Introduction

Canteen Management System is a web-based application that will be able to handle all the activities of the canteen. We need a system to handle products information, billing information, customers details and etc. At the same time, it should be secured.

This is a web-based application develop to replace the manual process of handling the canteen-related activity. The basic idea here is to allow users to add products, add a company of products, generate sales, and etc.

Our canteen management system provides a friendly User Interface for

numerous food outlets, menu design, billing features and lots

more.Implementation of such a system makes the operation of the kitchen and

the whole of the canteen as effective and quick as possible.

**PROBLEM DEFINITION**

These days the world has become a digital world where everything is available in a single click or touch.

The definition of our problem lies in manual system and a fully automated system of Canteen management system.

**Manual system:**

In Manual system is more prone to errors and sometimes it encounters various problems which are unstructured. Because things are managing by the human on the paper there might high chances to get mistakes as well as its time consuming and high money consuming.

**Need of Canteen management system**

A few factors that direct us to develop a new system are given below

* Faster System
* Accuracy
* Reliability
* Informative
* Product Delivered.

**OBJECTIVES**

* To provide a bug-free application..
* The main objective is to build a secured, robust Canteen Management System Project where the information of the canteen is managed properly.
* It maintains the record of products, customer billing detail efficiently so that it would be easy to access at any time 24\*7.

**USED TOOLS AND TECHNOLOGIES**

We have a wide range of options of programming languages. From these options we can choose appropriate platform tools, technologies and languages for development of the airline reservation project.

**Some of these are as following**

**Programming Languages**: Java.

**Relational Database:** MYSQL.

**SOFTWARE REQUIREMENTS:**

**Operating system** : Windows Family

Front End : CSS, JSP, HTML

**Back End** : Servlet, JDBC.

Server : Tomcat Server

**HARDWARE SPECIFICATIONS**

**Processor** : (i3) Intel Pentium or more

Ram : 4 GB

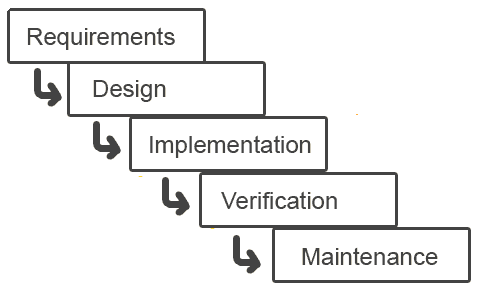
**Hard** **disk** : 16 GB hard disk recommended

**SOFTWARE ENGINEERING APPROACH**

The field of software engineering is related to the development software in systematic manner unlike simple programs which can be developed in isolation and there may not be any systematic approach being followed. there is a big difference between programming and software engineering process. As it provides models that lead to the production of well documented software in a manner that is predictable.

For a planning process, it should be possible to determine in advance the time and effort will be required to develop the final product.

The model I have used is **Waterfall Model or Classic Life Cycle**. In this model first of all the existed system is observed. Then customer requirements are taken in consideration then planning, modeling, construction and finally deployment.



**ANALYSIS & SYSTEM DESIGN**

Requirement Analysis collects data through interviews, questionnaires, on-site observations, and procedural manuals and like. It is required to organize and convert the data through system flowcharts, data flow diagrams, structured decision tables and the like that support future development of the system.

The Data flow diagrams and various processing logic techniques show how, where, and when data are used or changed in an information system, but these techniques do not show the definition, structure and relationships within the data.

It is a way to focus on functions rather than the physical implementation. This is analogous to the architect’s blueprint as a starting point for system design. The design is a solution, a “how to” approach, compared to analysis, a “what is” orientation.

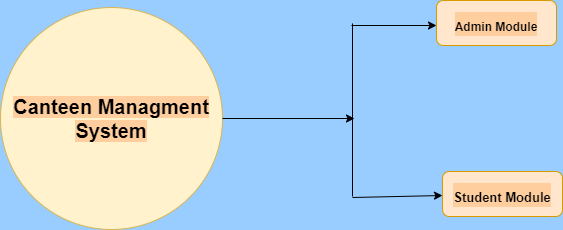
System design is a highly creative process. This system design process is also referred as data modeling. The most common formatted used the E-R notation explains the characteristics and structure of data independent of how the data may be stored in computer memories.

The process of system design can be divided into three stages. They are two

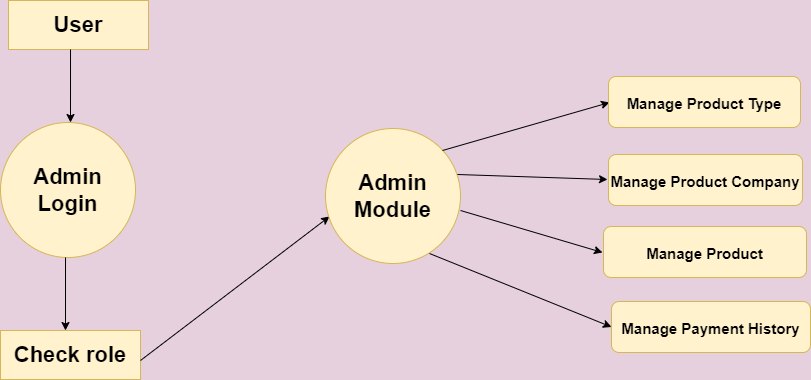
Structure design

* Database design
* Interface design
* As we know that system design is a solution to “How to approach to the creation of new system”. It provides the understudying and procedural details necessary for implementing the system. The steps involved during system design were as follow

