***INTRODUCTION***

**“ZuperMarket (On-Line Shopping System / E-commerce Website)”** is a web-based project which is made for remote-shopping or shopping through Internet. As the technology is being advanced the way of life is changing accordance. Now a day’s we can place the order for anything from our home. There is no need to go the shop of the things we want. The order can be placed online through Internet. The payment, the confirmation of purchasing; we can do everything that we want. Now we can think that how the days have been changed with time. People had to stand in rows to wait there terms to buy a particular thing from a popular shop. But what is happening now a day’s; we can extremely surprise that those things can be available on the door-step in few hours. People had to suffer the rush of the market when they went for shopping. They used to think hundred times to buy anything having the sufficient money for shopping. The problem was the rush; the quarrel at the time of buying the things. But the advancement of technology brought the new way for shopping. The way of shopping was completely changed with the coming of Internet Technology. People have to fill a simple form on the internet to place their order on any popular shop or shopping-mall for the thing they want to buy. Now they can place their order from the home. This project entitled “On-Line Shopping” is an implementation of the above description. It means, it implements the E-shopping or in other word shopping through Internet. It lets the user to place their order online for any article.

***OBJECTIVE***

Today the internet and its boom have created a new economic scenario that not only stresses on the classical concept of the “product” but also on the modern concept of “service”. It is this level of service that dictates whether a commercial venture will succeed or not in the market. To provide a high accessibility of service we will design the online shopping website, so that potential customers need not go to a physical shop to buy products or services. There are several objective of this websites are following given below:

• This site is gives all the information about the e-shopping to provide better service for the customer.

• It provides the facility to the customers who want to shop on-line due to lack of time.

• It’s providing the full details about the product and related information about the product like cost, size etc.

• With the help of it we can save the time and money also.

• It provides multiple payment methods for shopping by the cash, Debit card and credit card also.

• It provides better security and good delivery service to the customer.

***MODULE DESCRIPTION***

We know that it is quite difficult to do anything of its whole part at a time. So it is the task of the programmer to break/split the whole set of task into various small module so that one can handle them effectively. But it depends upon the software system that in which Extent it will be broken so that there is no inter-dependency among them.

This project developed for “On-Line Shopping” is broken into following module.

1. Admin Module

2. User Module

3. Customer Service Module

4. Payment Module

5. Purchase Module

6. Searching Module

**ADMIN MODULE:**

Admin module is mainly for the owner of the system. This module performs the work of regular checking process. Everyone needs current information about anything. So, this module is responsible for updating the information continuously if any, in the system so that one can find timely information. But the main function of this module is to handle security of this system. Different types of authentication and authorization process is performed for this purpose. This module is also responsible for advertisement bid. Any organizations that are eager for advertisement can contact us for their ad. This module handles the entire task related to advertisement like specifying and allocating the space and related cost.

**USER MODULE:**

User module is the front page of this system. It is the main page of the system in which all the links of this system are kept. Now the main thing is how to make the home page interactive and user friendly so that there should not be any problem for the user. This is the task of the system administrator to make the home page Interactive.

**CUSTOMER SERVICE MODULE:**

In the customer service module, the customer will use this website to on-line shopping. And also the customer send the request to the shopping.

**PAYMENT MODULE:**

In this module, the customer will make payment in different mode like cash on delivery, or by credit card, or by debit card.

**PURCHASE MODULE:**

In this module, the all selected item could be purchased by the customer. And will do payment.

**SEARCH MODULE:**

Search module provides search facilities to the search different product to the purchase.

***IMPLEMENTATION METHODOLOGY***

**MVC** is short for Model, View, and Controller. MVC is a popular way of organizing your code. The big idea behind MVC is that each section of your code has a purpose, and those purposes are different. Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app functions.

MVC is a way to organize your code’s core functions into their own, neatly organized boxes. This makes thinking about your app, revisiting your app, and sharing your app with others much easier and cleaner.

Model View Controller or MVC as it is popularly called is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

* **Model –** The lowest level of the pattern which is responsible for maintaining data. Model code typically reflects real-world things. This code can hold raw data, or it will define the essential components of your app. For instance, if you were building a To-do app, the model code would define what a “task” is and what a “list” is – since those are the main components of a todo app.
* **View –** This is responsible for displaying all or portion of the data to the user. View code is made up of all the functions that directly interact with the user. This is the code that makes your app look nice, and otherwise defines how your user sees and interacts with it.
* **Controller –** Software code that controls the interactions between the Model and View. Controller code acts as a liaison between the Model and the View, receiving user input and deciding what to do with it. It’s the brains of the application, and ties together the model and the view.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The view then uses the data prepared by the Controller to generate a final presentable response.

**MVC IN THE REAL-WORLD**

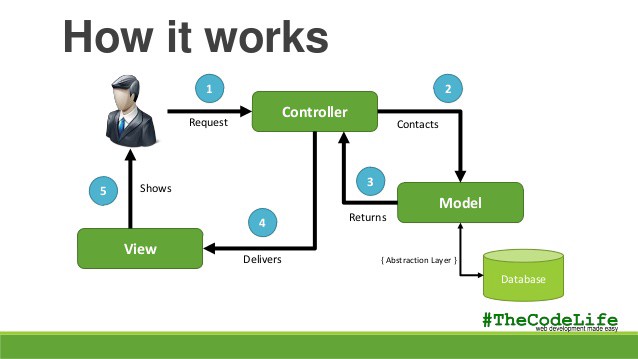
MVC is helpful when planning your app, because it gives you an outline of how your ideas should be organized into actual code.

For instance, let's imagine you're creating a To-do list app. This app will let users create tasks and organize them into lists.

The Model in a todo app might define what what a “task” is and that a “list” is a collection of tasks.

The View code will define what the todos and lists looks like, visually. The tasks could have large font, or be a certain color.

Finally, the Controller could define how a user adds a task, or marks another as complete. The Controller connects the View’s add button to the Model, so that when you click “add task,” the Model adds a new task.



***SYSTEM ANALYSIS***

The analysis model is a concise, precise abstraction of what the desired system must do, and not how it will be done after the study of the existing system is completed. This basically includes system study and the requirement analysis.

Interacting with the clients regarding their requirements and expectations from the system does requirement analysis.

The steps, which are essential for system analysis, are:

1. Research and define essential components.
2. Analyze current processes and identify gaps.
3. Interview users, Trainee/ Trainers and other concerned personnel regarding essential components and current processes.
4. Write requirements document.
5. Define standards for standards, policies, and procedures.
6. Review draft requirements document with users, Trainee, Trainers and other concerned personnel.
7. Update and expand project plan.

**Problem Definition**

The ZuperMarket (On-Line Shopping System / E-commerce) Administrator is the super user and has complete control over all the activities that can be performed. The application notifies the administrator of all shop creation requests, and the administrator can then approve or reject them. The administrator also manages the list of available product categories. The administrator can also view and delete entries in the guestbook.

**Identification of Need**

Online Shopping Management Systems described above can lead to error free, secure reliable and fast management system for Shopping . It can assist the staff to concentrate on their related (for Shopping) activities rather to concentrate on the records and reports of issuing Products, registration and suppliers. This will help organization in better utilization of human resources. In this phase, the user identifies the need for a new or improved system. In large organizations this identification may be part of a systems planning process.

**Shop Owner:**

Any user can submit a shop creation request through the application. When the request is approved by the Mall Administrator, the requester is notified, and from there on is given the role of Shop Owner. The Shop Owner is responsible for setting up the shop and maintaining it. The job involves managing the sub-categories of the items in the shop. Also, the shop owner can add or remove items from his shop. The Shop Owner can view different reports that give details of the sales and orders specific to his shop. The Shop Owner can also decide to close shop and remove it from the mall.

**Employees:**

Purchase department under a Purchase manager to overlook purchasing activities if warehousing needs arise. Sales department under a Sales manager who will look after the sale of products and services, the most important activity. Accounts department under an Accounts manager to look after the accounting activities of the enterprise.

**Requirement Specification:**

It is quite difficult and time consuming task to find the information as well as maintaining information manually. If all these information are to be kept at a single place it is also not possible in the manual system. Computerized system will upgrade and manage information very easily. As it is a web-based application so it uses Internet technologies and its hardware/software requirement will also be more comprehensive than Desktop application system. Some Network devices will be required like modems, switches, Internet connection. Software required for the system is also different from a normal desktop system. First of all a server software will be mandatory (here Internet Information Server (IIS)). A browser is also needed as a client process on the user side.

**Preliminary Investigation:**

System Analysis is not only time consuming but also a rigorous task. But it is crucial and most important phase of Software development process. Preliminary Investigation is the process of gathering data for requirement analysis. It is more helpful for problem definition and requirement specification.

**Benefit to Organization**

The organization will obviously be able to gain benefits such as savings in operating cost, reduction in, paperwork, better utilization of human resources and more presentable image increasing goodwill. The other benefits are improved service  
and faster and better access to up-to-date information.

**The Initial Cost**

The initial cost of setting up the system will include the cost of hardware (server/clients, network adapter and related hardware), software (server OS, add-on Software, utilities) & labor (setup & maintenance). The same has to bear by the  
organization.

**Running Cost**

Besides, the initial cost the long term cost will include the running cost for the system including the AMC, Product charges, cost for human resources, cost for update/renewal of various related software.

**Need for Training**

The users along with the administrator need to be trained at the time of implementation of the system for smooth running of the system. The client will provide the training site.

**Depending upon this definition**, we analyzed the present Shopping Management System, during the analysis process, we had to go through the entire manual system that was being followed in the Organization for past few years.

We talked to the management people who were managing a the financial issues of the center, staff who were keeping the records in lots of registers and the reporting manager

regarding their existing system, their requirements and their expectations from the new proposed system. Then, we did the system study of the entire system based on their requirements and the additional features they wanted to incorporate in this system.

The manual system was very complex to be managed and searching or updating the information was also very difficult.

There was a lengthy procedure for entering the details of Products, members, visitors, and then' making their timings and Bills.

The other problem with the manual system was of report generation. The reports to be generated on a daily/weekly/monthly basis required lot of paper work and calculations etc. Thus, Reporting Manager" had to manually calculate all the consultation fees of every player at the end of every day/week/month. Even a small calculation mistake was leading to reanalysis and recreation of whole work.

***FEASIBILITY ANALYSIS***

Whatever we think need not be feasible .It is wise to think about the feasibility of any problem we undertake. Feasibility is the study of impact, which happens in the organization by the development of a system. The impact can be either positive or negative. When the positives nominate the negatives, then the system is considered feasible. They were three key consideration involved in this feasibility analysis each consideration has reviewed to depict how it relates to the system effort.

They are as follows:-

1. Technical feasibility

2. Economic feasibility

3. Operational feasibility

**Technical Feasibility:**

We can strongly says that it is technically feasible, since there will not be much difficulty in getting required resources for the development and maintaining the system as well. All the resources needed for the development of the software as well as the maintenance of the same is available in the organization here we are utilizing the resources which are available already.

Technical feasibility can be evaluated only after those phases during which technical issues are resolved- namely after the evaluation and design phase of our life cycle have been completed. Today very little is technically impossible. Consequently technical feasibility looks at what is practical and reasonable. Technical feasibility addresses three major issues:

* Is the proposed technology or solution practical?
* Do we currently possess the necessary technology?
* Do we possess the necessary technical expertise, and is the schedule reasonable?

**IS THE PROPOSED TECHNOLOGY OR SOLUTION PRACTICAL?**

The technology is very practical for the current system. Organization is using Angular4,SpringBoot And Microservice application support with postGreSQL Database Management Software. Plus Apache Tomcat Server Site Server.

**DO WE CURRENTLY POSSESS THE NECESSARY TECHNOLOGY?**

The organization will buy all the necessary technology for example the Pc’s, Routers,Blade Servers, WAN devices, SAN etc

DO WE POSSESS THE NECESSARY TECHNICAL EXPERTISE?

We may have the technology but that does not mean that we have the required expertise and skills to implement the technology effectively. We will hire

**Economical Feasibility:**

Development of this application is highly economically feasible .The organization needed not spend much money for the development of the system already available. The only thing is to be done is making an environment for the development with an effective supervision. If we are doing so, we can attain the maximum usability of the corresponding resources .Even after the development, the organization will not be in condition to invest more in the organization. Therefore, the system is economically feasible.

The manual efforts involved in maintaining the record and student information, is tremendous. This is so because the volume of information to be handled is tremendous. Maintaining the records of these many student is not easy and the manpower involved is great.

ALTERNATIVES

1. Hire more staff

2. Develop a computerized system using VISUAL BASIC & SQL server as database provider.

3. Develop a computerized system using VISUAL BASIC & ORACLE as the database provider.

4. Develop a computerized system using C++ & maintaining data file i.e. \*.dat for database.

Its Advantages are:-

1. The system will be platform independent and there will be no need is for purchasing other very expensive software. The jdk/jvm is easily and freely available and is compatible with most of the computers. The system can be upgraded and maintained easily to keep in tune with specific needs of the customer. The memory required by this software will not be much, so we won’t have to upgrade the present system (computer).

The last alternatives suggested by the software development team are the cheapest possible one. The software is text based so no formal training will have to be imparted. The compiler is freely available. The only expenditure that needs to be done is money cost of software team.

**Operational Feasibility :**

It is mainly related to human organizational and political aspects. The points to be considered are:

• What changes will be brought with the system?

• What organizational structures are distributed?

• What new skills will be required? Do the existing staff members have these skills?

• If not, can they be trained in due course of time?

Generally project will not be rejected simply because of operational infeasibility but such considerations are likely to critically affect the nature and scope of the eventual recommendations.

For operational feasibility study we appointed a small group of people who are familiar with information system techniques, who understand the parts of the business that are relevant to the project and are skilled in system analysis and design process.

Operational feasibility criteria measure the urgency of the problem (survey and study phases) or the acceptability of the solution. There are two aspects of operational feasibility to be considered:

* Will the solution to the problem work?
* How do the customer and organization feel about the solution?

**WILL THE SOLUTION TO THE PROBLEM WORK**

PIECES framework can be used as a basis for analyzing the urgency of the problem or the effectiveness of a solution. The following is a list of questions that addressed these issues:

* P- Performance. Does the system provide adequate throughput and response time?
* I- Information. Does the system provide end-users and managers with timely, pertinent, accurate and usefully formatted information .
* E- Economy. Does the system offer adequate service level and capacity to reduce the costs of the business or increase the profits of the business?
* C- Control. Does the system offer adequate controls to protect against fraud and embezzlement and to guarantee the accuracy and security of data and information?
* E- Efficiency. Does the system make maximum use of available resources including people, time, flow of forms, minimum processing details and the like?
* S- Services. Does the system provide desirable and reliable service to those who need it? Is the system flexible and expandable?

**HOW DO THE END-USERS/CUSTOMERS FEEL ABOUT THE SOLUTION?**

It’s important not only to evaluate whether a system can work but we must also evaluate whether a system will work. A workable solution might fail because of end-user and management resistance.

***COST BENEFITS ANALYSIS FOR NEW ESHOPPING WEBSITE***

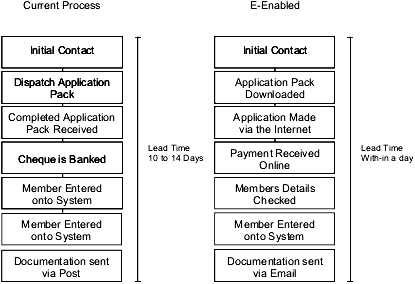
**OVERVIEW**

This case study will try and asses the feasibility for an ecommerce website.

**CHALLENGE**

The challenge in this case study is do the following

Analyze the costs and benefits associated with the implementation of an ecommerce website for a membership society.



**Fig. Membership Process**

Figure 1 illustrates typifies the processes of a generic membership system. You can see the effect e-enablement has on the current process on the right site of the illustration.

`

**Costs for technology**

To produce an e-shopping website requires a high speed connection to the Internet, a web server, and software. Other costs that are relevant is the cost of the payment system, whether it is taking online payment directly from the Societies web site or an alternative third-party like Pay pal or more expensively using an online bank.

**Costs for technological development**

This will involve a number of programmers who are able to interpret your functional requirements and program/create your website.

**Costs for the consultancy support (design and implementation)**

You would require the services of specialists in e-business design and implementation to guide you through this process.

**Costs for the organization for piloting training**

During the technological development of a website it is always a good idea to allow admin staff who will be using the system on a daily basis to pilot the system to as a training initiative.

**Running costs**

These are an upkeep of the web server and maintenance costs.

**Running costs for change process**

This is the cost of factoring in for your employees to train and adapting to the newly introduced technology, mainly the strategies used to make the change as smooth as possible.

Additionally being on the Internet would result in the your company having to become familiar to respond to emails, queries, and complaints that require instant or quick responses as opposed to replying to a Customer/Client via a letter. To be successful online, your company would have to address this issue of Change Management in that it would have to incorporate into its business, processes in order to guide the company to successfully maximise its effectiveness on the Internet. “Business is streamlined and service is almost instantaneous when it is done on the Web.

