");

categoryDAO.saveOrUpdate(category);

user.setUserName("skfr");

user.setEmailId("jhvd@gmail.com");

user.setPassword("jfkd");

user.setContactNumber("98765");

user.setAddress("s12,sejkajdh");

user.setZipcode(75426);

role.setUserName("kjgdffd");

role.setEmailId("hgfshdghj");

role.setContactNumber("6567");
```

```
user.setRole(role);

role.setUser(user);

role.setUserName("abcd@gmail.com");

role.setEmailId("4567");

role.setContactNumber("656756788");

user.setRole(role);

role.setUser(user);

UserDao.saveOrUpdate(user);

RoleDao.saveOrUpdate(role);

product.setProductName("fdknmfg");

ProductDao.saveOrUpdate(product);

billingaddress.setAddress("hfdjdufuinv");

billingaddress.setContactNumber("897465");

BillingaddressDao.saveOrUpdate(billingaddress);

cart.setProductName("dfhjj");

CartDao.saveOrUpdate(cart);

supplier.setSupplierName("hddsjk");

supplier.setContactNumber(87968764);
```

```
                SupplierDao.saveOrUpdate(supplier);

                shippingaddress.setAddress("fdgfs");

                shippingaddress.setUserName("jkfgjd");

                ShippingaddressDao.saveOrUpdate(shippingaddress);

        }

}
```

H2DATABASE CONFIGURATION:

```
package com.niit.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.orm.hibernate4.HibernateTransactionManager;

import org.springframework.orm.hibernate4.LocalSessionFactoryBuilder;
```

```
import org.springframework.transaction.annotation.
EnableTransactionManagement;

import com.niit.dao.BillingaddressDAO;

import com.niit.dao.CartDAO;

import com.niit.dao.CategoryDAO;

import com.niit.dao.ProductDAO;

import com.niit.dao.RoleDAO;

import com.niit.dao.ShipmentDAO;

import com.niit.dao.SupplierDAO;

import com.niit.dao.UserDAO;

import com.niit.daoImpl.BillingaddressDAOImpl;

import com.niit.daoImpl.CartDAOImpl;

import com.niit.daoImpl.CategoryDAOImpl;

import com.niit.daoImpl.ProductDAOImpl;

import com.niit.daoImpl.RoleDAOImpl;

import com.niit.daoImpl.ShipmentDAOImpl;

import com.niit.daoImpl.SupplierDAOImpl;

import com.niit.daoImpl.UserDAOImpl;
```

```
import com.niit.model.Billingaddress;

import com.niit.model.Cart;

import com.niit.model.Category;

import com.niit.model.Product;

import com.niit.model.Role;

import com.niit.model.Shipment;

import com.niit.model.Supplier;

import com.niit.model.User;

@Configuration

@ComponentScan("com.niit.*")

@EnableTransactionManagement

public class hiberConfig {

        @Bean(name = "dataSource")

        public DataSource getH2DataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

        dataSource.setDriverClassName("org.h2.Driver");

        dataSource.setUrl("jdbc:h2:tcp://localhost/~/test");