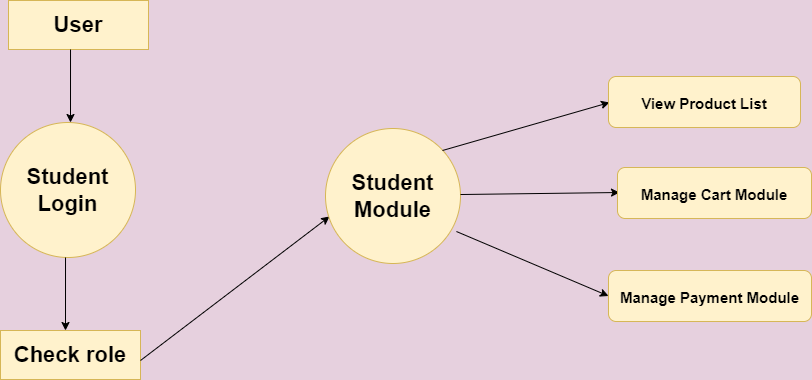
**DFD for Canteen management System**

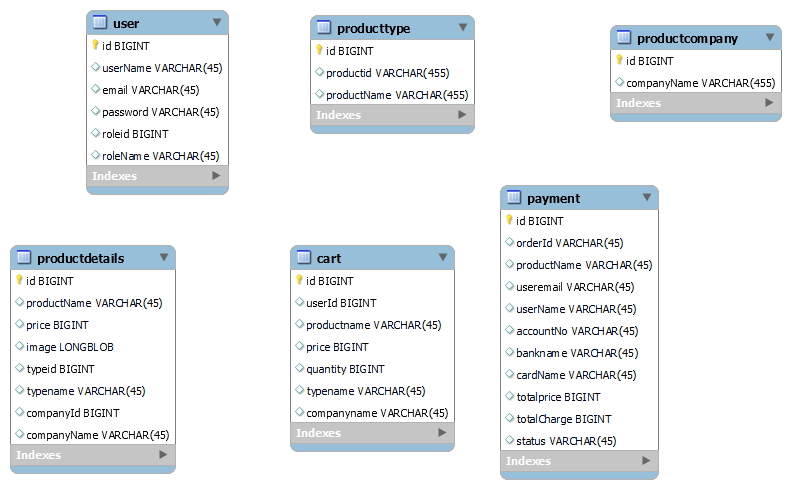


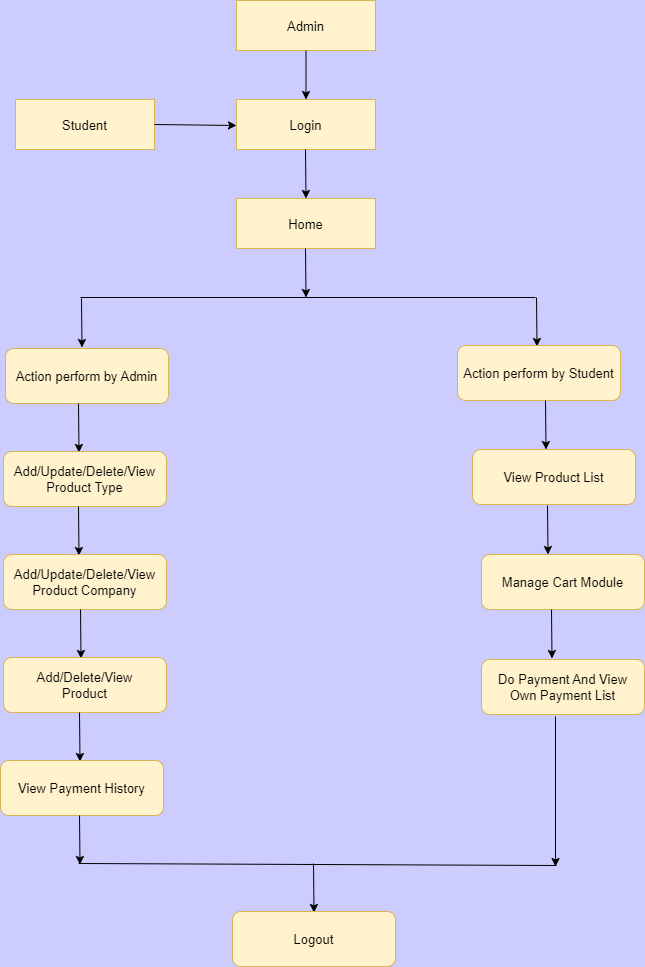
* **First Level of Data Flow Diagram for Admin**

****

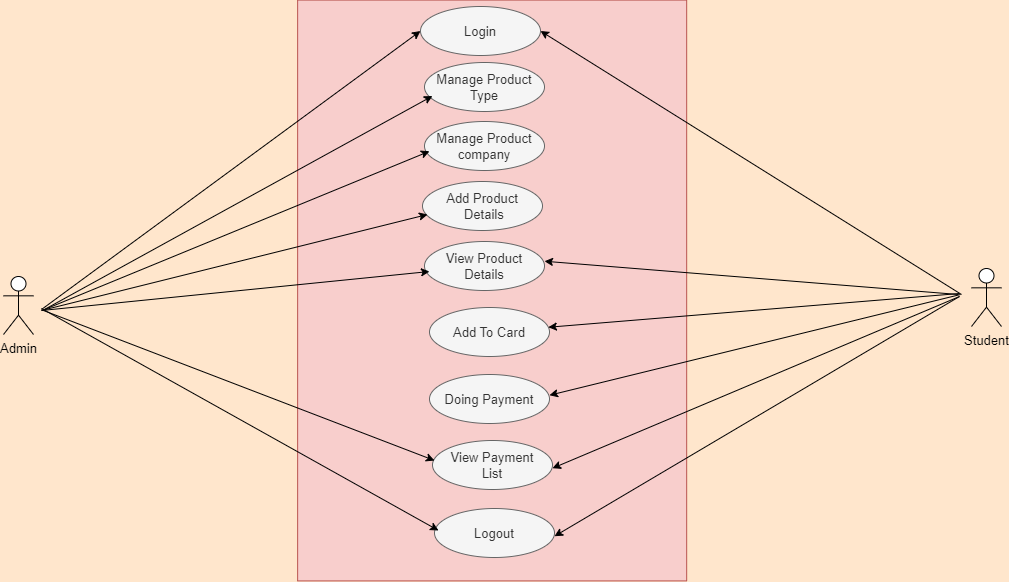
**Second Level of Data Flow Diagram for Client**