***FINANCIAL BENEFITS***

**Improve Cash flow**

Online payment would result in the lead-time to receive payments for membership with in the same day of the application being made rather than the average 14 day delay. E-enabling the membership process is not just about reengineering a process so that it is quicker than before, it would result in a complete overhaul of the previous way of managing membership. Figure 1 illustrates the current membership process on the left and the E-enabled equivalent. As you can see in the E-enabled application and payment are made online. Once the application has been completed the documentation is sent via email to the member adding value by cutting the waiting time. This also saves on printing and postage for the Society. Even if the member does not have an email address the documentation will be available for download for registered members to the web site.

**Increase Revenues**

The Internet will increase the volume of members. By going online with your business, you will generate revenue from places you never imagined”

***NON-FINANCIAL BENEFITS***

**Communication**

Direct email marketing incurs little or no cost compared to the traditional direct mail marketing. Direct email marketing allows the flexibility of sending the your companies message day or night, exactly when they want.

**Transparency**

The Information Management Website will allow the membership process to become transparent. For example, for the first time ever, Management will be able to know as a matter of fact:

* The total number of members
* Those members who need to renew their membership
* Those members who are in arrears with their membership fees
* Total number of members and accredited members
* Forecast for the expected revenue that will be generated in advance and look at historic monthly generated revenues.

**Exposure**

The Internet means that your company will become a global Business. Thus attracting potential members from internationally. “Using the Web to sell your products removes the physical boundaries from your customer base. Customers from all over the world can learn about and purchase your products online”

***PROJECT PLANNING***

Software project plan can be viewed as the following :

1. **Within the organization :** How the project is to be implemented ? What are various constraints (time, cost, staff)? What is market strategy?
2. **With respect to the customer :** Weekly or timely meetings with the customer with presentation on status reports. Customer feedback is also taken and further modification and developments are done. Project milestones and deliverables are also presented to the customer.

For successful software project, the following steps can be followed:

* **Select a project**
  + Identifying project’s aims and objectives
  + Understanding requirements and specification
  + Method of analysis, design and implementation
  + Testing techniques
  + Documentation
* **Project milestones and deliverables**
* **Budget Allocation**
* **Exceeding limits within control**
* **Project Estimates**
  + Cost
  + Time
  + Size of Code
  + Duratrion
* **Resource Allocation**
  + Hardware
  + Software
  + Previous relevant project information
  + Digital Shopping
* **Risk Management**
  + Risk Avoidance
  + Risk Detection

***ONLINE SHOPPING SYSTEM PROCESS WORKFLOW***

**Continue Shopping**

Clear Cart

Proceed To Checkout

**Confirmation Page**

Purchase Confirmation Details

Submit Purchase

Enter Personal Details

Purchase Calculations

**CheckOut Page**

Cart Items

Item 1

Item 2

Item 3

Item 4

**Cart Page**

**Category**

Category 1

Category 2

Category 3

Category 4

**Index Page**

View Cart

Add To Cart

**Product**

Product 1

Product 2

Product 3

Product 4

**Category Page**

***DATA FLOW DIAGRAM***

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That’s why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

**Symbols and Notations Used in DFDs**

Two common systems of symbols are named after their creators:

* Yourdon and Coad
* Yourdon and DeMarco
* Gane and Sarson

One main difference in their symbols is that Yourdon-Coad and Yourdon-DeMarco use circles for processes, while Gane and Sarson use rectangles with rounded corners, sometimes called lozenges. There are other symbol variations in use as well, so the important thing to keep in mind is to be clear and consistent in the shapes and notations you use to communicate and collaborate with others.

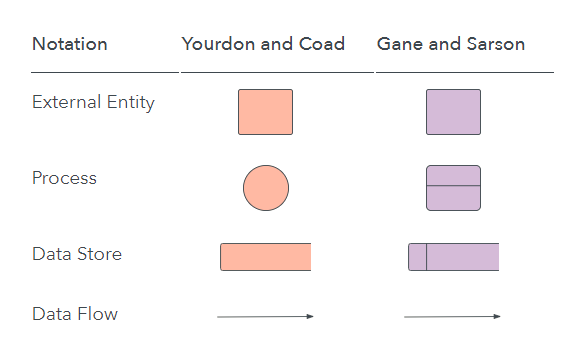
Using any convention’s DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

**External entity:** An outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

**Process:** Any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as “Submit payment.”

**Data store:** Files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as “Orders.”

**Data flow:** The route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.”

****

**DFD rules and tips**

* Each process should have at least one input and an output.
* Each data store should have at least one data flow in and one data flow out.
* Data stored in a system must go through a process.
* All processes in a DFD go to another process or a data store.

**DFD levels and layers: From context diagrams to pseudocode**

A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish.

* DFD Level 0 is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.
* DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its sub-processes.
* DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system’s functioning.
* Progression to Levels 3, 4 and beyond is possible, but going beyond Level 3 is uncommon. Doing so can create complexity that makes it difficult to communicate, compare or model effectively.

Using DFD layers, the cascading levels can be nested directly in the diagram, providing a cleaner look with easy access to the deeper dive.

By becoming sufficiently detailed in the DFD, developers and designers can use it to write pseudocode, which is a combination of English and the coding language.  Pseudocode facilitates the development of the actual code.

**Examples of how DFDs can be used**

Data flow diagrams are well suited for analysis or modeling of various types of systems in different fields.

**DFD in software engineering:** This is where data flow diagrams got their main start in the 1970s. DFDs can provide a focused approached to technical development, in which more research is done up front to get to coding.

**DFD in business analysis:** Business analysts use DFDs to analyze existing systems and find inefficiencies. Diagramming the process can uncover steps that might otherwise be missed or not fully understood.

**DFD in business process re-engineering:**  DFDs can be used to model a better, more efficient flow of data through a business process. BPR was pioneered in the 1990s to help organizations cut operational costs, improve customer service and better compete in the market.

**DFD in agile development:** DFDs can be used to visualize and understand business and technical requirements and plan the next steps. They can be a simple yet powerful tool for communication and collaboration to focus rapid development.

**DFD in system structures:** Any system or process can be analyzed in progressive detail to improve it, on both a technical and non-technical basis.

**DFD vs. Unified Modeling Language (UML)**

While a DFD illustrates how data flows through a system, UML is a modeling language used in Object Oriented Software Design to provide a more detailed view. A DFD may still provide a good starting point, but when actually developing the system, developers may turn to UML diagrams such as class diagrams and structure diagrams to achieve the required specificity.

**Logical DFD vs. Physical DFD**

These are the two categories of a data flow diagram. A Logical DFD visualizes the data flow that is essential for a business to operate. It focuses on the business and the information needed, not on how the system works or is proposed to work. However, a Physical DFD shows how the system is actually implemented now, or how it will be. For example, in a Logical DFD, the processes would be business activities, while in a Physical DFD, the processes would be programs and manual procedures.

**DFD Level 0**

Vendor Management

Place Order

User Details Management

Database

Cart Management

View Product

Product Management

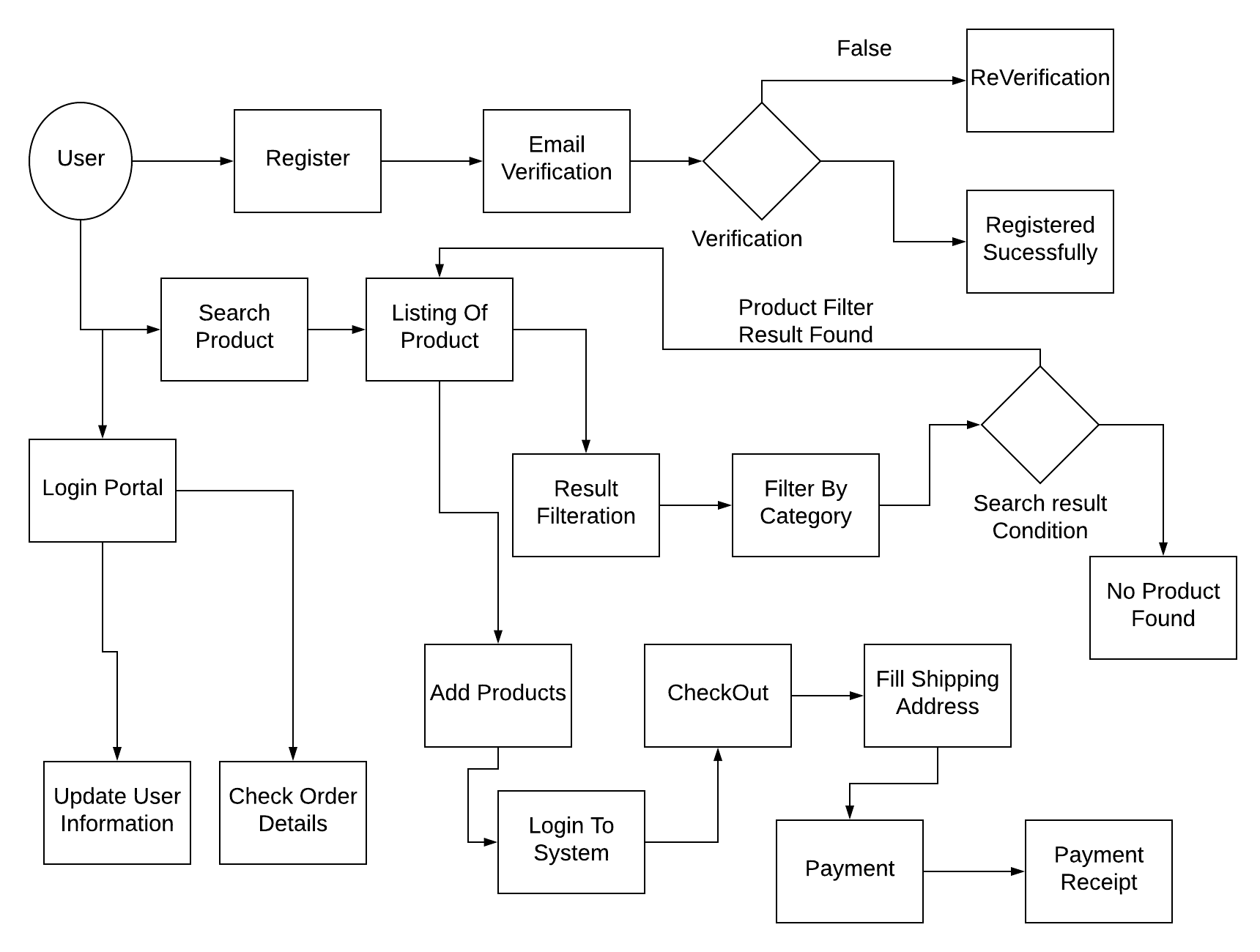
User Management

**E-commerce Portal Management**

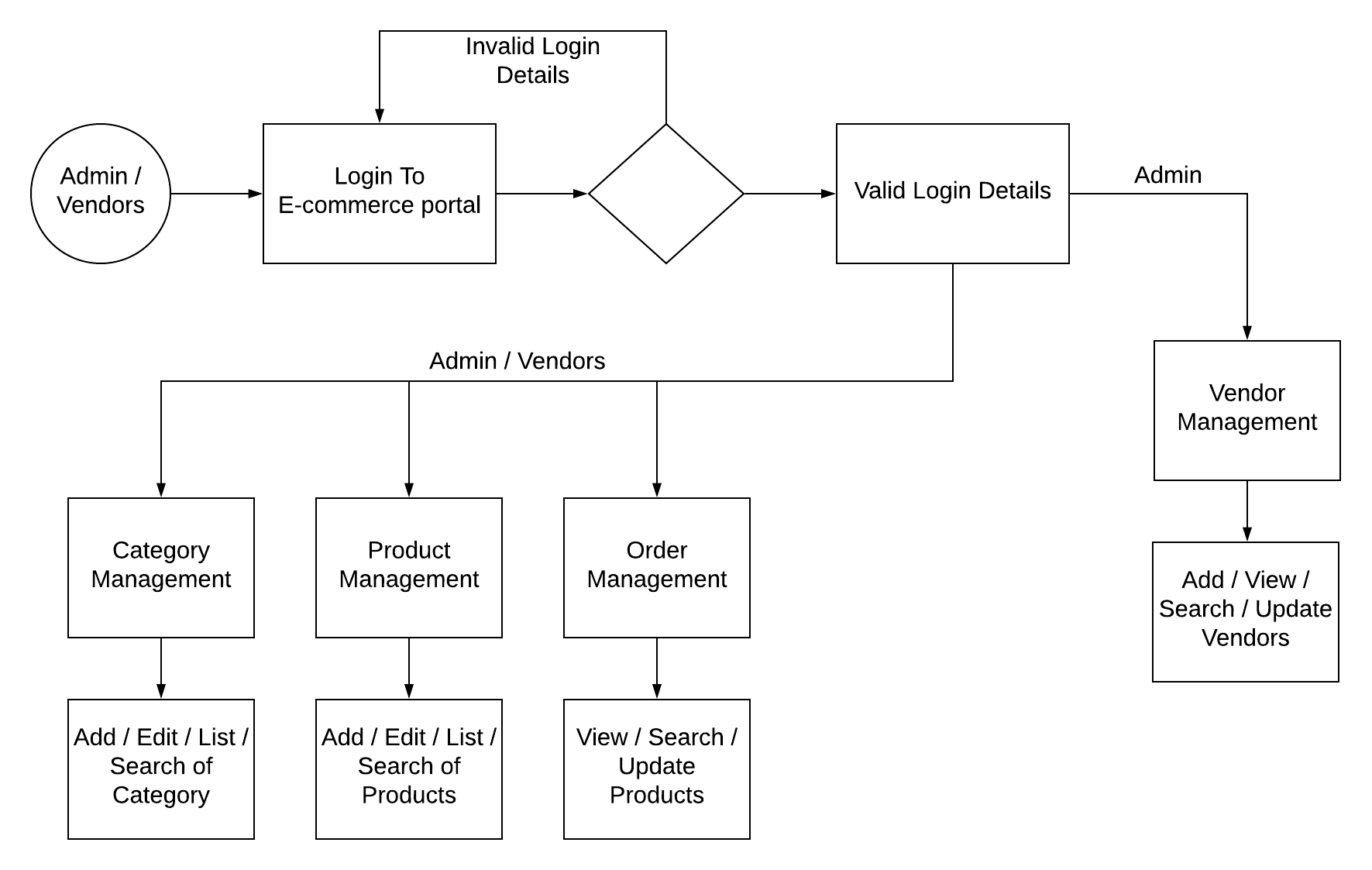
Admin/ Vendor

Users/ Customers

**DFD Level 1**

****

**DFD Level 2**

****

***ENTITY RELATIONSHIP DIAGRAM***

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves. ER diagrams also are often used in conjunction with data flow diagrams (DFDs), which map out the flow of information for processes or systems.

**Uses of entity relationship diagrams**

**Database design:** ER diagrams are used to model and design relational databases, in terms of logic and business rules (in a logical data model) and in terms of the specific technology to be implemented (in a physical data model.) In software engineering, an ER diagram is often an initial step in determining requirements for an information systems project. It’s also later used to model a particular database or databases. A relational database has an equivalent relational table and can potentially be expressed that way as needed.

**Database troubleshooting:** ER diagrams are used to analyze existing databases to find and resolve problems in logic or deployment. Drawing the diagram should reveal where it’s going wrong.

**Business information systems:** The diagrams are used to design or analyze relational databases used in business processes. Any business process that uses fielded data involving entities, actions and interplay can potentially benefit from a relational database. It can streamline processes, uncover information more easily and improve results.

**Business process re-engineering (BPR):** ER diagrams help in analyzing databases used in business process re-engineering and in modeling a new database setup.

**Education:** Databases are today’s method of storing relational information for educational purposes and later retrieval, so ER Diagrams can be valuable in planning those data structures.

**Research:** Since so much research focuses on structured data, ER diagrams can play a key role in setting up useful databases to analyze the data.

**The components and features of an ER diagram**

ER Diagrams are composed of entities, relationships and attributes. They also depict cardinality, which defines relationships in terms of numbers. Here’s a glossary:

**Entity**

A definable thing—such as a person, object, concept or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle.

ENTITY

**Entity type:** A group of definable things, such as students or athletes, whereas the entity would be the specific student or athlete. Other examples: customers, cars or products.

**Entity set:** Same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.

**Entity categories:** Entities are categorized as strong, weak or associative. A strong entity can be defined solely by its own attributes, while a weak entity cannot. An associative entity associates entities (or elements) within an entity set.

ASSOCIATIVE ENTITY

WEAK ENTITY

**Entity keys:** Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate or primary.

* **Super key:** A set of attributes (one or more) that together define an entity in an entity set.
* **Candidate key:** A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key.
* **Primary key:** A candidate key chosen by the database designer to uniquely identify the entity set. Foreign key: Identifies the relationship between entities.

**Relationship**

How entities act upon each other or are associated with each other. Think of relationships as verbs. For example, the named student might register for a course. The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way. Relationships are typically shown as diamonds or labels directly on the connecting lines.