        dataSource.setUsername("sa");
```

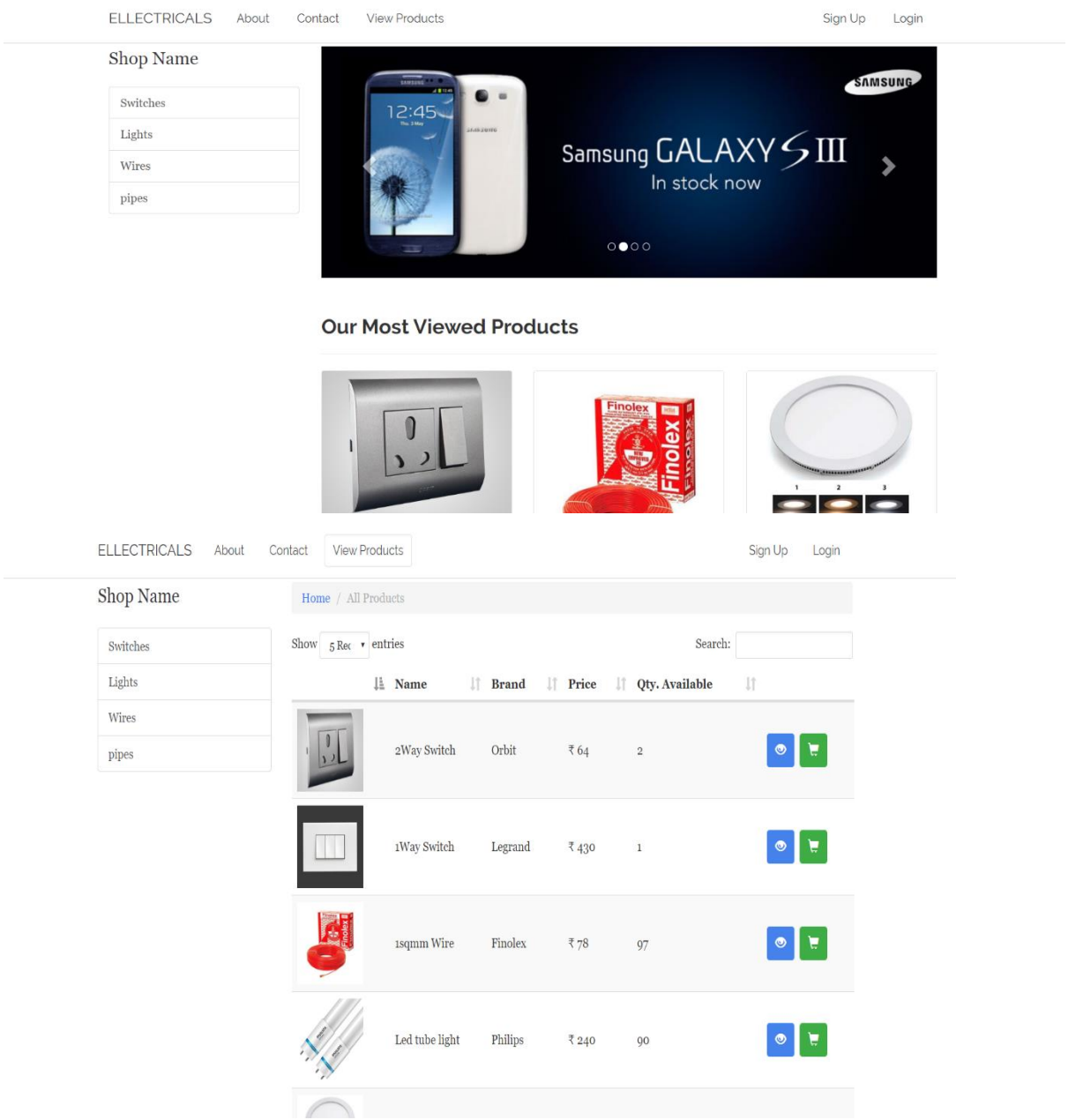
```
dataSource.setPassword("sa");

return dataSource;

}

private Properties getHibernateProperties() {

        Properties properties = new Properties();

        properties.put("hibernate.show_sql", "true");

        properties.put("hibernate.hbm2ddl.auto", "update");

        properties.put("hibernate.dialect", "org.hibernate.dialect.H2Dialect");

        properties.put("hibernate.format_sql", "true");

        return properties;

}

@Autowired

@Bean(name = "sessionFactory")

public SessionFactory getSessionFactory(DataSource dataSource) {

LocalSessionFactoryBuilder sessionBuilder = new LocalSessionFactoryBuilder(dataSource);

sessionBuilder.addProperties(getHibernateProperties());

sessionBuilder.addAnnotatedClass(Category.class);