****

**ER-Diagram**

**DFD Diagram-**

**UseCase Diagram-**

****

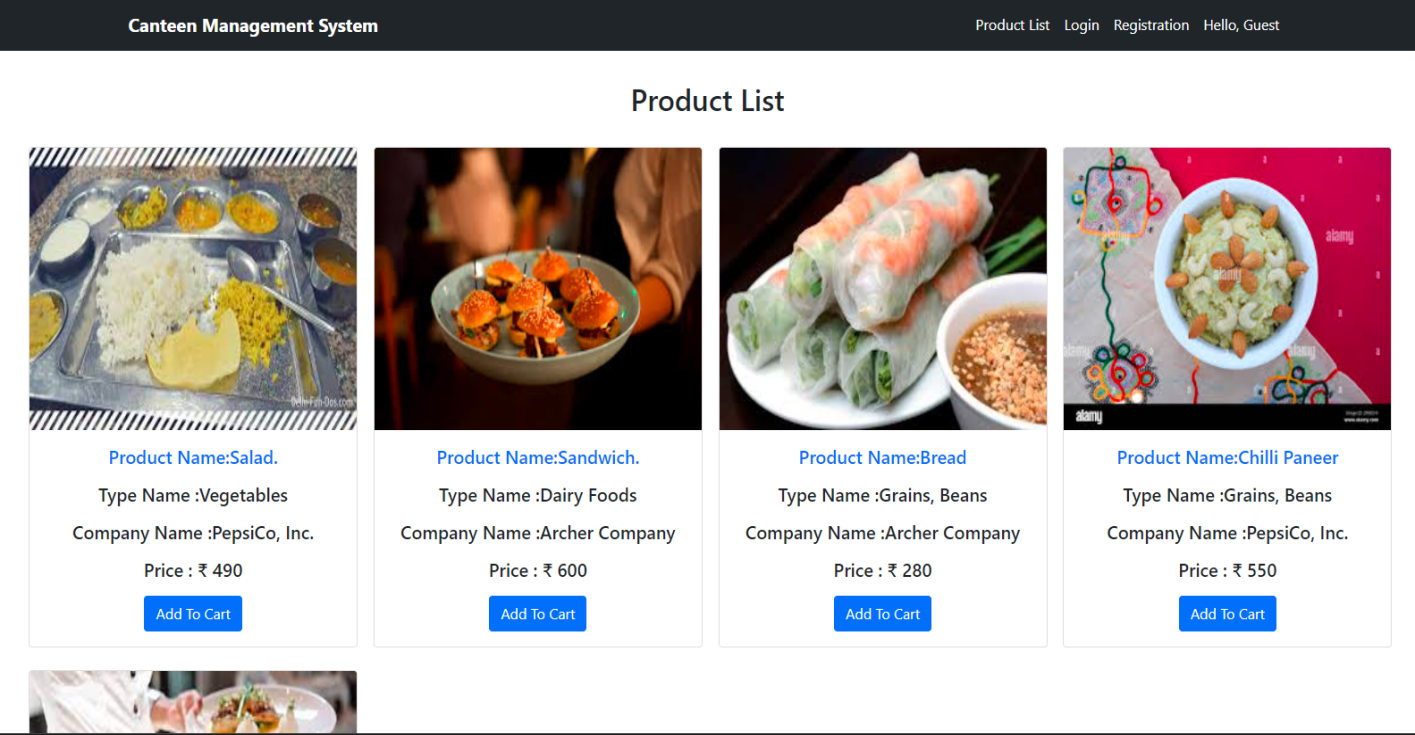
**LOGICAL AND PHYSICAL DESIGN**

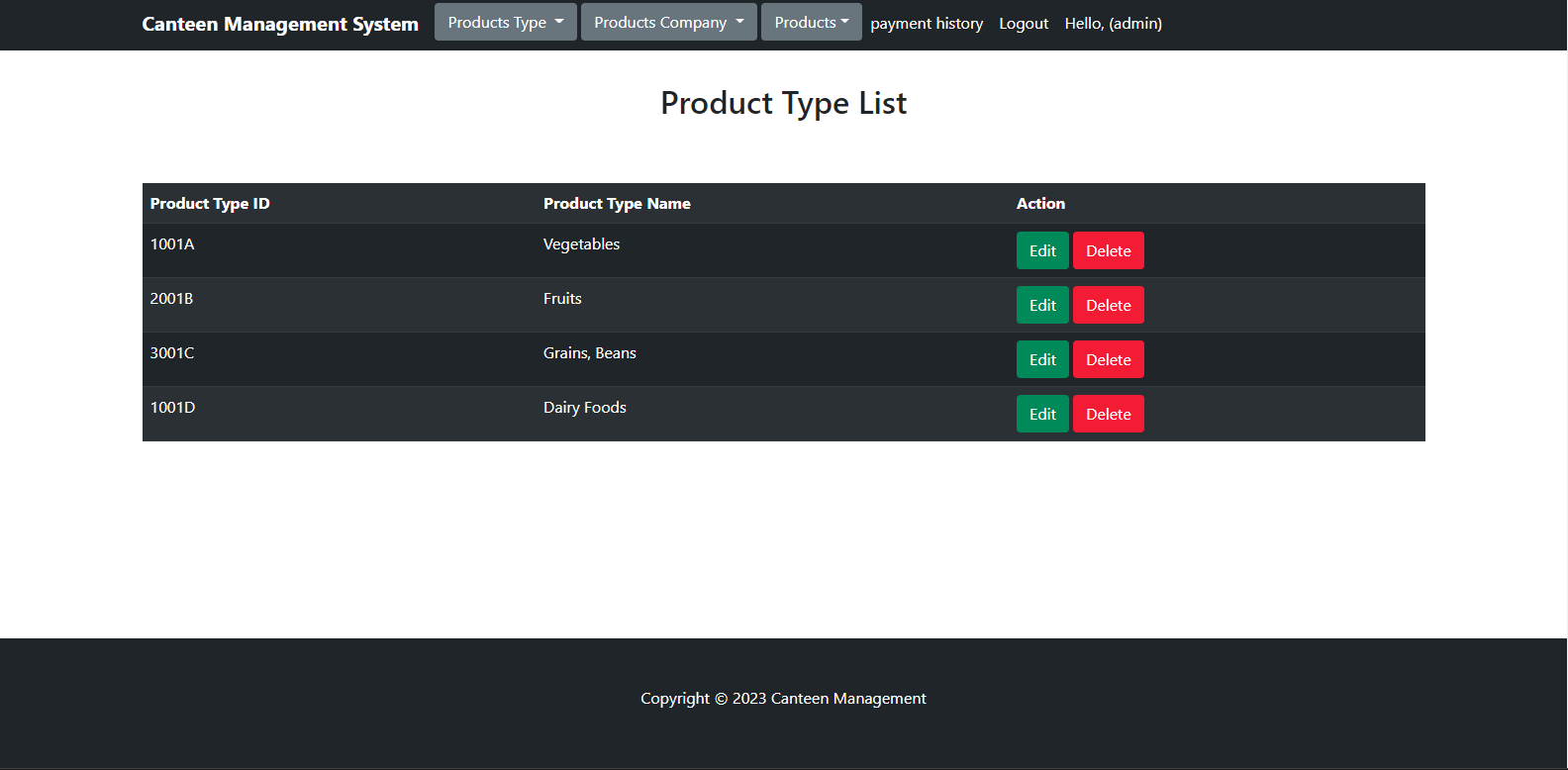
In the current physical system is thoroughly reviewed from point of view like how the data flow, what are file contents, its volumes, complexity and frequency etc.