WEAK RELATION-SHIP

RELATIONSHIP

**Recursive relationship:** The same entity participates more than once in the relationship.

**Attribute**

A property or characteristic of an entity. Often shown as an oval or circle.



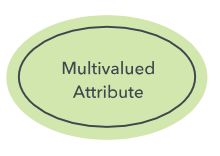
**Descriptive attribute:** A property or characteristic of a relationship (versus of an entity.)

**Attribute categories:** Attributes are categorized as simple, composite, derived, as well as single-value or multi-value.

* **Simple:** Means the attribute value is atomic and can’t be further divided, such as a phone number.
* **Composite:** Sub-attributes spring from an attribute.
* **Derived:** Attributed is calculated or otherwise derived from another attribute, such as age from a birthdate.

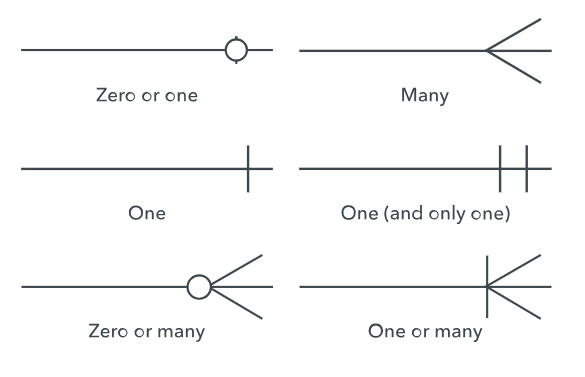


**Multi-value:** More than one attribute value is denoted, such as multiple phone numbers for a person.



**Single-value:** Just one attribute value. The types can be combined, such as: simple single-value attributes or composite multi-value attributes.

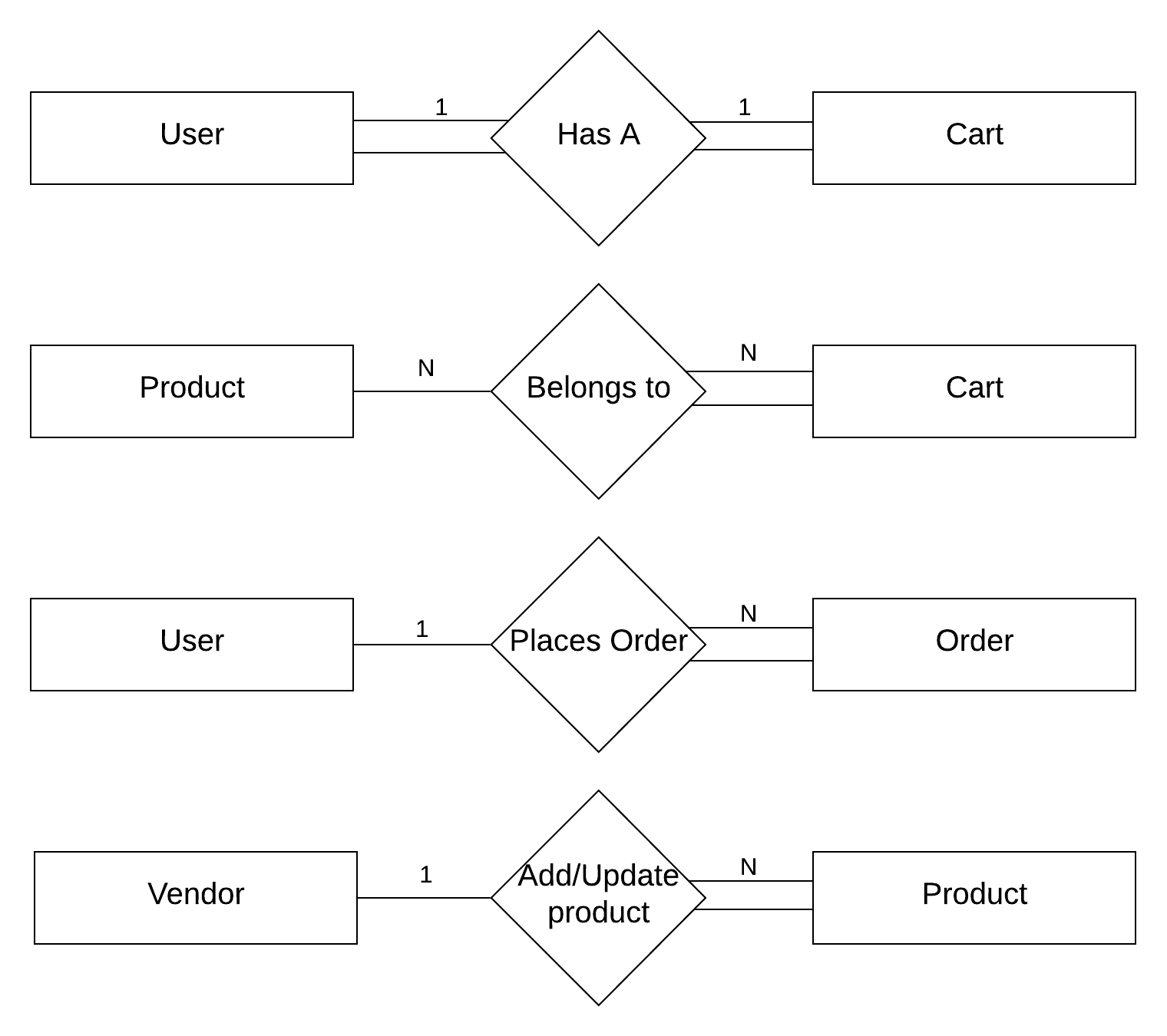
**Cardinality**

Defines the numerical attributes of the relationship between two entities or entity sets. The three main cardinal relationships are one-to-one, one-to-many, and many-many. A one-to-one example would be one student associated with one mailing address. A one-to-many example (or many-to-one, depending on the relationship direction): One student registers for multiple courses, but all those courses have a single line back to that one student. Many-to-many example: Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students 

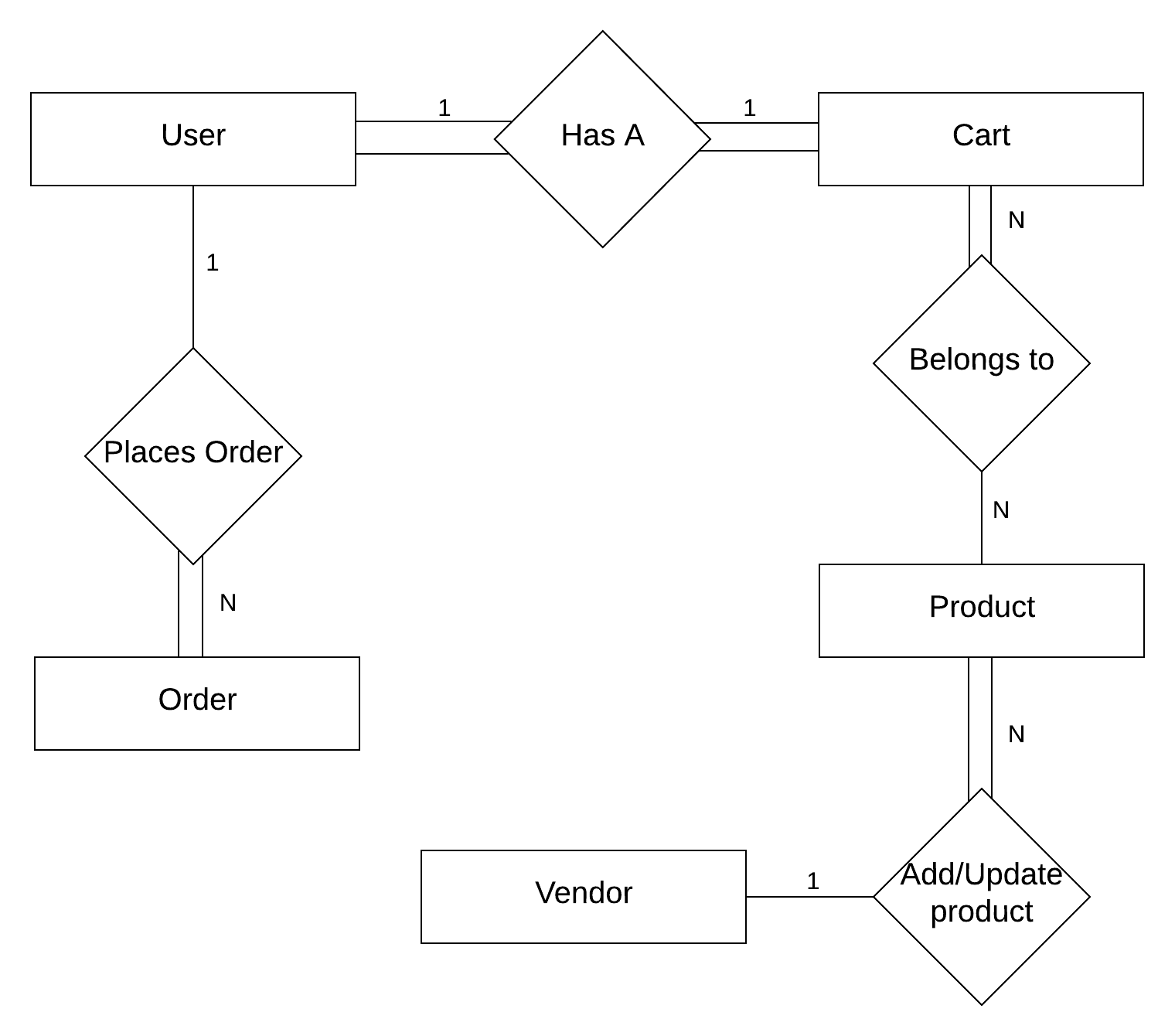
**Cardinality views:** Cardinality can be shown as look-across or same-side, depending on where the symbols are shown.

**Cardinality constraints:** The minimum or maximum numbers that apply to a relationship.

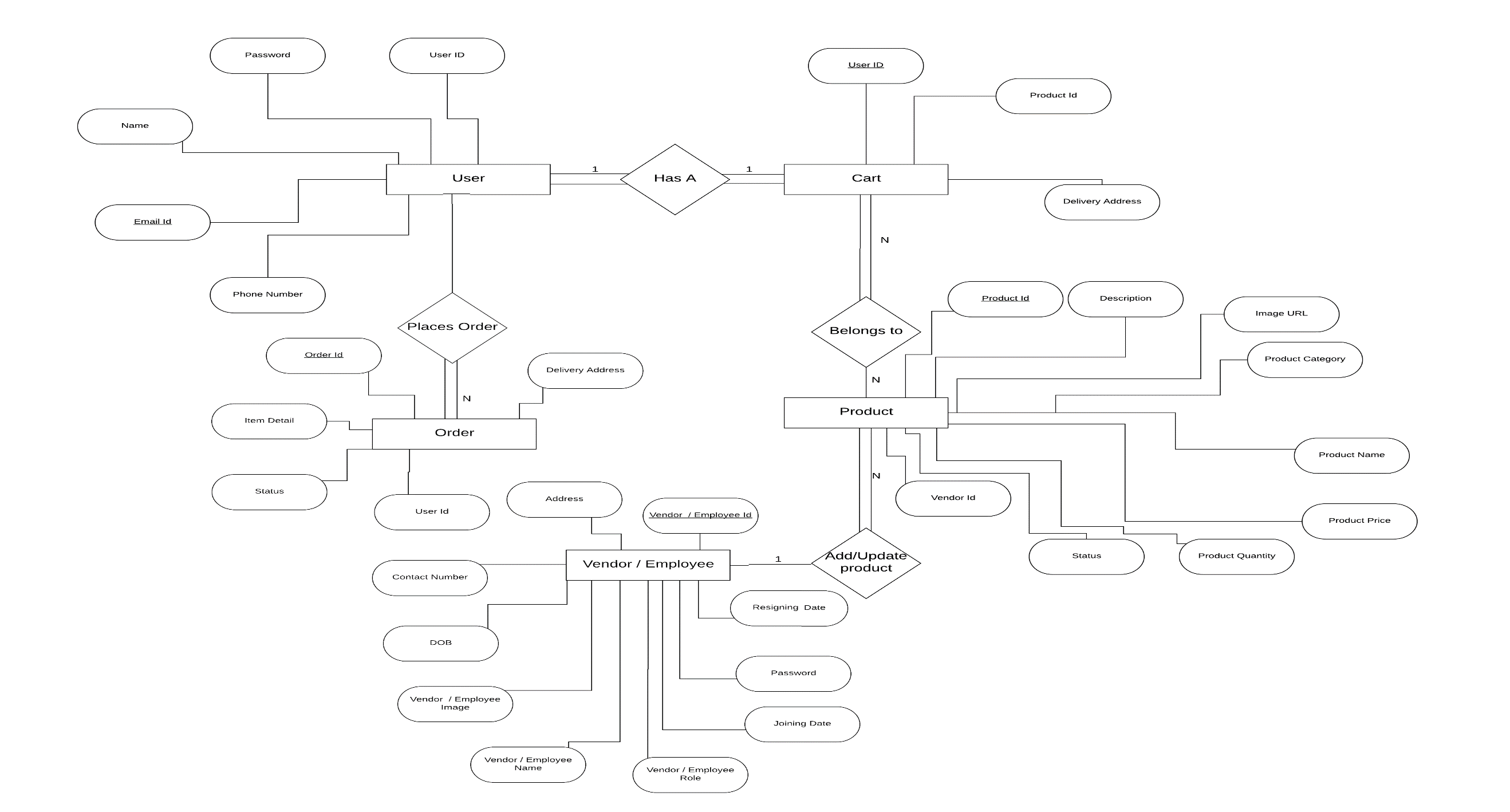
**Relationship Between Entities**



**Complete Structure of Relationship Between Entities**



**ER Diagram**



***TABLE STRUCTURE***

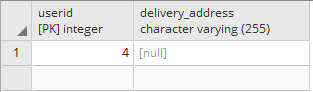
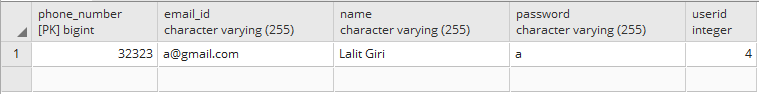
******

Table 1 Cart Details

****

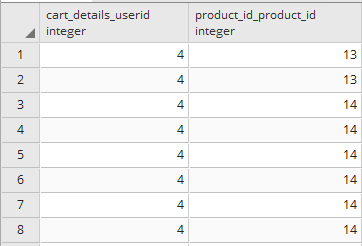
******

Table 3 Cart\_Details\_Product\_Id

Table 2 User Details Table

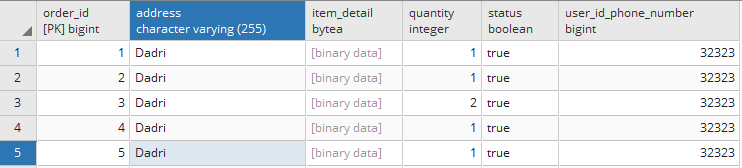
******

Table 4 Order Details Table

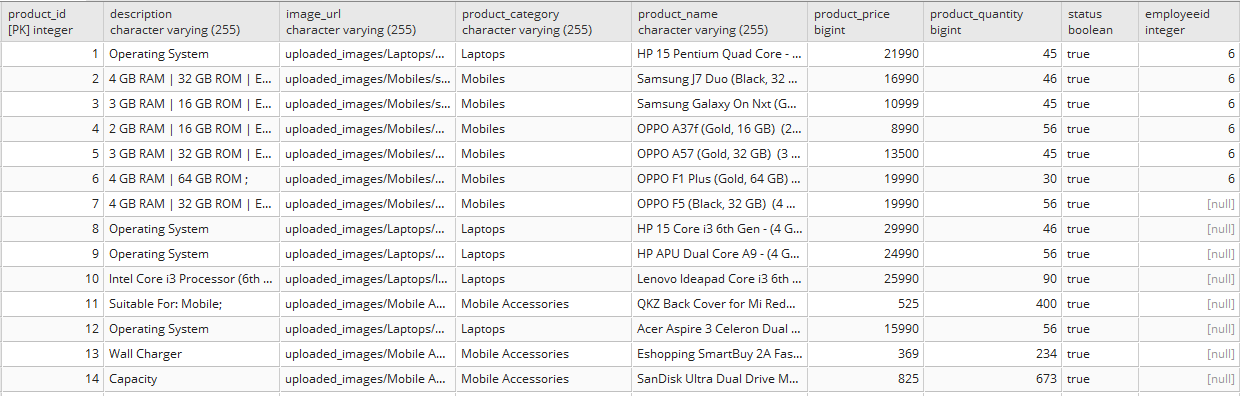
******

Table 5 Product Details Table

# ***SOFTWARE ENGINEERING AND SOFTWARE PARADIGMS***

The term "software engineering" was coined in about 1969 to mean "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines".

This view opposed uniqueness and "magic" of programming in an effort to move the development of software from "magic" (which only a select few can do) to "art" (which the talented can do) to "science" (which supposedly anyone can do!). There have been numerous definitions given for software engineering (including that above and below).