sessionBuilder.addAnnotatedClass(User.class);
```

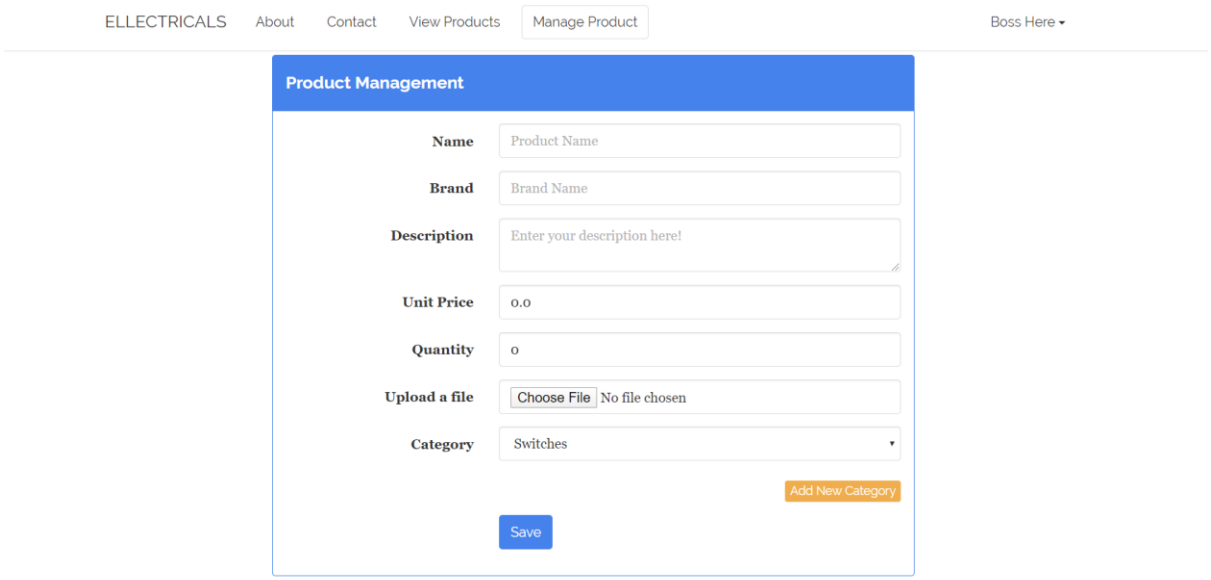
```
sessionBuilder.addAnnotatedClass(Product.class);

sessionBuilder.addAnnotatedClass(Billingaddress.class);

sessionBuilder.addAnnotatedClass(Cart.class);

sessionBuilder.addAnnotatedClass(Supplier.class);

sessionBuilder.addAnnotatedClass(Shipment.class);

sessionBuilder.addAnnotatedClass(Role.class);

return sessionBuilder.buildSessionFactory();

}

@Autowired

@Bean(name = "transactionManager")

public HibernateTransactionManager getTransactionManager(SessionFactory sessionFactory) {

HibernateTransactionManager transactionManager = new
HibernateTransactionManager(sessionFactory);

        return transactionManager;

}

@Autowired(required = true)

@Bean(name = "CategoryDAO")

public CategoryDAO getCategoryDAO(SessionFactory sessionFactory) {
```

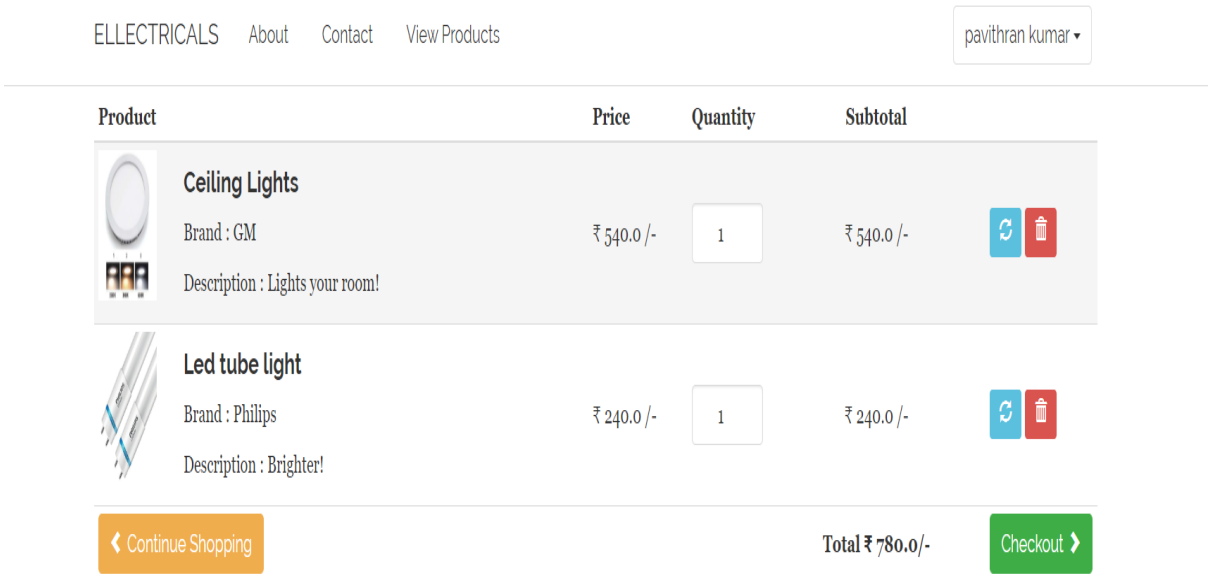
```
        return new CategoryDAOImpl(sessionFactory);
}