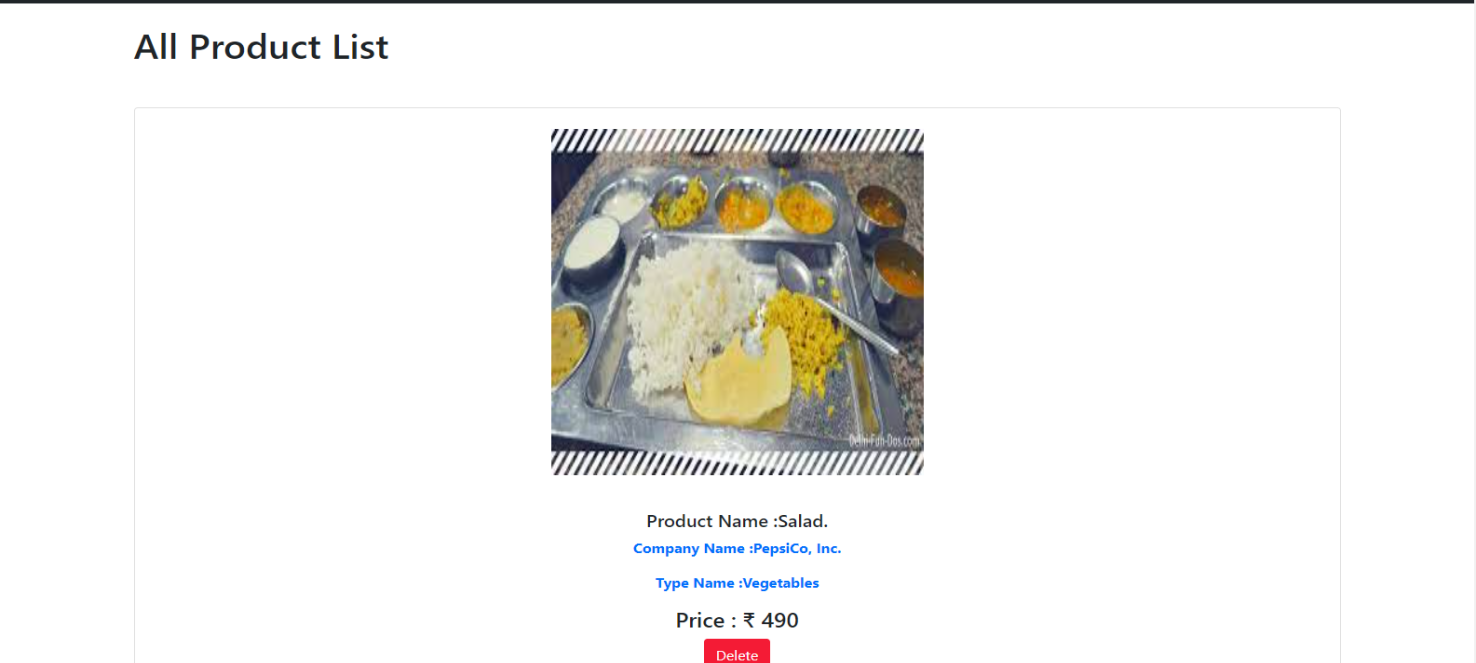
After this input, output specifications security and control specification is prepare. It was also decided that how physical data information will flow through the system and a physical design walkthrough.