Software Engineering is not a discipline; it is an aspiration, as yet unachieved. Many approaches have been proposed including reusable components, formal methods, structured methods and architectural studies. These approaches chiefly emphasize the engineering product; the solution rather than the problem it solves.

Software Development current situation:

* People developing systems were consistently wrong in their estimates of time, effort, and costs
* Reliability and maintainability were difficult to achieve
* Delivered systems frequently did not work
* 1979 study of a small number of government projects showed that:
* 2% worked
* 3% could work after some corrections
* 45% delivered but never successfully used
* 20% used but extensively reworked or abandoned
* 30% paid and undelivered
* Fixing bugs in delivered software produced more bugs
* Increase in size of software systems
* NASA
* StarWars Defense Initiative
* Social Security Administration
* financial transaction systems
* Changes in the ratio of hardware to software costs
* early 60's - 80% hardware costs
* middle 60's - 40-50% software costs
* today - less than 20% hardware costs
* Increasingly important role of maintenance
* Fixing errors, modification, adding options
* Cost is often twice that of developing the software
* Advances in hardware (lower costs)
* Advances in software techniques (e.g., users interaction)
* Increased demands for software
* Medicine, Manufacturing, Entertainment, Publishing
* Demand for larger and more complex software systems
* Airplanes (crashes), NASA (aborted space shuttle launches),
* "ghost" trains, runaway missiles,
* ATM machines (have you had your card "swallowed"?), life-support systems, car systems, etc.
* US National security and day-to-day operations are highly dependent on computerized systems.

Manufacturing software can be characterized by a series of steps ranging from concept exploration to final retirement; this series of steps is generally referred to as a *software lifecycle*.

Steps or phases in a software lifecycle fall generally into these categories:

* Requirements (Relative Cost 2%)
* Specification (analysis) (Relative Cost 5%)
* Design (Relative Cost 6%)
* Implementation (Relative Cost 5%)
* Testing (Relative Cost 7%)
* Integration (Relative Cost 8%)
* Maintenance (Relative Cost 67%)
* Retirement

Software engineering employs a variety of methods, tools, and paradigms.

Paradigms refer to particular approaches or philosophies for designing, building and maintaining software. Different paradigms each have their own advantages and disadvantages which make one more appropriate in a given situation than perhaps another (!).

A method (also referred to as a technique) is heavily depended on a selected paradigm and may be seen as a procedure for producing some result. Methods generally involve some formal notation and processes.

Tools are automated systems implementing a particular method.

Thus, the following phases are heavily affected by selected software paradigms

* Design
* Implementation
* Integration
* Maintenance

The software development cycle involves the activities in the production of a software system. Generally the software development cycle can be divided into the following phases:

* Requirements analysis and specification
* Design
* Preliminary design
* Detailed design
* Implementation
* Component Implementation
* Component Integration
* System Documenting
* Testing
* Unit testing
* Integration testing
* System testing
* Installation and Acceptance Testing
* Maintenance
* Bug Reporting and Fixing
* Change requirements and software upgrading

Software lifecycles that will be briefly reviewed include:

1. Build and Fix model
2. Waterfall and Modified Waterfall models
3. Rapid Prototyping
4. Boehm's spiral model

## Build and Fix model

This works OK for small, simple systems, but is completely unsatisfactory for software systems of any size. It has been shown empirically that the cost of changing a software product is relatively small if the change is made at the requirements or design phases but grows large at later phases.

The cost of this process model is actually far greater than the cost of a properly specified and designed project. Maintenance can also be problematic in a software system developed under this scenario.

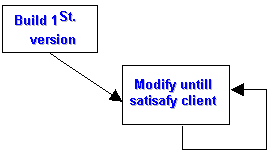


Figure: Build and Fix model

## Waterfall and Modified Waterfall models

### Waterfall Model

Derived from other engineering processes in 1970. Offered a means of making the development process more structured. Expresses the interaction between subsequent phases.

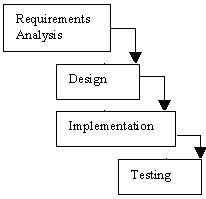


Figure: Waterfall model

Each phase cascades into the next phase. In the original waterfall model, a strict sequentially was at least implied. This meant that one phase had to be completed before the next phase was begun.

It also did not provide for feedback between phases or for updating/re-definition of earlier phases. Implies that there are definite breaks between phases, i.e., that each phase has a strict, non-overlapping start and finish and is carried out sequentially.

Critical point is that no phase is complete until the documentation and/or other products associated with that phase are completed.

### Modified Waterfall Model

Needed to provide for overlap and feedback between phases. Rather than being a simple linear model, it needed to be an iterative model. To facilitate the completion of the goals, milestones, and tasks, it is normal to freeze parts of the development after a certain point in the iteration. Verification and validation are added. Verification checks that the system is correct (building the system right). Validation checks that the system meets the users desires (building the right system).

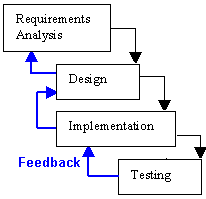


Figure: Modified Waterfall model

The waterfall model (and modified waterfall model) are inflexible in the partitioning of the project into distinct phases. However, they generally reflect engineering practice.

Considerable emphasis must be placed on discerning users' needs and requirements prior to the system being built. The identification of users' requirements as early as possible, and the agreement between user and developer with respect to those requirements, often is the deciding factor in the success or failure of a software project. These requirements are documented in the requirements specification, which is used to verify whether subsequent phases are complying with the requirements. Unfortunately specifying users' requirements is very much an art, and as such is extremely difficult. Validation feedback can be used to prevent the appearance of a strong divergence between the system under development and the users' expectations for the delivered system. Unfortunately, the waterfall lifecycle (and the modified waterfall lifecycle) are inadequate for realistic validation activities. They are exclusively document driven models. The resulting design reality is that only 50% of the design effort occurs during the actual design phase with 1/3 of the design effort occurring during the coding activity! This is topped by the fact that over 16% of the design effort occurs after the system is supposed to be completed! In general the behavior of many individuals in this type of process is opportunistic. The boundaries of phases are indiscriminately crossed with deadlines being somewhat arbitrary.

## Rapid Prototyping

Prototyping also referred to as evolutionary development, prototyping aims to enhance the accuracy of the designer's perception of the user's requirements. Prototyping is based on the idea of developing an initial implementation for user feedback, and then refining this prototype through many versions until an satisfactory system emerges. The specification, development and validation activities are carried out concurrently with rapid feedback across the activities. Generally, prototyping is characterized by the use of very high-level languages, which probably will not be used in the final software implementation but which allow rapid development, and the development of a system with less functionality with respect to quality attributes such as robustness, speed, etc.



Figure: Rapid Prototyping model

Prototyping allows the clarification of users requirements through, particularly, the early development of the user interface. The user can then try out the system, albeit a (sub) system of what will be the final product. This allows the user to provide feedback before a large investment has been made in the development of the wrong system.

There are two types of prototypes:

* **Exploratory programming:** Objective is to work with the user to explore their requirements and deliver a final system. Starts with the parts of the system which are understood, and then evolves as the user proposes new features.
* **Throw-away prototyping:** Objective is to understand the users' requirements and develop a better requirements definition for the system. Concentrates on poorly understood components.

Experiments with prototyping showed that this approach took 40% less time and resulted in 45% less code; however, it produced code which was not as robust, and therefore more difficult to maintain. Documentation was often sacrificed or done incompletely. The schedule expectations of users and managers tended to be unrealistic especially with respect to throw-away prototypes.

## Boehm’s Spiral Model

Need an improved software lifecycle model which can subsume all the generic models discussed so far. Must also satisfy the requirements of management.

Boehm proposed a spiral model where each round of the spiral

1. identifies the sub problem which has the highest risk associated with it
2. finds a solution for that problem.

***SYSTEM REQUIREMENTS AND SPECIFICATIONS***

Once the system analyst have identified the precise user requirements and analyzed these requirements to weed out in consistencies, they proceed to write the document called the Software Requirement Specification (SRS). It is the final output of the requirements analysis and specification phase. An SRS document should clearly document the following:

• Functional requirements of the system

• Non-functional requirements of the system

• Constraints of the system

A SRS should have characteristics like concise, unambiguous, consistent, complete, well-structured etc.

***SOFTWARE REQUIREMENTS***

**Platform** : Web Based

**Framework** : SpringBoot, Hibernate, Angular 4

**Front-End** : HTML, CSS, BootStrap4

**Back-End Tool(DB)** : Postgre Sql

**Ide Tool** : Visual Studio Code, Spring Tool Suit

**Build Tool** : Maven

***REASON FOR OPTING ANGULAR 4, SPRING HIBERNATE FRAMEWOK AND MAVEN***

**Angular 4** is a JavaScript framework for building web applications and apps in JavaScript, html, and TypeScript, which is a superset of JavaScript. Angular provides built-in features for animation, http service, and materials which in turn has features such as auto-complete, navigation, toolbar, menus, etc. The code is written in TypeScript, which compiles to JavaScript and displays the same in the browser.

Angular JS is an open source framework built over JavaScript. It was built by the developers at Google. This framework was used to overcome obstacles encountered while working with Single Page applications. Also, testing was considered as a key aspect while building the framework. It was ensured that the framework could be easily tested. The initial release of the framework was in October 2010.

There are three major releases of Angular. The first version that was released is Angular1, which is also called AngularJS. Angular1 was followed by Angular2, which came in with a lot of changes when compared to Angular1.

The structure of Angular is based on the components/services architecture. AngularJS was based on the model view controller. Angular 4 released in March 2017 proves to be a major breakthrough and is the latest release from the Angular team after Angular2.

Angular 4 is almost the same as Angular 2. It has a backward compatibility with Angular 2. Projects developed in Angular 2 will work without any issues with Angular 4.

**Spring** framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly. Spring help to make the layered architecture of the application that makes the modules of application loosely coupled.

Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

* **SpringBoot** is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework.
* It is used to create stand alone spring based application that you can just run because it needs very little spring configuration.
* Spring Boot does not generate code and there is absolutely no requirement for XML configuration.
* It uses convention over configuration software design paradigm that means it decrease the effort of developer.

**Advantages of Spring Boot**

* Create stand-alone Spring applications that can be started using java -jar.
* Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.
* It provides opinionated 'starter' POMs to simplify your Maven configuration.
* It automatically configure Spring whenever possible.
* It provides production-ready features such as metrics, health checks and externalized configuration.
* Absolutely no code generation and no requirement for XML configuration.

**Hibernate** is an Object-Relational Mapping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.



**Hibernate Advantages**

* Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
* Provides simple APIs for storing and retrieving Java objects directly to and from the database.
* If there is change in the database or in any table, then you need to change the XML file properties only.
* Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
* Hibernate does not require an application server to operate.
* Manipulates Complex associations of objects of your database.
* Minimizes database access with smart fetching strategies.
* Provides simple querying of data.

**Maven** is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following −

* Builds
* Documentation
* Reporting
* Dependencies
* SCMs
* Releases
* Distribution
* Mailing list

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of the build related tasks.

***SYSTEM DESIGN***

System design is the solution to the creation of a new system. This phase is composed of several systems. This phase focuses on the detailed implementation of the feasible system. It emphasis on translating design specifications to performance specification. System design has two phases of development logical and physical design.

During logical design phase the analyst describes inputs (sources), out puts (destinations), databases (data sores) and procedures (data flows) all in a format that meats the uses requirements. The analyst also specifies the user needs and at a level that virtually determines the information flow into and out of the system and the data resources. Here the logical design is done through data flow diagrams and database design.

The physical design is followed by physical design or coding. Physical design produces the working system by defining the design specifications, which tell the programmers exactly what the candidate system must do. The programmers write the necessary programs that accept input from the user, perform necessary processing on accepted data through call and produce the required report on a hard copy or display it on the screen.

**LOGICAL DESIGN :**

Logical design of an information system shows the major features and also how they are related to one another. The first step of the system design is to design logical design elements. This is the most creative and challenging phase and important too. Design of proposed system produces the details of the state how the system will meet the requirements identified during the system analysis that is, in the design phase we have to find how to solve the difficulties faced by the existing system. The logical design of the proposed system should include the details that contain how the solutions can be implemented. It also specifies how the database is to be built for storing and retrieving data, what kind of reports are to be created and what are the inputs to be given to the system. The logical design includes input design, output design, and database design and physical design.

**INPUT DESIGN**

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data into a usable form for processing data entry. The activity of putting data into the computer for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling errors, avoiding delay, avoiding extra steps and keeping the process simple.

The system needs the data regarding the asset items, depreciation rates, asset transfer and physical verification for various validation, checking, calculation and record generation. The error raising method is also included in the software, which helps to raise error message while wrong entry of input is done. So in input design the following things are considered.

What data should be given as input?

* How the data should be arranged or coded?
* The dialogue to guide the operating personnel in providing input.
* Methods for preparing input validations and steps to follow when error occur.
* The samples of screen layout are given in the appendix.

**OUTPUT DESIGN**

Computer output is the most important and direct information source to the user. Output design is a process that involves designing necessary outputs in the form of reports that should be given to the users according to the requirements. Efficient, intelligible output design should improve the system's relationship with the user and help in decision making. Since the records are directing referred by the management for taking decisions and to draw conclusions they must be designed with almost care and the details in the records must be simple, descriptive and clear to the user. So while designing output the following things are to be considered.

* Determine what information to present.
* Arrange the presentation of information in an acceptable format.
* Decide how to distribute the output to intended receipts.

Depending on the nature and future use of output required, they can be displayed on the monitor for immediate need. All screens are informative and interactive in such a way that the user can full fill his requirements through asking queries.

**PHYSICAL DESIGN**

The process of developing the program software is referred to as physical design. We have to design the process by identifying reports and the other outputs the system will produce. Coding the program for each module with its logic is performed in this step. Proper software specification is also done in this step.

**MODULAR DESIGN**

A software system is always divided into several sub systems that makes it easier for the development. A software system that is structured into several subsystems makes it easy for the development and testing. The different subsystems are known as the modules and the process of dividing an entire system into subsystems is known as modularization or decomposition.

A system cannot be decomposed into several subsystems in any way. There must some logical barrier, which facilitates the separation of each module. The separation must be simple but yet must be effective so that the development is not affected.

The system under consideration has been divided into several modules taking in consideration the above-mentioned criteria.

***CODING***

**Controllers**

1. **CartDetailsServiceController.java**

package com.ecommerce.main.controllers;

import java.util.List;

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

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.ecommerce.main.dao.CartDetails;

import com.ecommerce.main.dao.ProductTable;

import com.ecommerce.main.service.CartDetailsService;

@RestController

public class CartDetailsServiceController {

@Autowired

private CartDetailsService cartDetailsService;

@RequestMapping("token/getcart/{id}")

public String getCartDetails(@PathVariable int id){

return cartDetailsService.getCartDetails(id);

}

@RequestMapping(method=RequestMethod.POST,value="token/updatecart/{id}/{quantity}")

public String updateCartDetails(@PathVariable int id,@PathVariable int quantity,@RequestBody ProductTable product) {

String str=cartDetailsService.updateCartDetails(product, id,quantity);

return str;

}

@RequestMapping("token/getallcartsdetails")

public List<CartDetails> getAllCartDetails(){

return cartDetailsService.getAllCartDetails();

}

@RequestMapping(method=RequestMethod.POST,value="token/removefromcart/{id}")

public String removeItemFromCart(@RequestBody ProductTable product,@PathVariable int id) {

return cartDetailsService.removeItemFromCart(product, id);

}

}

1. **EmployeeDetailsServiceController.java**

package com.ecommerce.main.controllers;

import java.util.List;

import java.util.Optional;

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

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.RestController;

import com.ecommerce.main.dao.EmployeeDetails;

import com.ecommerce.main.service.EmployeeDetailsService;

@RestController

public class EmployeeDetailsServiceController {

@Autowired

private EmployeeDetailsService employeeDetailsService;

@RequestMapping(method=RequestMethod.POST,value="/addemployee")

public @ResponseBody String addEmployeeDetails(@RequestBody EmployeeDetails employee) throws Exception{

return employeeDetailsService.addEmployeeDetails(employee);

}

@RequestMapping("/deleteemployee/{id}")

public String deleteEmployeeDetails(@PathVariable int id) {

return employeeDetailsService.deleteEmployeeDetails(id);

}

@RequestMapping("getemployeedetail/{id}")

public Optional<EmployeeDetails> getEmployeeDetails(int id){

return employeeDetailsService.getEmployeeDetails(id);

}

@RequestMapping(method=RequestMethod.POST,value="/updateemployee/{id}")

public String updateEmployeeDetails(@RequestBody EmployeeDetails employee,int id) {

return employeeDetailsService.updateEmployeeDetails(employee, id);

}

@RequestMapping("/getallemployee")

public List<EmployeeDetails> getAllEmployeeDetails(){

return employeeDetailsService.getAllEmployeeDetails();

}

}

1. **OrderDetailsServiceController.java**

package com.ecommerce.main.controllers;

import java.util.List;

import java.util.Optional;