@Autowired(required = true)
@Bean(name = "UserDAO")
public UserDAO getUserDAO(SessionFactory sessionFactory) {
        return new UserDAOImpl(sessionFactory);
}
@Autowired(required = true)
@Bean(name = "ProductDAO")
public ProductDAO getProductDAO(SessionFactory sessionFactory) {
        return new ProductDAOImpl(sessionFactory);
}
@Autowired(required = true)
@Bean(name = "BillingaddressDAO")
public BillingaddressDAO getBillingaddresstDAO(SessionFactory sessionFactory) {
        return new BillingaddressDAOImpl(sessionFactory);
}
```

```
@Autowired(required = true)

@Bean(name = "CartDAO")

public CartDAO getCartDAO(SessionFactory sessionFactory) {

        return new CartDAOImpl(sessionFactory);

}

@Autowired(required = true)

@Bean(name = "SupplierDAO")

public SupplierDAO getSupplierDAO(SessionFactory sessionFactory) {

        return new SupplierDAOImpl(sessionFactory);

}

@Autowired(required = true)

@Bean(name = "ShipmentDAO")

public ShipmentDAO getShipmentDAO(SessionFactory sessionFactory) {

        return new ShipmentDAOImpl(sessionFactory);

}

@Autowired(required = true)

@Bean(name = "RoleDAO")

public RoleDAO getRoleDAO(SessionFactory sessionFactory) }
```

SCREENSHOTS:




ELLECTRICALS
About
Contact
View Products
Manage Product
Boss Here
Product Management
Name
Product Name
Brand
Brand Name
Description
Enter your description here!
Unit Price
0.0
Quantity
0
Upload a file
Choose File
No file chosen
Category
Switches
Add New Category
Save


ELLECTRICALS
About
Contact
View Products
pavithran kumar
Product
Price
Quantity
Subtotal
Ceiling Lights
Brand : GM
Description : Lights your room!
₹ 540.0 /-
1
₹ 540.0 /-
Led tube light
Brand : Philips
Description : Brighter!
₹ 240.0 /-
1
₹ 240.0 /-
Continue Shopping
Total ₹ 780.0/-
Checkout

ELECTRICALS

Ceiling Lights

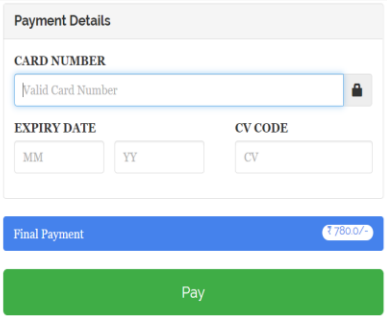
Quantity -1
Buying Price - ₹ 540.0/-

Grand Total - ₹ 540.0/-

Led tube light

Quantity -1
Buying Price - ₹ 240.0/-

Grand Total - ₹ 240.0/-



Online Shopping

Your Order is Confirmed!!

Invoice

Order # 29

Billed To:
pavithran kumar
50,vishnu nagar,krishna nagar 1st main road
west tambaram
chenna - 600045
tamilnadu - India

Shipped To:
pavithran kumar
50,vishnu nagar,krishna nagar 1st main road
west tambaram
chenna - 600045
tamilnadu - India

Payment Method:
Card Payment
pavithrankumar1999@gmail.com

Order Date:
Sun Mar 01 08:48:13 IST 2020

Order summary

Item	Price	Quantity	Totals
Ceiling Lights	₹ 540.0	1	₹ 540.0
Led tube light	₹ 240.0	1	₹ 240.0

Grand Total ₹ 780.0

CHAPTER 9

REFERENCES

- *www.wikipedia.com*
- *http://forum.java.sun.com*
- *http://www.javaworld.com*