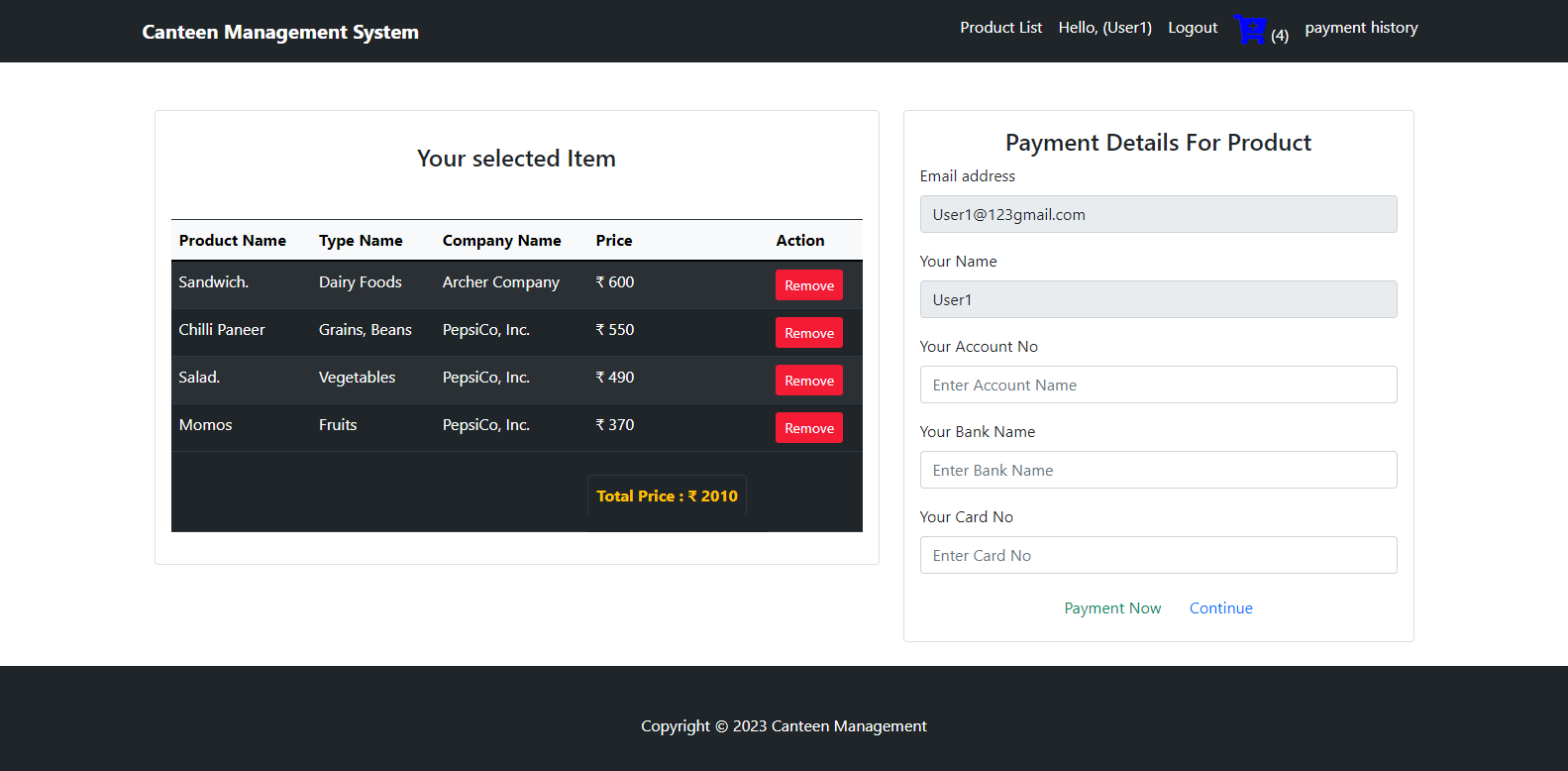
**OUTPUT DESIGN**

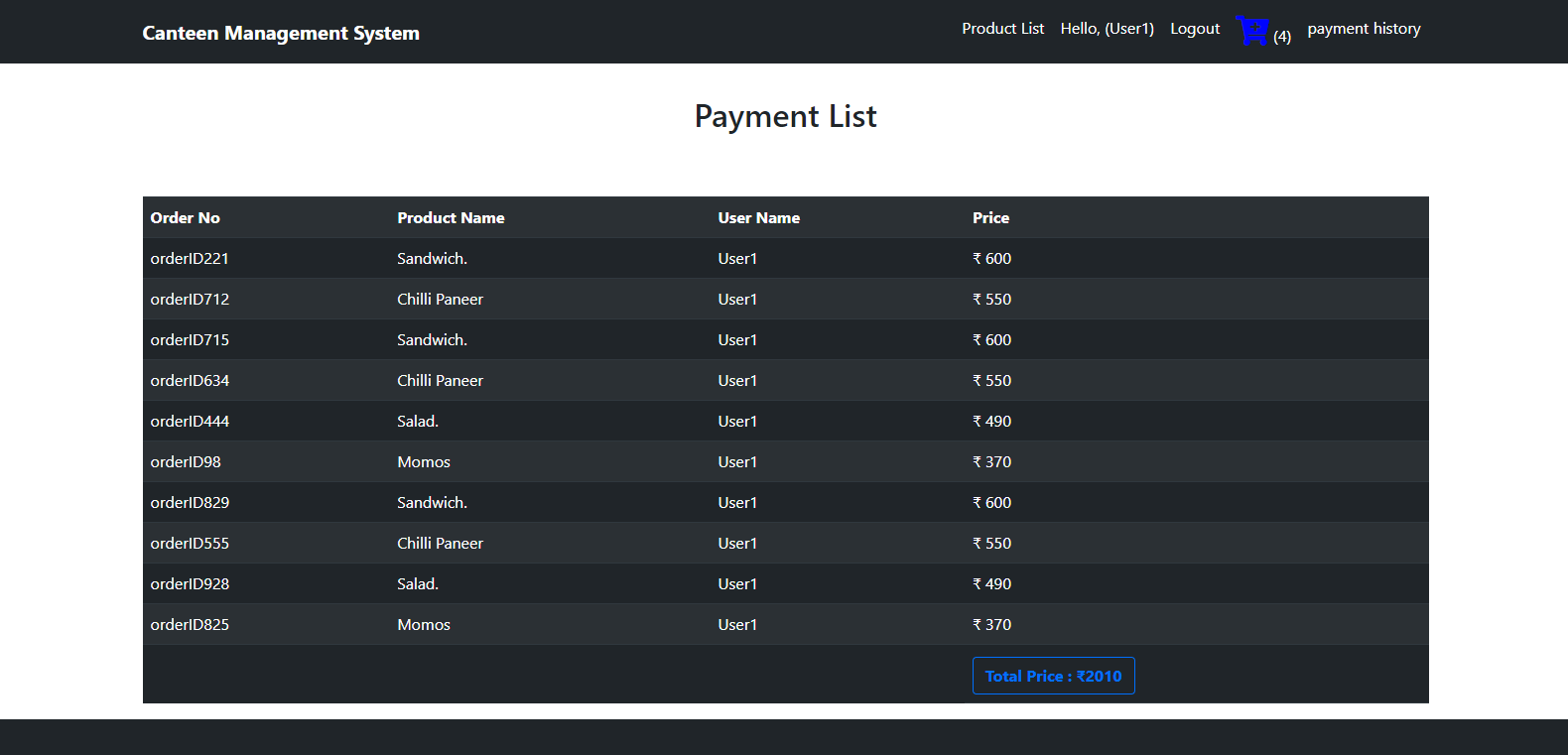
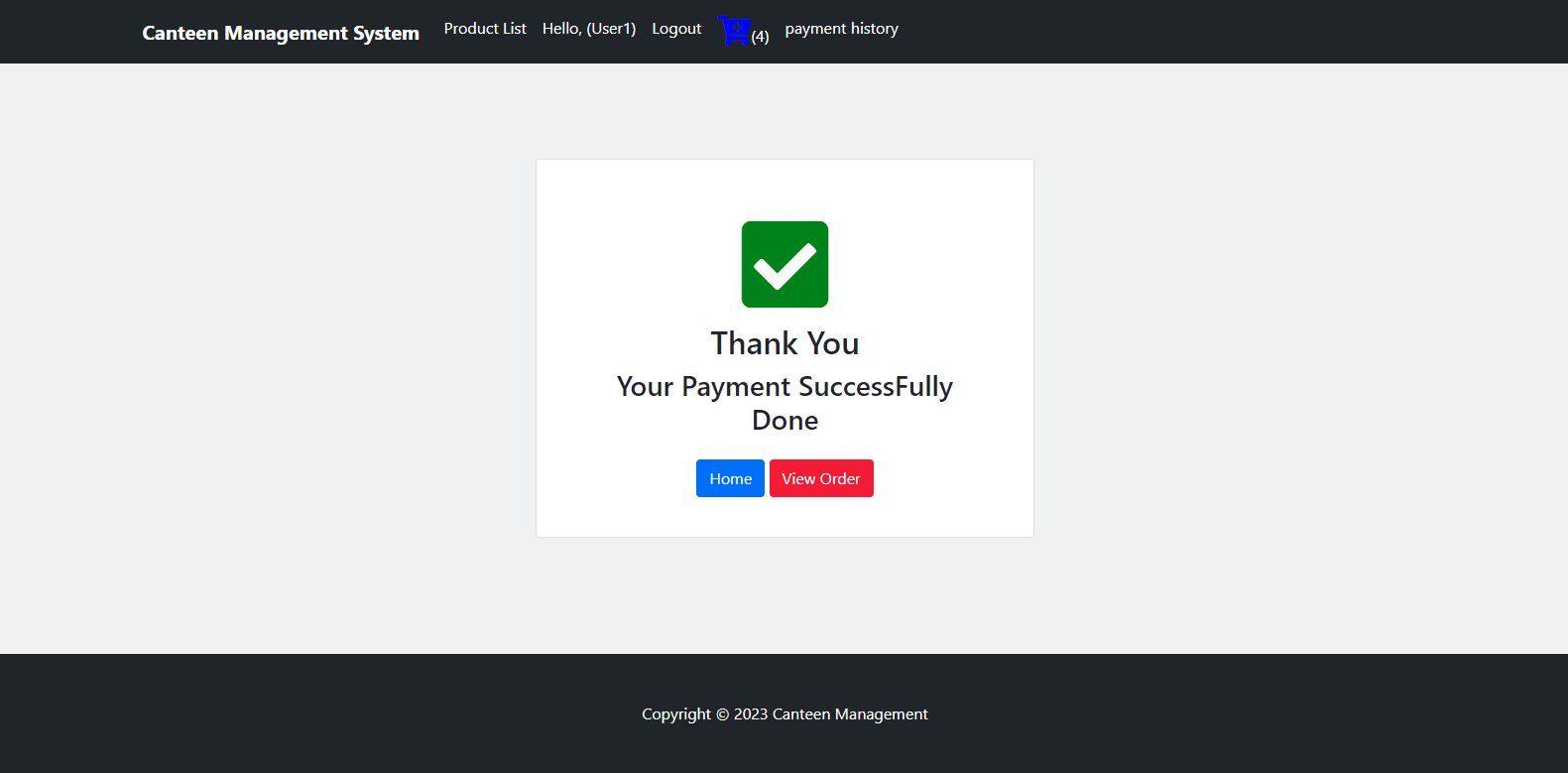
* The format of outputs is designed in user-friendly way that it is simple to read and interpret in the present output we have clearly labeled title, messages it contains date and time and all the fields are clearly mentioned as placeholders or label.