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

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.ecommerce.main.cartHandler.CartOrderHandlerSupportClass;

import com.ecommerce.main.dao.OrderDetails;

import com.ecommerce.main.service.OrderDetailsService;

@RestController

public class OrderDetailsServiceController {

@Autowired

private OrderDetailsService orderDetailsService;

@RequestMapping(method=RequestMethod.POST,value="/add\_cart\_order")

public String addCartOrder(@RequestBody CartOrderHandlerSupportClass order) {

return orderDetailsService.addCartOrder(order);

}

@RequestMapping(method=RequestMethod.POST,value="/addorder")

public String addOrder(@RequestBody OrderDetails order) {

return orderDetailsService.addOrder(order);

}

@RequestMapping("/getallorder")

public List<OrderDetails> getAllOrder(){

return orderDetailsService.getAllOrder();

}

@RequestMapping("/deleteorder/{id}")

public String deleteOrder(@PathVariable int id) {

return orderDetailsService.deleteOrder(id);

}

@RequestMapping("/getorderbyid/{id}")

public Optional<OrderDetails> getOrderById(@PathVariable int id){

return orderDetailsService.getOrderById(id);

}

@RequestMapping(method=RequestMethod.POST,value="/updateAddress/{id}")

public String updateDeliveryAddress(@RequestBody OrderDetails order,@PathVariable int id) {

return orderDetailsService.updateDeliveryAddress(order, id);

}

}

1. **ProductTableServiceController.java**

package com.ecommerce.main.controllers;

import java.util.List;

import java.util.Optional;

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

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.ecommerce.main.dao.ProductTable;

import com.ecommerce.main.service.ProductTableService;

@RestController

public class ProductTableServiceController {

@Autowired

private ProductTableService productTableService;

@RequestMapping(method=RequestMethod.POST,value="/addproduct/{employeeId}")

public String addProduct(@RequestBody ProductTable product,@PathVariable int employeeId) throws Exception {

return productTableService.addProduct(product, employeeId);

}

@RequestMapping(method=RequestMethod.POST,value="/deleteproduct")

public String deleteProduct(@RequestBody ProductTable id[]){

return productTableService.deleteProduct(id);

}

@RequestMapping("/getproduct/{id}")

public Optional<ProductTable> getProductDetails(@PathVariable int id) {

return productTableService.getProductDetails(id);

}

@RequestMapping(method=RequestMethod.POST,value="/updateproduct/{id}")

public String updateProductDetails(@RequestBody ProductTable product,@PathVariable long id) {

return productTableService.updateProductDetails(product, id);

}

@RequestMapping("/getallproduct")

public List<ProductTable> getAllProducts(){

return productTableService.getAllProducts();

}

@RequestMapping("/getallproductCategory")

public List<String> getAllProductCategory(){

return productTableService.getAllProductCategory();

}

@RequestMapping("/getproductbycategory/{category}")

public List<ProductTable> getProductByCategory(@PathVariable String category)

{

return productTableService.getProductByCategory(category);

}

@RequestMapping(value="/searchProduct/{category}/{name}",method=RequestMethod.GET)

public Iterable<ProductTable> getByProductNameAndCategory(@PathVariable String category,@PathVariable String name){

return productTableService.getByProductNameAndCategory(category.toUpperCase(), name.toUpperCase());

}

}

1. **UserDetailsServiceController.java**

**package com.ecommerce.main.controllers;**

**import java.util.List;**

**import java.util.Optional;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

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

**import org.springframework.web.bind.annotation.PathVariable;**

**import org.springframework.web.bind.annotation.RequestBody;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestMethod;**

**import org.springframework.web.bind.annotation.RestController;**

**import com.ecommerce.main.dao.UserDetails;**

**import com.ecommerce.main.interceptors.TokenProvider;**

**import com.ecommerce.main.interceptors.User;**

**import com.ecommerce.main.interceptors.UserService;**

**import com.ecommerce.main.service.UserDetailsService;**

**@RestController**

**public class UserDetailsServiceController {**

**@Autowired**

**private UserService userService;**

**@Autowired**

**private UserDetailsService userDetailsService;**

**@RequestMapping(method = RequestMethod.POST, value = "/adduser")**

**public String addUserDetails(@RequestBody UserDetails newUser,HttpServletRequest request, HttpServletResponse response) throws Exception {**

**String message= userDetailsService.addUserDetails(newUser);**

**if(message.equals("Sucessfully Added")) {**

**User user=new User();**

**String emailId = newUser.getEmailId();**

**String password = newUser.getPassword();**

**user.setRole("User");**

**user.setEmailAddress(emailId);**

**user.setId(password);**

**user.setUserName(newUser.getName());**

**user.setPhoneNumber(newUser.getPhoneNumber());**

**user.setUserId(newUser.getCart().getUserId());**

**userService.addUser(user);**

**request.getSession(true).setAttribute("user", user);**

**TokenProvider tokenProvider =new TokenProvider();**

**String token = tokenProvider.generate(user);**

**return token;**

**}**

**return message;**

**}**

**@RequestMapping("/deleteuser/{id}")**

**public String deleteUserDetails(@PathVariable long id) {**

**return userDetailsService.deleteUserDetails(id);**

**}**

**@RequestMapping("/getuser/{id}")**

**public UserDetails getUserDetails(@PathVariable long id) {**

**//System.out.println("hello");**

**Optional<UserDetails> user = userDetailsService.getUserDetails(id);**

**if(user.isPresent()) {**

**return user.get();**

**}**

**return null;**

**}**

**@RequestMapping(method = RequestMethod.POST, value = "/updateuser/{id}")**

**public String updateUserDetails(@RequestBody UserDetails user, @PathVariable long id) {**

**return userDetailsService.updateUserDetails(user, id);**

**}**

**@RequestMapping("/getalluser")**

**public List<UserDetails> getAllUserDetails() {**

**return userDetailsService.getAllUserDetails();**

**}**

**}**

**Service**

1. **CartDetailsService.java**

package com.ecommerce.main.service;

import java.util.List;

import com.ecommerce.main.dao.CartDetails;

import com.ecommerce.main.dao.ProductTable;

public interface CartDetailsService {

public String getCartDetails(int id);

public String updateCartDetails(ProductTable product,int id,int quantity) ;

public List<CartDetails> getAllCartDetails();

public String removeItemFromCart(ProductTable product,int id);

}

1. **EmployeeDetailsService.java**

package com.ecommerce.main.service;

import java.util.List;

import java.util.Optional;

import com.ecommerce.main.dao.EmployeeDetails;

public interface EmployeeDetailsService {

public String addEmployeeDetails(EmployeeDetails employee) throws Exception ;

public String deleteEmployeeDetails(int id);

public Optional<EmployeeDetails> getEmployeeDetails(int id) ;

public String updateEmployeeDetails(EmployeeDetails employee,int id) ;

public List<EmployeeDetails> getAllEmployeeDetails();

public EmployeeDetails userAuthentication(String userName, String password);

}

1. **OrderDetailsService.java**

package com.ecommerce.main.service;

import java.util.List;

import java.util.Optional;

import com.ecommerce.main.cartHandler.CartOrderHandlerSupportClass;

import com.ecommerce.main.dao.OrderDetails;

public interface OrderDetailsService {

public String addOrder(OrderDetails order);

public List<OrderDetails> getAllOrder();

public String deleteOrder(int id);

public Optional<OrderDetails> getOrderById(int id);

public String updateDeliveryAddress(OrderDetails order,int id);//this is only for to update address...

public String addCartOrder(CartOrderHandlerSupportClass order);

}

1. **ProductTableService.java**

package com.ecommerce.main.service;

import java.util.List;

import java.util.Optional;

import com.ecommerce.main.dao.ProductTable;

public interface ProductTableService {

public String addProduct(ProductTable product,int employeeId) throws Exception ;

public String deleteProduct(ProductTable id[]);

public Optional<ProductTable> getProductDetails(int id);

public String updateProductDetails(ProductTable product,long id) ;

public List<ProductTable> getAllProducts();

public List<String> getAllProductCategory();

public List<ProductTable> getProductByCategory(String category);

public Iterable<ProductTable> getByProductNameAndCategory(String category,String name);

}

1. **UserDetailsService.java**

package com.ecommerce.main.service;

import java.util.List;

import java.util.Optional;

import com.ecommerce.main.dao.UserDetails;

public interface UserDetailsService {

public String addUserDetails(UserDetails user) throws Exception ;

public String deleteUserDetails(long id);

public Optional<UserDetails> getUserDetails(long id);

public String updateUserDetails(UserDetails user,long id);

public List<UserDetails> getAllUserDetails();

public UserDetails userAuthentication(String emailId,String password);

}

**Service Implementation**

1. CartDetailsServiceImpl.java

package com.ecommerce.main.serviceImpl;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Optional;

import java.util.Set;

import java.util.stream.Collectors;

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

import org.springframework.stereotype.Service;

import com.ecommerce.main.cartHandler.CartOrderHandlerSupportClass;

import com.ecommerce.main.cartHandler.ProductQuantity;

import com.ecommerce.main.dao.CartDetails;

import com.ecommerce.main.dao.ProductTable;

import com.ecommerce.main.repository.CartDetailsRepository;

import com.ecommerce.main.service.CartDetailsService;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

@Service

public class CartDetailsServiceImpl implements CartDetailsService {

@Autowired

private CartDetailsRepository cartDetailsRepository;

CartDetails cart = new CartDetails();

List<ProductTable> productList = new ArrayList<>();

public String getCartDetails(int id) {

String jsonResult = null;

CartOrderHandlerSupportClass cartOrderHandlerSupportClass = new CartOrderHandlerSupportClass();

List<ProductQuantity> productQuantity = new ArrayList<ProductQuantity>();

if (cartDetailsRepository.findById(id).get() != null) {

CartDetails cartDetails = cartDetailsRepository.findById(id).get();

List<ProductTable> list = cartDetails.getProductId();

Set<ProductTable> uniqueProducts = new HashSet<>();

list.forEach(p -> uniqueProducts.add(p));

List<ProductTable> uniqueIds = uniqueProducts.stream().collect(Collectors.toList());

for (ProductTable i : uniqueIds) {

int count = 0;

for (ProductTable p : list) {

if (p.equals(i)) {

count++;

}

}

productQuantity.add(new ProductQuantity(count, i));

}

cartOrderHandlerSupportClass.setProductQuantity(productQuantity);

cartOrderHandlerSupportClass.setDeliveryAddress(cartDetails.getDeliveryAddress());

cartOrderHandlerSupportClass.setUserId(cartDetails.getUserId());

ObjectMapper mapper = new ObjectMapper();

try {

jsonResult = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(cartOrderHandlerSupportClass);

} catch (JsonProcessingException e) {

e.printStackTrace();

}

return jsonResult;

}

return "hello";

}

public String updateCartDetails(ProductTable product, int id, int quantity) {

boolean flag=false;

for (int i = 0; i < quantity; i++) {

if (product != null) {

cart.setUserId(id);

if (addItemToCart(product, id))

cart.setProductId(productList);

cart.setDeliveryAddress(null);

if (cartDetailsRepository.save(cart) != null)

flag=true;

else

{

flag=false;

break;

}

}

}

if(flag==true)

return "SucessFully Updated";

return "error";

}

public boolean addItemToCart(ProductTable product, int id) {

productList = new ArrayList<>();

Optional<CartDetails> cart2 = cartDetailsRepository.findById(id);

if (cart2.isPresent()) {

productList = cart2.get().getProductId();

productList.add(product);

return true;

}

return false;

}

public List<CartDetails> getAllCartDetails() {

List<CartDetails> cartList = new ArrayList<>();

cartDetailsRepository.findAll().forEach(cartList::add);

return cartList;

}

public String removeItemFromCart(ProductTable product, int id) {

Optional<CartDetails> cart2 = cartDetailsRepository.findById(id);

if (cart2.isPresent()) {

productList = cart2.get().getProductId();

if (removeItemLocal(product)) {

cart2.get().setProductId(productList);

cartDetailsRepository.save(cart2.get());

return "itemRemoved";

}

return "item not removed";

}

return "Item Not Avaliable In Cart";

}

public boolean removeItemLocal(ProductTable product) {

List<ProductTable> list = new ArrayList<>();

productList.forEach(u -> {

if (u.getProductId() != product.getProductId()) {

list.add(u);

}

});

if (list != null) {

productList = list;

return true;

}

return false;

}

}

1. **EmployeeDetailsServiceImpl**

package com.ecommerce.main.serviceImpl;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.ecommerce.main.dao.EmployeeDetails;

import com.ecommerce.main.repository.EmployeeDetailRepository;

import com.ecommerce.main.service.EmployeeDetailsService;

@Service

public class EmployeeDetailsServiceImpl implements EmployeeDetailsService {

@Autowired

private EmployeeDetailRepository employeeDetailsReposiotory;

public String addEmployeeDetails(EmployeeDetails employee) throws Exception {

int i = employeeDetailsReposiotory.CountRow();

employee.setEmployeeId(i);

employeeDetailsReposiotory.save(employee);

return "Sucessfully Added";

}

public String deleteEmployeeDetails(int id) {

if (employeeDetailsReposiotory.existsById(id)) {

//employeeDetailsReposiotory.deleteById(id);

employeeDetailsReposiotory.ChangeStatus(id);

return "SucessFully Deleted";

}

return "Employee NotFound";

}

public Optional<EmployeeDetails> getEmployeeDetails(int id) {

return employeeDetailsReposiotory.findById(id);

}

public String updateEmployeeDetails(EmployeeDetails employee, int id) {

if (employeeDetailsReposiotory.existsById(id)) {

employeeDetailsReposiotory.save(employee);

return "SucessFully Updated";

}

return "Illegal Modification";

}

public List<EmployeeDetails> getAllEmployeeDetails() {

List<EmployeeDetails> employeeList = new ArrayList<>();

employeeDetailsReposiotory.findAll().forEach(employeeList::add);

if (employeeList == null)

return null;

return employeeList;

}

@Override

public EmployeeDetails userAuthentication(String userName, String password) {

EmployeeDetails user = employeeDetailsReposiotory.findByemployeeNameAndPassword(userName, password);

if (user != null)

return user;

return null;

}

}

1. **OrderDetailsServiceImpl.java**

package com.ecommerce.main.serviceImpl;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.ecommerce.main.cartHandler.CartOrderHandlerSupportClass;

import com.ecommerce.main.dao.OrderDetails;

import com.ecommerce.main.dao.UserDetails;

import com.ecommerce.main.repository.OrderDetailsRepository;

import com.ecommerce.main.repository.UserDetailsRepository;

import com.ecommerce.main.service.OrderDetailsService;

@Service

public class OrderDetailsServiceImpl implements OrderDetailsService {

@Autowired

private OrderDetailsRepository orderDetailsRepository;

@Autowired

private UserDetailsRepository userDetailsRepository;

@Override

public String addOrder(OrderDetails order) {

long c = orderDetailsRepository.count()+1;

order.setOrderId(c);

order.setStatus(true);

orderDetailsRepository.save(order);

return "Order Placed Sucessfully";

}

@Override

public List<OrderDetails> getAllOrder() {

List <OrderDetails> orderList = new ArrayList<OrderDetails>();

orderDetailsRepository.findAll().forEach(orderList::add);

if(orderList==null)

return null;

return orderList;

}

@Override

public String deleteOrder(int id) {

orderDetailsRepository.deleteById(id);

return "Sucessfully Deleted";

}

@Override

public Optional<OrderDetails> getOrderById(int id) {

return orderDetailsRepository.findById(id);

}

@Override

public String updateDeliveryAddress(OrderDetails order, int id) {

if(orderDetailsRepository.existsById(id))

{

orderDetailsRepository.save(order);

return "SucessFully Updated";

}

return "Illegal Modification";

}

@Override

public String addCartOrder(CartOrderHandlerSupportClass order) {

order.getProductQuantity().forEach(u -> {

OrderDetails orderDetails = new OrderDetails();

// System.out.println("User Id:" +order.getUserId());

orderDetails.setUserId(userDetailsRepository.findById((long)order.getUserId()).get());

orderDetails.setAddress(order.getDeliveryAddress());

orderDetails.setStatus(true);

long c = orderDetailsRepository.count()+1;

orderDetails.setOrderId(c);

orderDetails.setQuantity(u.getQuantity());

orderDetails.setItemDetail(u.getProductTable());

orderDetailsRepository.save(orderDetails);

});

return "Sucessfully Order Placed !!!";

}

}

1. **ProductTableServiceImpl.java**

package com.ecommerce.main.serviceImpl;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.ecommerce.main.dao.ProductTable;

import com.ecommerce.main.repository.EmployeeDetailRepository;

import com.ecommerce.main.repository.ProductTableRepository;

import com.ecommerce.main.service.ProductTableService;

@Service

public class ProductTableServiceImpl implements ProductTableService {

@Autowired

private ProductTableRepository productTableRepository;

@Autowired

private EmployeeDetailRepository employeeDetailRepository;

public String addProduct(ProductTable product,int employeeId) throws Exception {

int i=productTableRepository.CountRow();

product.setProductId(i+1);

product.setEmployeeId(employeeDetailRepository.findById(employeeId).get());

//System.out.println("i="+i+" i+1="+(i+1));

productTableRepository.save(product);

return "Sucessfully Added";

}

public String deleteProduct(ProductTable id[]){

for(ProductTable i : id) {

try{if(productTableRepository.existsById(i.getProductId()))

{

i.setStatus(false);

productTableRepository.save(i);

}

}catch(Exception e) {

return "Error Deleting item with id=" + i;

}

}

return "Sucessfully Deleted";

}

public Optional<ProductTable> getProductDetails(int id) {

return productTableRepository.findById(id);

}

public String updateProductDetails(ProductTable product,long id) {

productTableRepository.save(product);

return "SucessFully Updated";

}

public List<ProductTable> getAllProducts(){

List <ProductTable> productList = new ArrayList<>();

productTableRepository.findByStatus().

forEach(productList::add);

if(productList==null)

return null;

return productList;

}

public List<String> getAllProductCategory(){

/\*List<ProductTable> productList = (List<ProductTable>) productTableRepository.findAll();

List<String> categories = new ArrayList<>();

productList.forEach(p -> {

if(!categories.contains(p.getProductCategory())) {

categories.add(p.getProductCategory());

}

});

return categories;\*/

return productTableRepository.findCategory();

}

@Override

public List<ProductTable> getProductByCategory(String category) {

return productTableRepository.findByproductCategory(category);

}

public Iterable<ProductTable> getByProductNameAndCategory(String category,String name){

return productTableRepository.findByProductNameWithCategory(category,name);

}

}

1. **UserDetailsServiceImpl.java**

package com.ecommerce.main.serviceImpl;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import javax.servlet.http.HttpSession;

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

import org.springframework.stereotype.Service;

import com.ecommerce.main.dao.CartDetails;

import com.ecommerce.main.dao.UserDetails;

import com.ecommerce.main.repository.UserDetailsRepository;

import com.ecommerce.main.service.UserDetailsService;

@Service

public class UserDetailsServiceImpl implements UserDetailsService {

@Autowired

private UserDetailsRepository userDetailsRepository;

public String addUserDetails(UserDetails user) throws Exception {

CartDetails cart=new CartDetails();

int i=userDetailsRepository.CountRow();

cart.setUserId(i+1);

user.setCart(cart);

user.setOrderId(null);

userDetailsRepository.save(user);

return "Sucessfully Added";

}

public String deleteUserDetails(long id){

if(userDetailsRepository.existsById(id)) {

userDetailsRepository.deleteById(id);

return "SucessFully Deleted";

}

return "Illegal Operation";

}

public Optional<UserDetails> getUserDetails(long id) {

// System.out.println("hye"+id);

return userDetailsRepository.findById(id);

}

public String updateUserDetails(UserDetails user,long id) {

if(userDetailsRepository.existsById(id)) {

userDetailsRepository.save(user);

return "SucessFully Updated";

}

return "Illegal Operation";

}

public List<UserDetails> getAllUserDetails(){

List <UserDetails> userList = new ArrayList<>();

userDetailsRepository.findAll().

forEach(userList::add);

if(userList== null)

return null;

return userList;

}

public UserDetails userAuthentication(String emailId,String password) {

UserDetails user=userDetailsRepository.findByemailIdAndPassword(emailId,password);

if(user!=null)

return user;

return null;

}

}

**Data Access Object**

1. **CartDetails.java**

package com.ecommerce.main.dao;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

@Entity

public class CartDetails implements Serializable {

@Id

@Column(name = "UserID")

private int userId; //primaryKey

@ManyToMany

private List<ProductTable> productId =new ArrayList<>(); //foreginKey

private String deliveryAddress;

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public String getDeliveryAddress() {

return deliveryAddress;

}

public void setDeliveryAddress(String deliveryAddress) {

this.deliveryAddress = deliveryAddress;

}

public List<ProductTable> getProductId() {

return productId;

}

public void setProductId(List<ProductTable> productId) {

this.productId = productId;

}

}

1. **EmployeeDetails.java**

package com.ecommerce.main.dao;

import java.io.Serializable;

import java.sql.Date;

import java.util.Collection;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import org.hibernate.annotations.Fetch;

import org.hibernate.annotations.FetchMode;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity

public class EmployeeDetails implements Serializable {

@Id

private int employeeId; //primary Key

private String employeeName;

private String employeeRole;

private long contactNumber;

private String address;

private Date dob;

private String password;

private String empImageUrl;

private boolean status;

private Date joiningDate,resigningDate;

@JsonIgnore

@OneToMany(mappedBy="employeeId",fetch=FetchType.LAZY)

@Fetch(FetchMode.SELECT)

private Collection <ProductTable> productId;

public Collection<ProductTable> getProductId() {

return productId;

}

public void setProductId(Collection<ProductTable> productId) {

this.productId = productId;

}

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

public String getEmployeeName() {

return employeeName;

}

public void setEmployeeName(String employeeName) {

this.employeeName = employeeName;

}

public String getEmployeeRole() {

return employeeRole;

}

public void setEmployeeRole(String employeeRole) {

this.employeeRole = employeeRole;

}

public long getContactNumber() {

return contactNumber;

}

public void setContactNumber(long contactNumber) {

this.contactNumber = contactNumber;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Date getJoiningDate() {

return joiningDate;

}

public void setJoiningDate(Date joiningDate) {

this.joiningDate = joiningDate;

}

public Date getResigningDate() {

return resigningDate;

}

public void setResigningDate(Date resigningDate) {

this.resigningDate = resigningDate;

}

public boolean isStatus() {

return status;

}

public void setStatus(boolean status) {

this.status = status;

}

public String getEmpImageUrl() {

return empImageUrl;

}

public void setEmpImageUrl(String empImageUrl) {

this.empImageUrl = empImageUrl;

}

public Date getDob() {

return dob;

}

public void setDob(Date dob) {

this.dob = dob;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

1. **OrderDetails.java**

package com.ecommerce.main.dao;

import java.io.Serializable;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.ManyToOne;

@Entity

public class OrderDetails implements Serializable {

@Id

private long orderId; //primaryKey

private String address;

private ProductTable itemDetail;

private int quantity ;

private boolean status;

@ManyToOne//(fetch=FetchType.EAGER)

private UserDetails userId;

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public long getOrderId() {

return orderId;

}

public void setOrderId(long orderId) {

this.orderId = orderId;

}

public UserDetails getUserId() {

return userId;

}

public void setUserId(UserDetails userId) {

this.userId = userId;

}

public boolean isStatus() {

return status;

}

public void setStatus(boolean status) {

this.status = status;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public ProductTable getItemDetail() {

return itemDetail;

}

public void setItemDetail(ProductTable itemDetail) {

this.itemDetail = itemDetail;

}

}

1. **ProductTable.java**

package com.ecommerce.main.dao;

import java.io.Serializable;

import java.util.List;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToMany;

import javax.persistence.ManyToOne;

import javax.persistence.Transient;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity

public class ProductTable implements Serializable{

private static final long serialVersionUID = 1L;

@Id

private int productId; //primaryKey

private String productName;

private String productCategory;

private long productQuantity;

private long productPrice;

private String description;

private String imageUrl;

private boolean status;

@ManyToOne

@JoinColumn(name = "EmployeeID")

private EmployeeDetails employeeId; //foreginKey

@Transient

@ManyToMany(mappedBy="productId")

private List<CartDetails> cartId;

public List<CartDetails> getCartId() {

return cartId;

}

public void setCartId(List<CartDetails> cartId) {

this.cartId = cartId;

}

public int getProductId() {

return productId;

}

public void setProductId(int productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getProductCategory() {

return productCategory;

}

public void setProductCategory(String productCategory) {

this.productCategory = productCategory;

}

public long getProductQuantity() {

return productQuantity;

}

public void setProductQuantity(long productQuantity) {

this.productQuantity = productQuantity;

}

public long getProductPrice() {

return productPrice;

}

public void setProductPrice(long productPrice) {

this.productPrice = productPrice;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

public EmployeeDetails getEmployeeId() {

return employeeId;

}

public void setEmployeeId(EmployeeDetails employeeId) {

this.employeeId = employeeId;

}

public boolean isStatus() {

return status;

}

public void setStatus(boolean status) {

this.status = status;

}

}

1. **UserDetails.java**

package com.ecommerce.main.dao;

import java.io.Serializable;

import java.util.Collection;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OneToMany;

import javax.persistence.OneToOne;

import javax.persistence.Transient;

@Entity

public class UserDetails implements Serializable {

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "UserID") //uniqueKey

private CartDetails cart;

@Id

private long phoneNumber; //primaryKey

private String name;

private String emailId;

private String password;

@OneToMany(mappedBy="userId")

@Transient

private Collection<OrderDetails> orderId;

public Collection<OrderDetails> getOrderId() {

return orderId;

}

public void setOrderId(Collection<OrderDetails> orderId) {

this.orderId = orderId;

}

public CartDetails getCart() {

return cart;

}

public void setCart(CartDetails cart) {

this.cart = cart;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmailId() {

return emailId;

}

public void setEmailId(String emailId) {

this.emailId = emailId;

}

public long getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(long phoneNumber) {

this.phoneNumber = phoneNumber;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

**Repository**

1. **CartDetailsRepository.java**

package com.ecommerce.main.repository;

import org.springframework.data.repository.CrudRepository;

import com.ecommerce.main.dao.CartDetails;

public interface CartDetailsRepository extends CrudRepository<CartDetails,Integer> {

}

1. **EmployeeDetailsRepository.java**

package com.ecommerce.main.repository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.CrudRepository;

import org.springframework.data.repository.query.Param;

import com.ecommerce.main.dao.EmployeeDetails;

public interface EmployeeDetailRepository extends CrudRepository<EmployeeDetails,Integer>{

EmployeeDetails findByemployeeNameAndPassword(String UserName, String password);

public static final String COUNT\_ROWS = "SELECT count(\*) FROM employee\_details";

@Query(value = COUNT\_ROWS, nativeQuery = true)

public int CountRow();

public static final String CHANGE\_STATUS = "update employee\_details set p.status='False' where p.id = :id";

@Query(value =CHANGE\_STATUS, nativeQuery = true)

public int ChangeStatus(@Param("id") int id);

}

1. **OrderDetailsRepository.java**

package com.ecommerce.main.repository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.CrudRepository;

import com.ecommerce.main.dao.OrderDetails;

public interface OrderDetailsRepository extends CrudRepository<OrderDetails,Integer>{

public static final String COUNT\_ROWS = "SELECT count(product\_id) FROM Order\_Details";

@Query(value = COUNT\_ROWS, nativeQuery = true)

public int CountRow();

}

1. **ProductTableRepository.java**

package com.ecommerce.main.repository;

import java.util.List;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.CrudRepository;

import org.springframework.data.repository.query.Param;

import com.ecommerce.main.dao.ProductTable;

public interface ProductTableRepository extends CrudRepository<ProductTable,Integer>{

public static final String FIND\_CATEGORY = "SELECT DISTINCT product\_category FROM product\_table";

@Query(value = FIND\_CATEGORY, nativeQuery = true)

public List<String> findCategory();

public List<ProductTable> findByproductCategory(String category);

public static final String COUNT\_ROWS = "SELECT count(product\_id) FROM product\_table";

@Query(value = COUNT\_ROWS, nativeQuery = true)

public int CountRow();

public static final String FIND\_BY\_STATUS = "SELECT \* FROM product\_table where status=true";

@Query(value =FIND\_BY\_STATUS, nativeQuery = true)

public Iterable<ProductTable> findByStatus();

@Query(value="select \* from Product\_Table u where UPPER(u.product\_category) = :category AND UPPER(u.product\_name) LIKE CONCAT('%',:productName,'%')", nativeQuery = true)

public Iterable<ProductTable> findByProductNameWithCategory(@Param("category") String category, @Param("productName") String productName);

}

1. **UserDetailsRepository.java**

package com.ecommerce.main.repository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.CrudRepository;

import com.ecommerce.main.dao.UserDetails;

public interface UserDetailsRepository extends CrudRepository<UserDetails, Long> {

public UserDetails findByemailIdAndPassword(String emailId,String password);

public UserDetails findByemailId(String emailId);

public static final String COUNT\_ROWS = "SELECT count(\*) FROM user\_details";

@Query(value = COUNT\_ROWS, nativeQuery = true)

public int CountRow();

}

***VALIDATION CRITERIA***

The validation criteria in this project are as follows…

In “ZuperMarket” ( e-Shopping ) system, the user inputs are validated before storing them, and then after validation sent for other processes like displaying etc.

All the screens have a similar look and feel. They all have the almost same color combinations in its background. This provides a better user interface to the users.

The main validators in ZuperMarket system are as follows –

1. The primary key values cannot be duplicated.
2. All the entries in any combo box have been sorted in alphabetical order. This helps a user while selecting a value from the combo box.

**Importance of Testing**

During the systems testing, the system is used experimentally to ensure that the software does not fail. In other words, we can say that it will run according to its specifications and in the way users expect. Special test data are input for processing, and the results examined.

The importance of system testing is that the system is expected to run according to member’s requirement before delivering it to the customer.

The System is tested on the basis of specification so that it does not fail on user site.

***HUMAN RESOURCES***

The most important factor for any organization to keep abreast their project is to build a good employee management relationship. Working on the internal and external environment needs heavy amount of work to be performed in this sector. The resources for hiring good employees is categories as follows Local Newspaper and Media will be use to advertise. Such media aswww.rozee.pk and [www.brightspree.com](http://www.brightspree.com) for keeping track of the project Such Designation will be required

1) Network Administrator

2) Database Administrator

3) Web base Designers and Developers

4) Production Departmentt

5) Strategist

***IMPACT ON THE BUSINESS***

Impact of business is entirely depend on its internal and external factors. It is the aim of the organization to cover all their weaknesses with strength and take available opportunities and remove threats. Once the project completed it the entirely the Strategist to work on the factor will help improvising necessary terminologies /technologies in order to be effective. The following points illustrate the data we used to collect information in the system analysis phase.

* Learn all we can from existing documents, forms, reports and databases.
* If appropriate we will observe the system in action. We agree not to ask questions but just take notes and draw pictures. We must make sure that the workers know we are not evaluating individuals. Otherwise they may tend to work more efficiently than they usually do.
* Given all the facts we have already collected. We will design and distribute questionnaires to clear things up we don’t fully understand. This is also a good time to solicit opinions on problems and limitations. We understand that questionnaires do take much of the end-user time but they must know when its best to make that sacrifice.
* Conduct our interviews to verify and clarify the most important and difficult issues and problems.
* Follow up. We will use appropriate fact-finding techniques to verify facts, usually through interviews and observation.

***SYSTEM IMPLEMENTATION***

Implementation is the stage in the project where the theoretical design is turned into a working system. The implementation phase constructs, installs and operates the new system. The most crucial stage in achieving a new successful system is that it will work efficiently and effectively.

There are several activities involved while implementing a new project. They are

* Detailed Design Of Implementation
* Technical Design
* Programming And Testing
* End user training
* End user Education
* Training on the application software
* System Design
* Parallel Run and To New System
* Development Recap
* Post implementation Review

**Detailed Design Of Implementation:**

This phase of the systems development life cycle refines hardware and software specifications, establishes programming plans, trains users and operating specifications and/or provide the basis for further modification.

**Technical Design:**

This activity builds upon specifications produced during new system design, adding detailed technical specifications and documentation.

**Programming And Testing:**

This activity encompasses actual development, writing and testing of program units or modules.

**End user Training:**

The successful implementation of the new system will purely upon the involvement of the officers working in that department. The officers will be imparted the necessary training on the new technology

**End User Education:**

The education of the end user start after the implementation and testing is over. When the system is found to be more difficult to understand and complex, more effort is put to educate the end used to make them aware of the system, giving them lectures about the new system and providing them necessary documents and materials about how the system can do this.

**Training of application software:**

After providing the necessary basic training on the computer awareness, the users will have to be trained upon the new system such as the screen flows and screen design type of help on the screen, type of errors while entering the data, the corresponding validation check at each entry and the way to correct the data entered. It should then cover information needed by the specific user or group to use the system.

**Development Recap:**

A review of a project immediately after completion to find successes and potential problems in future work.

**Post Implementation View:**

A review, conducted after a new system has been in operation for some time, to evaluate actual system performance against original expectations and projections for cost-benefit improvements. Also identifies maintenance projects to enhance or improve the system.

The department is planning a method to know the states of the past implementation process. For that regular meeting will be arranged by the concerned officers about the implementation problem and success.

***SOFTWARE TESTING***

Software testing is a process of executing a program or application with the intent of finding the software bugs.

It can also be stated as the process of validating and verifying that a software program or application or product:

* Meets the business and technical requirements that guided it’s design and development
* Works as expected
* Can be implemented with the same characteristic.

Let’s break the definition of Software testing into the following parts:

1. Process: Testing is a process rather than a single activity.
2. All Life Cycle Activities: Testing is a process that’s take place throughout the Software Development Life Cycle (SDLC).