**Screen Short**

****







**Source Code**

**package** in.co.canteen.mg.Bean;

**import** java.sql.Blob;

**public** **class** ProductDetailsBean **extends** BaseBean{

**private** String productName;

**private** **long** price;

**private** Blob image;

**private** String typeName;

**private** **long** typeid;

**private** **int** quantity;

**public** **int** getQuantity() {

**return** quantity;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

**public** String getProductName() {

**return** productName;

}

**public** **void** setProductName(String productName) {

**this**.productName = productName;

}

**public** **long** getPrice() {

**return** price;

}

**public** **void** setPrice(**long** price) {

**this**.price = price;

}

**public** Blob getImage() {

**return** image;

}

**public** **void** setImage(Blob image) {

**this**.image = image;

}

**public** String getTypeName() {

**return** typeName;

}

**public** **void** setTypeName(String typeName) {

**this**.typeName = typeName;

}

**public** **long** getTypeid() {

**return** typeid;

}

**public** **void** setTypeid(**long** typeid) {

**this**.typeid = typeid;

}

**public** String getComapnyName() {

**return** comapnyName;

}

**public** **void** setComapnyName(String comapnyName) {

**this**.comapnyName = comapnyName;

}

**public** **long** getCompanyid() {

**return** companyid;

}

**public** **void** setCompanyid(**long** companyid) {

**this**.companyid = companyid;

}

**private** String comapnyName;

**private** **long** companyid;

**private** **long** cartid;

**public** **long** getCartid() {

**return** cartid;

}

**public** **void** setCartid(**long** cartid) {

**this**.cartid = cartid;

}

@Override

**public** String getKey() {

// **TODO** Auto-generated method stub

**return** id+"";

}

@Override

**public** String getValue() {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

package in.co.canteen.mg.Controller;

import java.io.IOException;

import java.io.InputStream;

import java.sql.Blob;

import java.sql.SQLException;

import java.time.LocalTime;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.MultipartConfig;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.Part;

import javax.sql.rowset.serial.SerialBlob;

import javax.sql.rowset.serial.SerialException;

import in.co.canteen.mg.Bean.BaseBean;

import in.co.canteen.mg.Bean.ProductDetailsBean;

import in.co.canteen.mg.Bean.ProductsTypeBean;

import in.co.canteen.mg.Exception.ApplicationException;

import in.co.canteen.mg.Exception.DuplicateRecordException;

import in.co.canteen.mg.Model.ProductCompanyModel;

import in.co.canteen.mg.Model.ProductDetailsModel;

import in.co.canteen.mg.Model.ProductTypeModel;

import in.co.canteen.mg.Utility.DataUtility;

import in.co.canteen.mg.Utility.DataValidator;

import in.co.canteen.mg.Utility.PropertyReader;

import in.co.canteen.mg.Utility.ServletUtility;

@WebServlet(name = "ProductdetailsCtl",urlPatterns = "/productdetailsCtl")

@MultipartConfig(maxFileSize = 16177215)

public class ProductdetailsCtl extends BaseCtl {

private static final long serialVersionUID = 1L;

public ProductdetailsCtl() {

super();

}

@Override

protected void preload(HttpServletRequest request) {

ProductTypeModel model = new ProductTypeModel();

ProductCompanyModel companymodel = new ProductCompanyModel();

try {

List producttypelist = model.list();

List companylist = companymodel.list();

request.setAttribute("productTypeName", producttypelist);

request.setAttribute("companyName", companylist);

} catch (Exception e) {

e.printStackTrace();

}

}

@Override

protected boolean validate(HttpServletRequest request) {

System.out.println("in validation method");

boolean pass = true;

if (DataValidator.isNull(request.getParameter("productName"))) {

request.setAttribute("productName", PropertyReader.getvalue("error.require", "productName"));

pass = false;

}

if (DataValidator.isNull(request.getParameter("price"))) {

request.setAttribute("price", PropertyReader.getvalue("error.require", "price"));

pass = false;

}

if (DataValidator.isNull(request.getParameter("type"))) {

request.setAttribute("type", PropertyReader.getvalue("error.require", "type"));

pass = false;

}

if (DataValidator.isNull(request.getParameter("company"))) {

request.setAttribute("company", PropertyReader.getvalue("error.require", "company"));

pass = false;

}

return pass;

}

@Override

protected BaseBean populateBean(HttpServletRequest request) {

ProductDetailsBean bean = new ProductDetailsBean();

bean.setId(DataUtility.getLong(request.getParameter("id")));

bean.setProductName(DataUtility.getString(request.getParameter("productName")));

bean.setTypeid(DataUtility.getLong(request.getParameter("type")));

bean.setCompanyid(DataUtility.getLong(request.getParameter("company")));

bean.setPrice(DataUtility.getLong(request.getParameter("price")));

bean.setCartid(0);

Blob blob = null;

Part filePart;

try {

filePart = request.getPart("image");

blob = medicinePacketUpload(filePart);

} catch (Exception e) {

e.printStackTrace();

}

System.out.println("image :" + blob);

bean.setImage(blob);

populateDTO(bean, request);

return bean;

}

public Blob medicinePacketUpload(Part part) throws IOException {

InputStream inputStream = null;

Blob blob = null;

inputStream = part.getInputStream();

byte[] b = new byte[inputStream.available()];

inputStream.read(b);

try {

blob = new SerialBlob(b);

} catch (SerialException e) {

e.printStackTrace();

} catch (SQLException e) {

e.printStackTrace();

}

return blob;

}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

ServletUtility.forward(getView(), request, response);

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

System.out.println("in do post");

ProductDetailsModel model = new ProductDetailsModel();

String op = DataUtility.getString(request.getParameter("operation"));

long id = DataUtility.getLong(request.getParameter("id"));

ProductDetailsBean bean = new ProductDetailsBean();

bean = (ProductDetailsBean) populateBean(request);

if (OP\_SUBMIT.equalsIgnoreCase(op)) {

try {

long pk = model.add(bean);

ServletUtility.setbean(bean, request);

ServletUtility.setSuccessMessage("Product Added !!", request);

ServletUtility.forward(getView(), request, response);

return;

} catch (DuplicateRecordException e) {

ServletUtility.setbean(bean, request);

ServletUtility.setErrorMessage(e.getMessage(), request);

ServletUtility.forward(getView(), request, response);

} catch (ApplicationException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}else {

//long pk = model.Update(bean);

ServletUtility.setbean(bean, request);

ServletUtility.setSuccessMessage("Product Updated !!", request);

}

ServletUtility.forward(getView(), request, response);

}

@Override

protected String getView() {

return CMSView.PRODUCT\_DETAILS\_VIEW;

}

}

package in.co.canteen.mg.Model;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import in.co.canteen.mg.Bean.ProductCompanyBean;

import in.co.canteen.mg.Bean.ProductDetailsBean;

import in.co.canteen.mg.Bean.ProductsTypeBean;

import in.co.canteen.mg.Exception.ApplicationException;

import in.co.canteen.mg.Exception.DuplicateRecordException;

import in.co.canteen.mg.Utility.JDBCDataSource;

public class ProductDetailsModel {

public Integer nextPk() throws Exception {

Connection conn = null;

int pk = 0;

try {

conn = JDBCDataSource.getConnection();

PreparedStatement ps = conn.prepareStatement("SELECT MAX(ID) FROM productdetails");

ResultSet rs = ps.executeQuery();

while (rs.next()) {

pk = rs.getInt(1);

}

} catch (SQLException e) {

e.printStackTrace();

}

return pk + 1;

}

public long add(ProductDetailsBean bean) throws Exception {

Connection conn = null;

int pk = 0;

ProductTypeModel model = new ProductTypeModel();

ProductsTypeBean typeBean = new ProductsTypeBean();

typeBean = model.findByPk(bean.getTypeid());

String typeName = typeBean.getProductName();

ProductCompanyModel model1 = new ProductCompanyModel();

ProductCompanyBean companyBean = new ProductCompanyBean();

companyBean = model1.findByPk(bean.getCompanyid());

String companyName = companyBean.getCompanyName();

ProductDetailsBean existbean = findByProductName(bean.getProductName());

if (existbean != null) {

throw new DuplicateRecordException("Product Name is already exist");

}

try {

conn = JDBCDataSource.getConnection();

pk = nextPk();

conn.setAutoCommit(false);

PreparedStatement ps = conn.prepareStatement("INSERT INTO productdetails VALUES(?,?,?,?,?,?,?,?)");

ps.setLong(1, pk);

ps.setString(2, bean.getProductName());

ps.setLong(3, bean.getPrice());

ps.setBlob(4, bean.getImage());

ps.setLong(5, bean.getTypeid());

ps.setString(6, typeName);

ps.setLong(7, bean.getCompanyid());

ps.setString(8, companyName);

ps.executeUpdate();

conn.commit();

ps.close();

} catch (Exception e) {

e.printStackTrace();

try {

conn.rollback();

} catch (Exception e2) {

e.printStackTrace();

throw new ApplicationException("Exception : add rollback exception " + e.getMessage());

}

} finally {

JDBCDataSource.closeconnection(conn);

}

return pk;

}

private ProductDetailsBean findByProductName(String productName) throws Exception {

ProductDetailsBean bean = null;

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

} catch (SQLException e) {

e.printStackTrace();

}

try {

PreparedStatement ps = conn.prepareStatement("SELECT \* FROM productdetails WHERE productName=?");

ps.setString(1, productName);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

bean = new ProductDetailsBean();

bean.setId(rs.getLong(1));

bean.setProductName(rs.getString(2));

bean.setPrice(rs.getLong(3));

bean.setImage(rs.getBlob(4));

bean.setTypeid(rs.getLong(5));

bean.setTypeName(rs.getString(6));

bean.setCompanyid(rs.getLong(7));

bean.setComapnyName(rs.getString(8));

}

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

return bean;

}

public ProductDetailsBean findByPk(long pk) throws Exception {

ProductDetailsBean bean = null;

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

} catch (SQLException e) {

e.printStackTrace();

}

try {

PreparedStatement ps = conn.prepareStatement("SELECT \* FROM productdetails WHERE id=?");

ps.setLong(1, pk);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

bean = new ProductDetailsBean();

bean.setId(rs.getLong(1));

bean.setProductName(rs.getString(2));

bean.setPrice(rs.getLong(3));

bean.setImage(rs.getBlob(4));

bean.setTypeid(rs.getLong(5));

bean.setTypeName(rs.getString(6));

bean.setCompanyid(rs.getLong(7));

bean.setComapnyName(rs.getString(8));

}

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

return bean;

}

public List list() throws Exception {

ArrayList list = new ArrayList();

try {

Connection conn = null;

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement("SELECT \* from productdetails");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

ProductDetailsBean bean = new ProductDetailsBean();

bean.setId(rs.getLong(1));

bean.setProductName(rs.getString(2));

bean.setPrice(rs.getLong(3));

bean.setImage(rs.getBlob(4));

bean.setTypeid(rs.getLong(5));

bean.setTypeName(rs.getString(6));

bean.setCompanyid(rs.getLong(7));

bean.setComapnyName(rs.getString(8));

list.add(bean);

}

} catch (Exception e) {

e.printStackTrace();

}

return list;

}

public static long delete(long id) throws Exception {

int pk = 0;

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

conn.setAutoCommit(false); // Begin transaction

PreparedStatement pstmt = conn.prepareStatement("DELETE FROM productdetails WHERE id=?");

pstmt.setLong(1, id);

pstmt.executeUpdate();

conn.commit(); // End transaction

pstmt.close();

} catch (Exception e) {

e.printStackTrace();

}

return pk;

}

public long cartIdUpdate(long id,long Cid) {

int pk = 0;

try {

Connection conn = JDBCDataSource.getConnection();

PreparedStatement ps = conn

.prepareStatement("update productdetails set cartid='" + Cid + "' where id=?");

ps.setLong(1, Cid);

ps.executeUpdate();

} catch (Exception e) {

e.printStackTrace();

}

return pk;

}

}

<%@page import=*"in.co.canteen.mg.Controller.ProductdetailsCtl"*%>

<%@page import=*"java.util.List"*%>

<%@page import=*"java.util.Iterator"*%>

<%@page import=*"in.co.canteen.mg.Controller.ProductTypeCtl"*%>

<%@page import=*"in.co.canteen.mg.Utility.ServletUtility"*%>

<%@page import=*"in.co.canteen.mg.Utility.HTMLUtility"*%>

<%@page import=*"in.co.canteen.mg.Utility.DataUtility"*%>

<%@page import=*"in.co.canteen.mg.Controller.CMSView"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<%@include file=*"Header.jsp"*%>