* The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it’s referred as “verifying the test basis via the test design”.
* The test basis includes documents such as the requirements and design specifications.
* **Static Testing:** It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing. For example: reviewing, walkthrough, inspection, etc.
* **Dynamic Testing:**  In dynamic testing the software code is executed to demonstrate the result of running tests. It’s done during validation process. For example: unit testing, integration testing, system testing, etc.
* **Planning:** We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.
* **Preparation:**  We need to choose what testing we will do, by selecting test conditions and designing test cases.
* **Evaluation:** During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.
* **Software products and related work products:** Along with the testing of code the testing of requirement and design specifications and also the related documents like operation, user and training material is equally important.

**The Steps In The Software Testing**

The steps involved during Unit testing are as follows :

1. Preparation of the test cases.
2. Preparation of the possible test data with all validation checks.
3. Complete code review of the module.
4. Actual testing done manually.
5. Modifications done for the errors found during testing.
6. Prepared the test result scripts.

**The unit testing done included the testing of the following items :**

1. Functionality of the entire module/forms.
2. Validators for user input.
3. Checking of the coding standards to be maintained during coding.
4. Testing the module with all possible test data.
5. Testing of the functionality involving all type of calculations etc.
6. Commenting standard in the source files.

After completing the Unit Testing of all the modules, the whole system is integrated with all its dependencies in that module. While System Integration, we integrated the modules one by one and tested the system at each step. This helped in reduction of errors at the time of the system testing.

**The steps involved during System testing are as follows:**

* Integration of all the modules/forms in the system.
* Preparation of the test cases.
* Preparation of the possible test data with all the validation checks.
* Actual testing done manually.
* Recording of all the reproduced errors.
* Modifications done for the errors found during testing.
* Prepared the test result scripts after rectification of the errors.

**The System Testing done included the testing of the following items:**

1. Functionality of the entire system as a whole.
2. User Interface of the system.
3. Testing the dependent modules together with all the possible test data scripts.
4. Verification and Validation testing.
5. Testing the reports with all its functionality.

After the completion of system testing, the next following phase was the Acceptance Testing. Clients at their end did this and accepted the system with appreciation.

Thus, we reached the final phase of the project delivery.

There are other six tests, which fall under special category. They are described below:

1. **Peak Load Test:** It determines whether the system will handle the volume of activities that occur when the system is at the peak of its processing demand. For example, test the system by activating all terminals at same time.
2. **Storage Testing:** It determines the capacity of the system to store transaction data on a disk or in other files.
3. **Performance Time Testing:** It determines the length of time system used by the system to process transaction data. This test is conducted prior to implementation to determine how long it takes to get a response to an inquiry, make a backup copy of a file, or send a transmission and get a response.
4. **Recovery Testing:** This testing determines the ability of user to recover data or re-start system after failure. For example, load backup copy of data and resume processing without or integrity loss.
5. **Procedure Testing:** It determines the clarity of documentation on operation and uses of system by having users do exactly what manuals request. For example, powering down system at the end of week or responding to paper-out light on printer.
6. **Human Factors Testing:** It determines how users will use the system when processing data or preparing reports.

***COST ESTIMATION OF THE PROJECT***

Software cost comprises a small percentage of overall computer- based system cost. There are a number of factors, which are considered, that can affect the ultimate cost of the software such as- human, technical, hardware and software availability etc.

The main point that was considered during the cost estimation of **project** was its sizing. In spite of complete software sizing, function point and approximate lines of code were also used to “size” each element of the Software and their costing.

The cost estimation done by me for **Project** also depend upon the baseline metrics collected from past projects and these were used in conjunction with estimation variables to develop cost and effort projections.

We have basically estimated this project mainly on two bases –

1. **Effort Estimation :** This refers to the total man-hours required for the development of the project. It even includes the time required for doing documentation and user manual.
2. **Hardware Required Estimation :**  This includes the cost of the PCs and the hardware cost required for development of this project.

***SYSTEM SECURITY MEASURES***

Security is the most important part of any system. It can be either the security of system program functionalities or underlying database. We have very cautious process of authentication of user that no one could change its contents in unauthorized manner. Security and integrity of database are very important for any software system because databases are the backbone of the system. Security need to be implemented at every level of the system so that only authorized user can access the system for updation and other significance process. Entering correct password while opening the system or we can say that entering the system is the process of authentication. If anyone is entering the password is wrong then he/she cannot access the system for any change purpose. The main purpose of the security is to save system from accidentally changes or loss of information or also getting wrong information. The system administrator is the person that can change the information or update the information. He can also grant the permission that who has to enter the system and what can he do. So security is the most important topic to be concerned.