<%

List producttypelist = (List) request.getAttribute("productTypeName");

List companylist = (List) request.getAttribute("companyName");

%>

<br>

<div class=*"container mt-5"*

style="position: *relative*; min-height: *58vh*">

<div class=*"row"*>

<div class=*"col-2"*></div>

<div class=*"col-8"*>

<form action=*"*<%=CMSView.PRODUCT\_DETAILS\_CTL%>*"* method=*"post"* enctype=*"multipart/form-data"*>

<h2>Add Product</h2>

<hr>

<h6 style="color: *red*;"><%=ServletUtility.getErrorMessage(request)%></h6>

<h6 style="color: *green*;"><%=ServletUtility.getSuccessMessage(request)%></h6>

<jsp:useBean id=*"bean"* scope=*"request"*

class=*"in.co.canteen.mg.Bean.ProductDetailsBean"* />

<input type=*"hidden"* name=*"id"* value=*"*<%=bean.getId()%>*"*>

<div class=*"container"*>

<div class=*"col-md-12"*>

<label for=*"inputAddress"* class=*"form-label"*>Product Name</label>

<input type=*"text"* class=*"form-control"* id=*"inputAddress"*

name=*"productName"* placeholder=*"Enter here..."* value=*""*>

</div>

<font color=*"red"*><%=ServletUtility.getErrorMessage("productName", request)%></font>

<!-- Type -->

<div class=*"col-md-12"*>

<label for=*"inputAddress"* class=*"form-label col-md-12"*>Prodcut Type:</label> <select name=*"type"* class=*"form-control"*>

<%=HTMLUtility.getList("type", String.valueOf(bean.getTypeid()), producttypelist)%>

<font color=*"red"*><%=ServletUtility.getErrorMessage("type", request)%></font>

</select>

</div>

<!-- Type End -->

<div class=*"col-md-12"*>

<label for=*"inputAddress"* class=*"form-label col-md-12"*>Product Company:</label> <select name=*"company"* class=*"form-control"*>

<%=HTMLUtility.getList("company", String.valueOf(bean.getCompanyid()), companylist)%>

<font color=*"red"*><%=ServletUtility.getErrorMessage("company", request)%></font>

</select>

</div>

<div class=*"col-md-12"*>

<label for=*"inputAddress"* class=*"form-label"*>Product Price</label>

<input type=*"number"* class=*"form-control"* id=*"inputAddress"*

name=*"price"* placeholder=*"Enter here..."* value=*""*>

</div>

<font color=*"red"*><%=ServletUtility.getErrorMessage("price", request)%></font>

<div class=*"col-md-12"*>

<label for=*"exampleFormControlInput1"* class=*"form-label"*>

Image:</label> <br><input type=*"file"* id=*"exampleFormControlInput1"* class=*"form-control"* required=*"required"*

name=*"image"* value=*"*<%=DataUtility.getStringData(bean.getImage())%>*"*>

</div>

<br>

<input type=*"submit"* class=*"btn btn-primary"* name=*"operation"*

style="margin-left: *300px*;" value=*"*<%=ProductdetailsCtl.OP\_SUBMIT%>*"*>

</div>

</form>

</div>

</div>

</div>

<div style="margin-top: *2%*;">

<%@include file=*"Footer.jsp"*%>

</div>

</body>

</html>

**TESTING**

**Software Testing**

Software testing is a process of *verifying* and *validating* that a software application or program.

Meets the business and technical requirements that guided its design and development, and Works as expected.

Software testing also identifies important *defects*, flaws, or errors in the application code that must be fixed. The modifier “important” in the previous sentence is, well, important because defects must be categorized by severity.

During test planning we decide what an important defect is by reviewing the requirements and design documents with an eye towards answering the question “Important to whom?” Generally speaking, an important defect is one that from the customer’s perspective affects the usability or functionality of the application.

The quality assurance aspect of software development documenting the degree to which the developers followed corporate standard processes or best practices is not addressed in this paper because assuring quality is not a responsibility of the testing team.

The testing team cannot improve quality they can only measure it, although it can be argued that doing things like designing tests before coding begins will improve quality because the coders can then use that information while thinking about their designs and during coding and debugging.

Software testing has three main purposes: verification, validation, and defect finding.

The *verification* process confirms that the software meets its technical specifications. A “specification” is a description of a function in terms of a measurable output value given a specific input value under specific preconditions.

A simple specification may be along the line of “a SQL query retrieving data for a single account against the multi-month account-summary table must return these eight fields <list> ordered by month within 3 seconds of submission.”

The *validation* process confirms that the software meets the business requirements. A simple example of a business requirement is “After choosing a branch office name, information about the branch’s customer account managers will appear in a new window. The window will present manager identification and summary information about each manager’s customer base: <list of data elements>.”

Other requirements provide details on how the data will be summarized, formatted and displayed.

A *defect* is a variance between the expected and actual result. The defect’s ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

## Testing methods

Software testing methods are traditionally divided into black box testing and white box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

#### Black box testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

**Specification-based testing**: Specification-based testing aims to test the functionality of software according to the applicable requirements. Thus, the tester inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case.

Specification-based testing is necessary, but it is insufficient to guard against certain risks.

**Advantages and disadvantages**:

The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code *must* have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not.

*But,* on the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed.

As a result, there are situations when 1 a tester writes many test cases to check something that could have been tested by only one test case, and/or 2 some parts of the back-end are not tested at all.

Therefore, black box testing has the advantage of "an unaffiliated opinion," on the one hand, and the disadvantage of "blind exploring," on the other.

#### White box testing

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

Types of white box testing

API testing (application programming interface) - Testing of the application using Public and Private APIs

Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)

Fault injection methods - improving the coverage of a test by introducing faults to test code paths

**Mutation testing methods**

**Static testing** White box testing includes all static testing

**A sample testing life cycle**

Although variations exist between organizations, there is a typical cycle for testing:

**Requirements analysis**: Testing should begin in the requirements phase of the software development life cycle. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.

**Test planning**: Test strategy, test plan, tested creation. Since many activities will be carried out during testing, a plan is needed.

**Test development**: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software.

**Test execution**: Testers execute the software based on the plans and tests and report any errors found to the development team.

**Test reporting**: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.

**Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be treated, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.

**Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing team.

**Regression testing**: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.

**Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

**Conclusion**

• The development of Canteen Management System involved many

phases. The approach used is a top-down one concentrating on what first

then how and moving to successive levels of details.

• The first phase started with a detailed study of the problems and prospects

of ordering in Foods.

• This Software is efficient in maintaining customer’s details and can easily

perform operations on platform