***VARIOUS TYPES OF MODULES***

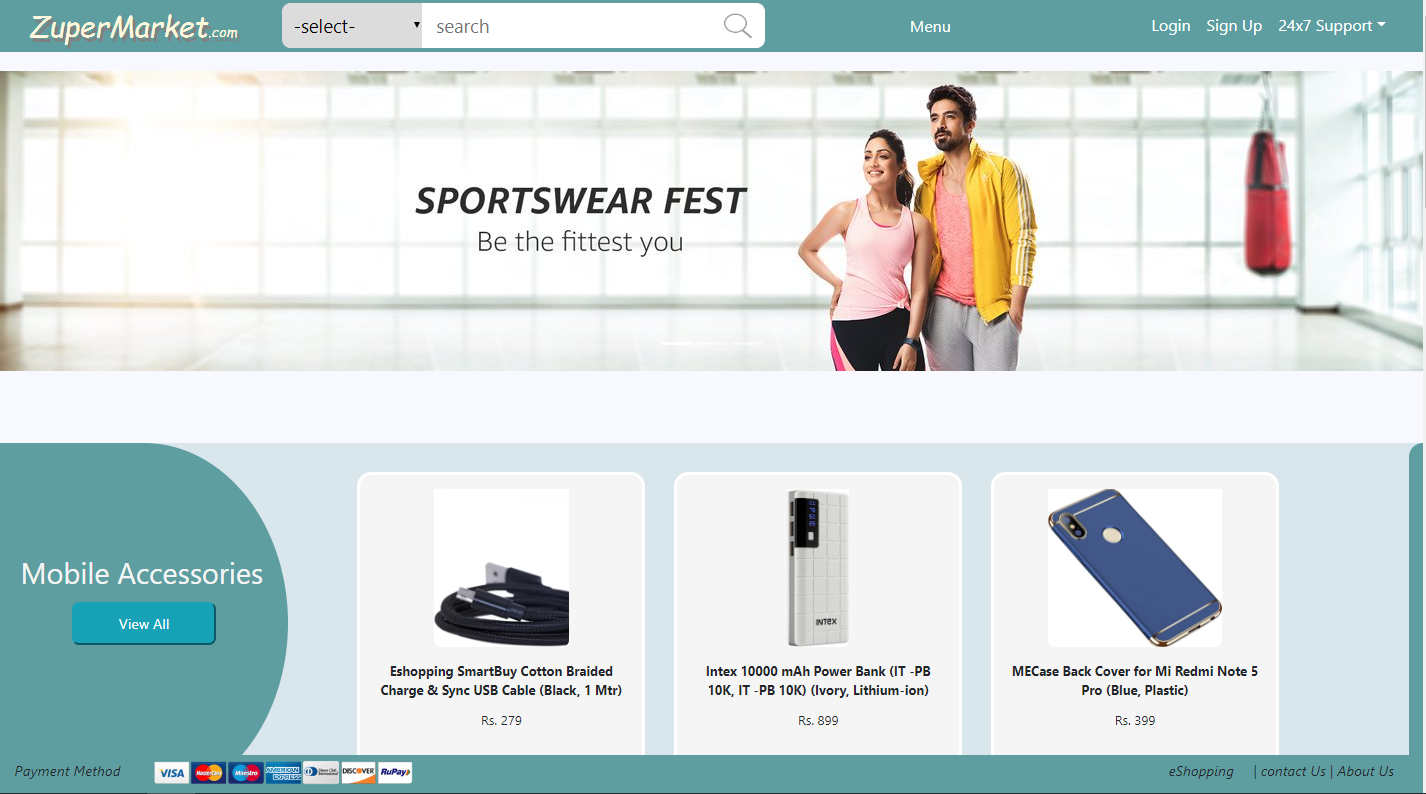
******

Figure 1Home Page

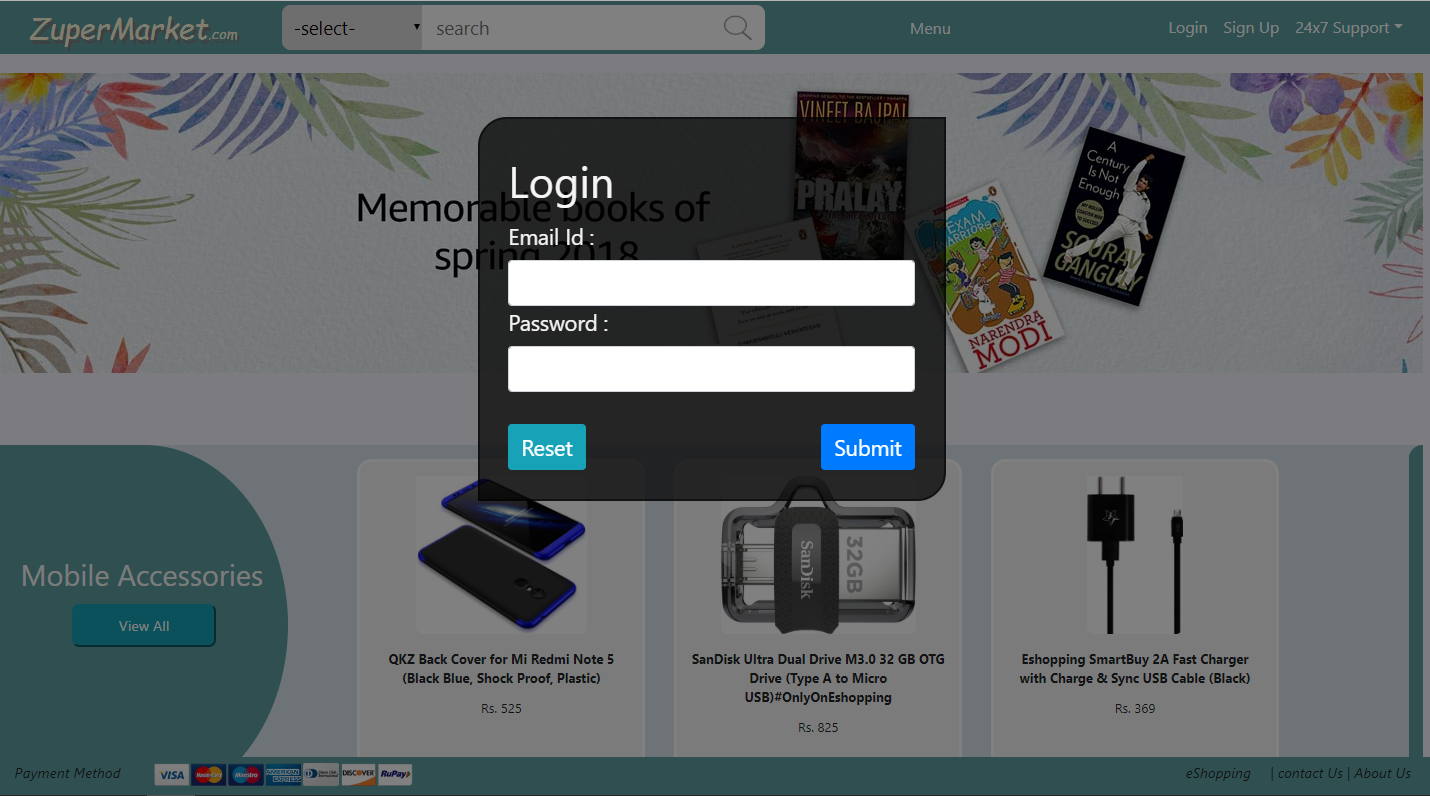
******

Figure 2 Login

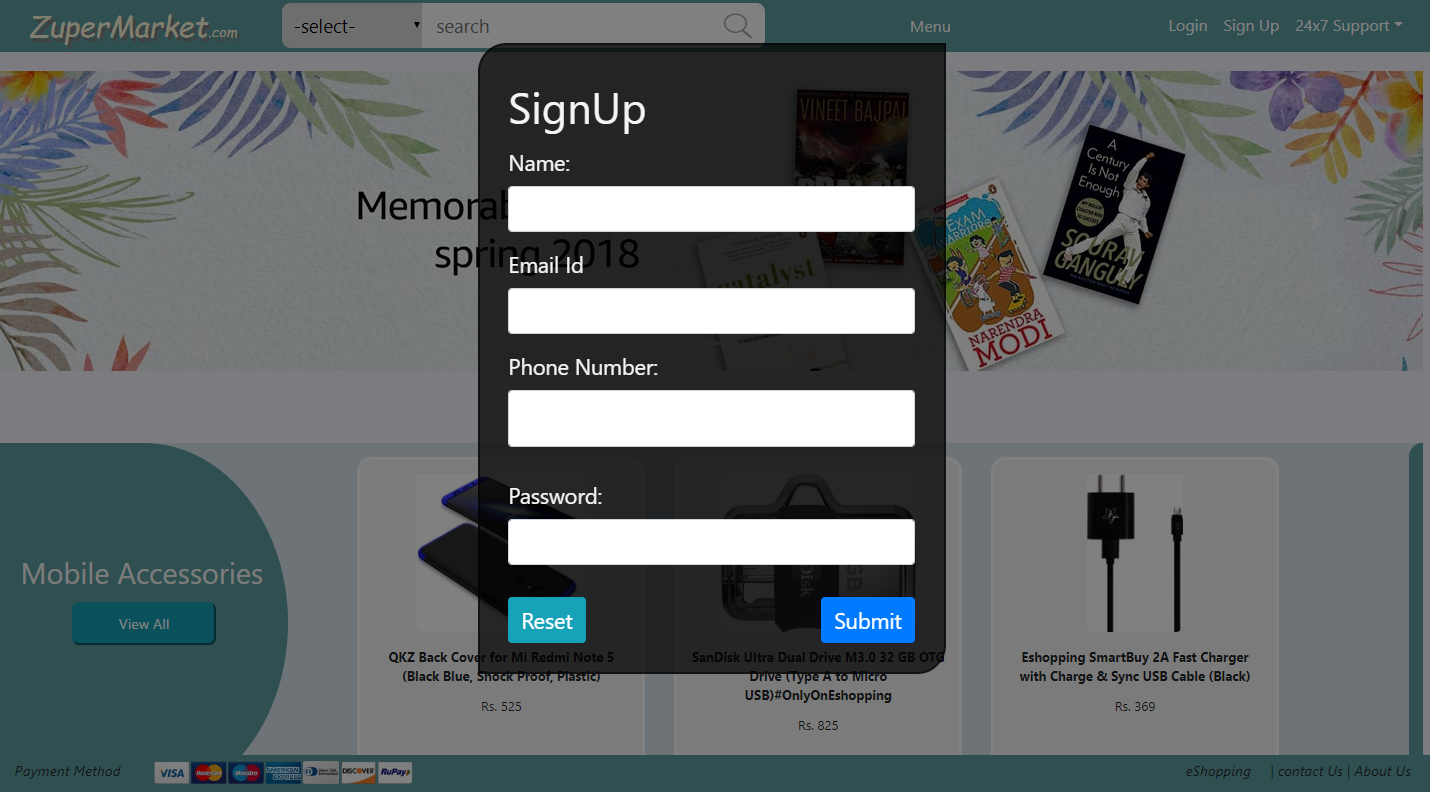
******

Figure 3 SignUp

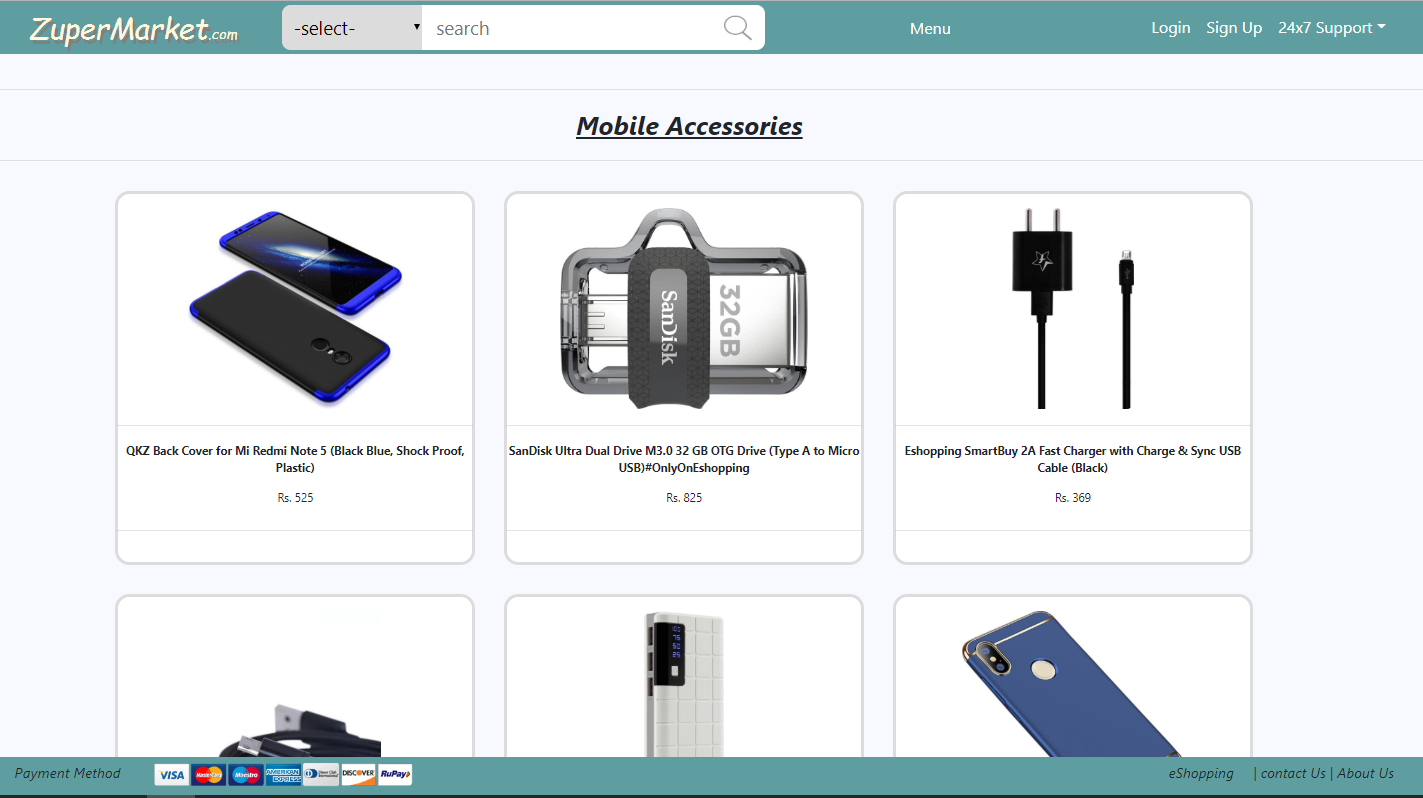
******

Figure 4 Categories By Product Listing

******

Figure 5 Categories By Product Listing Of Mobiles

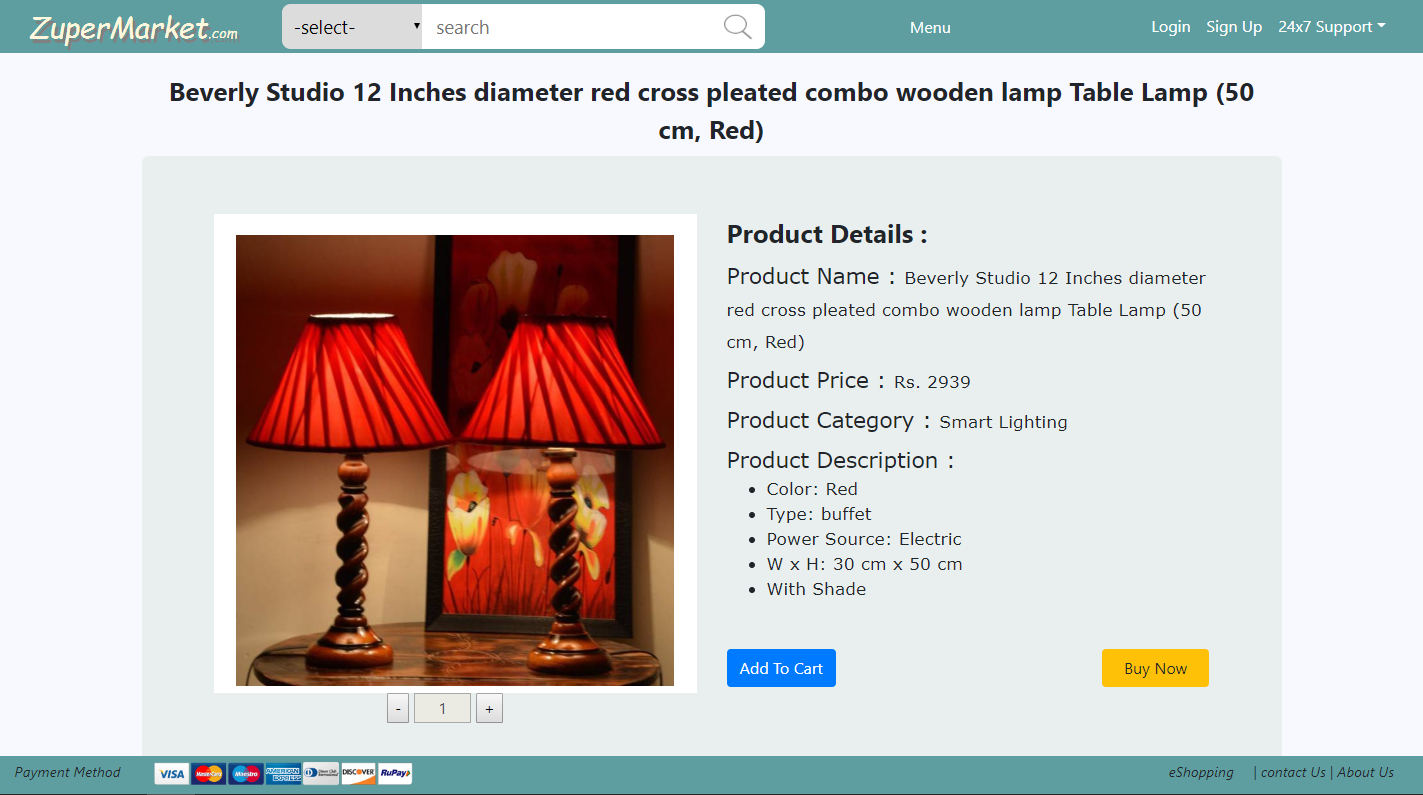
******

Figure 6 Description Of product

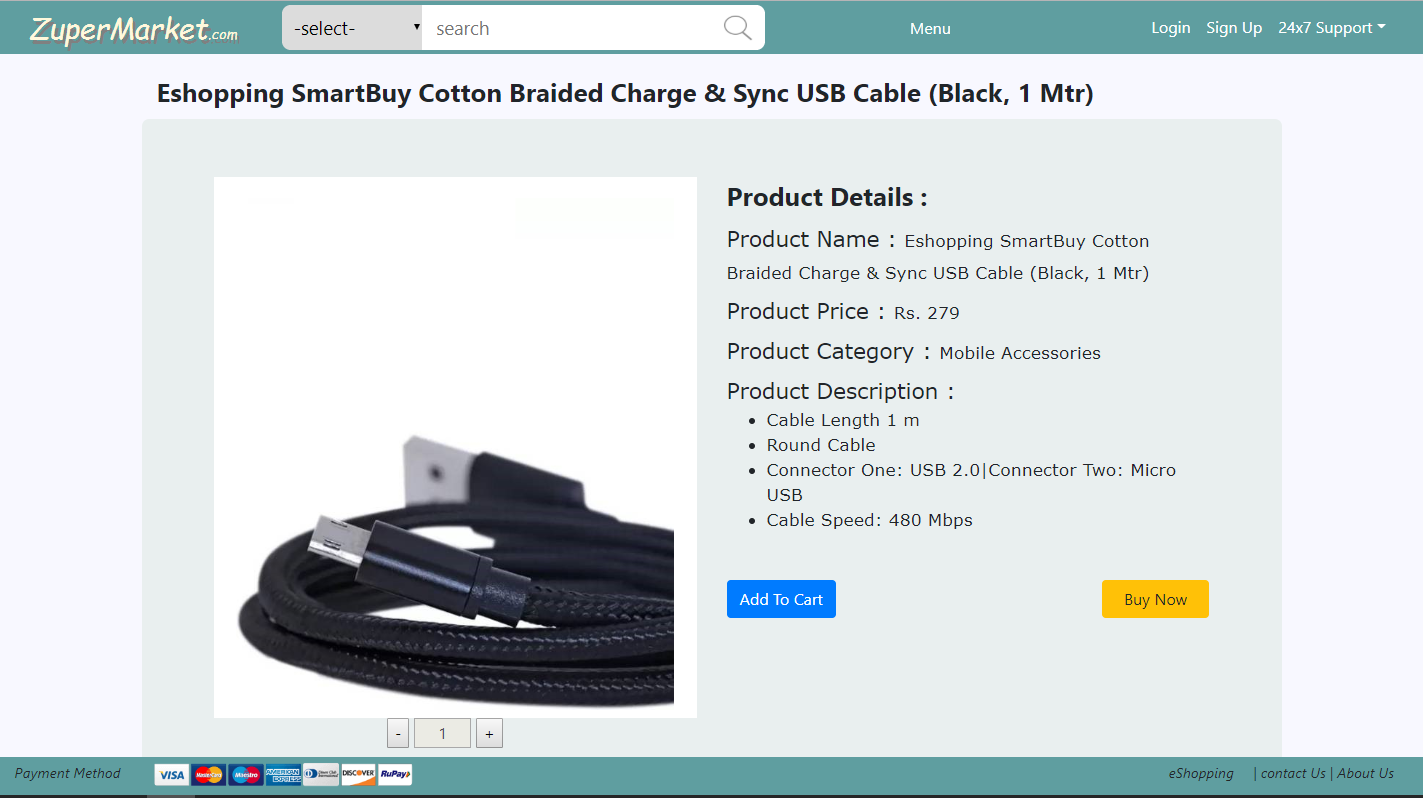
******

Figure 7 Description Of Product

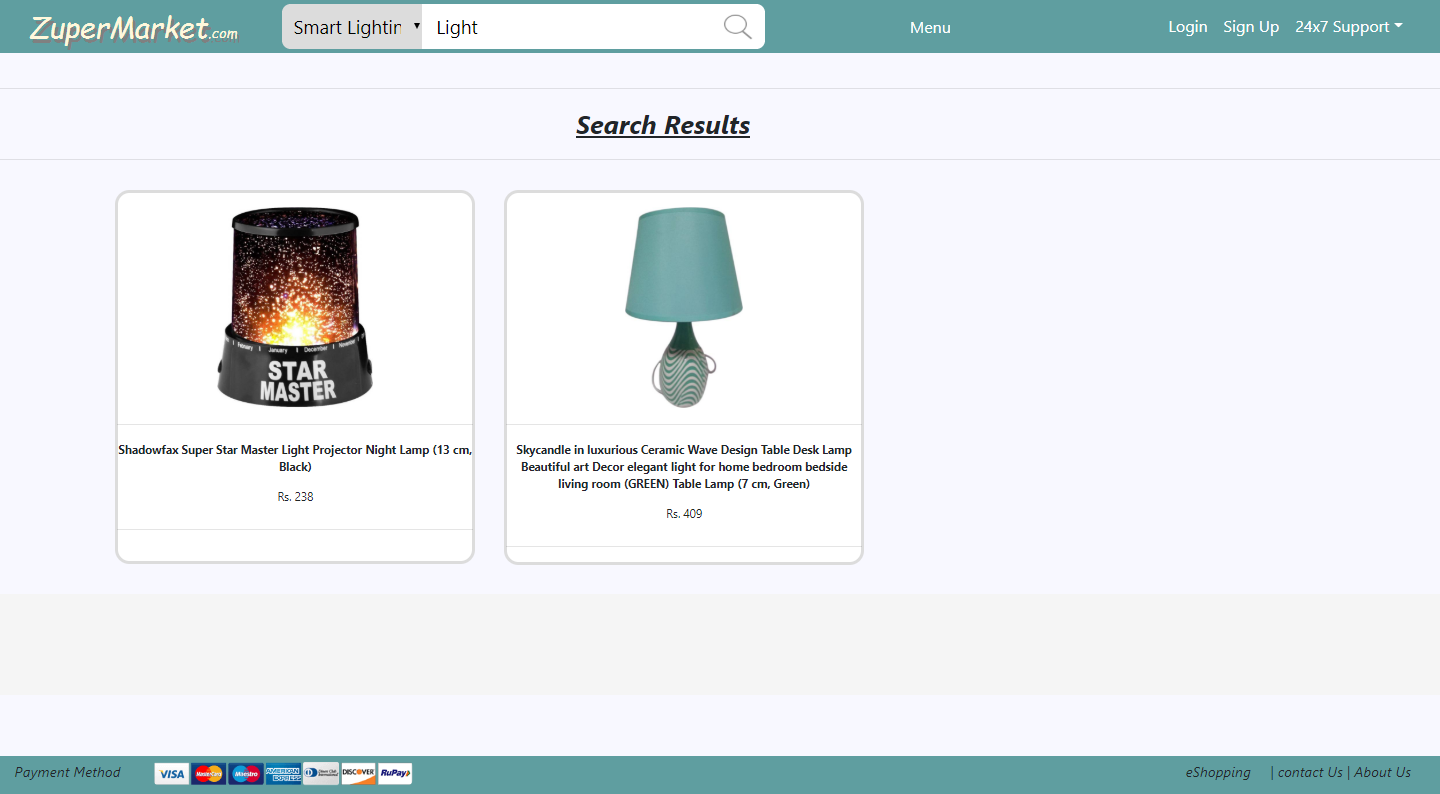
******

Figure 8 Search Result Of Light

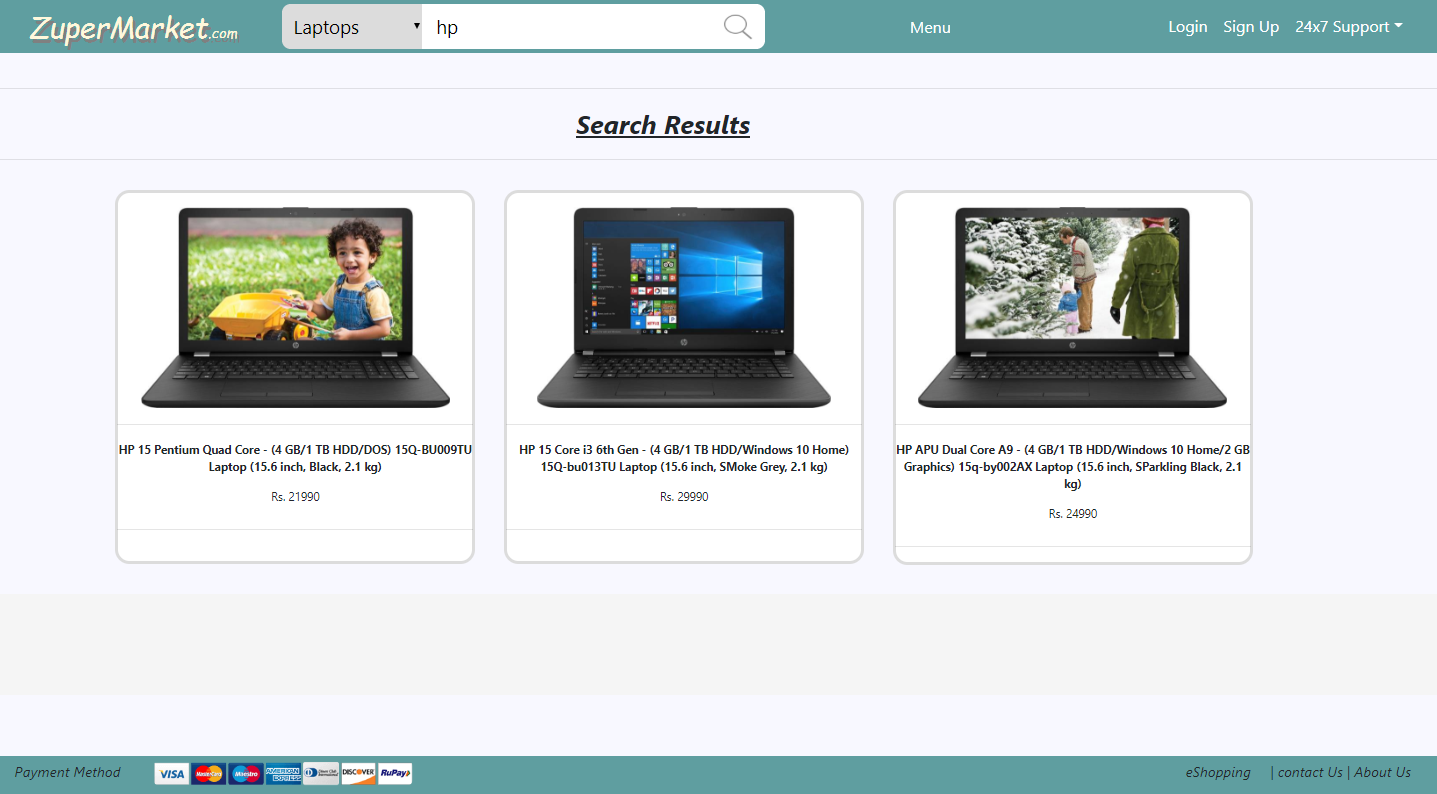
******

Figure 9 Search Results Of Hp

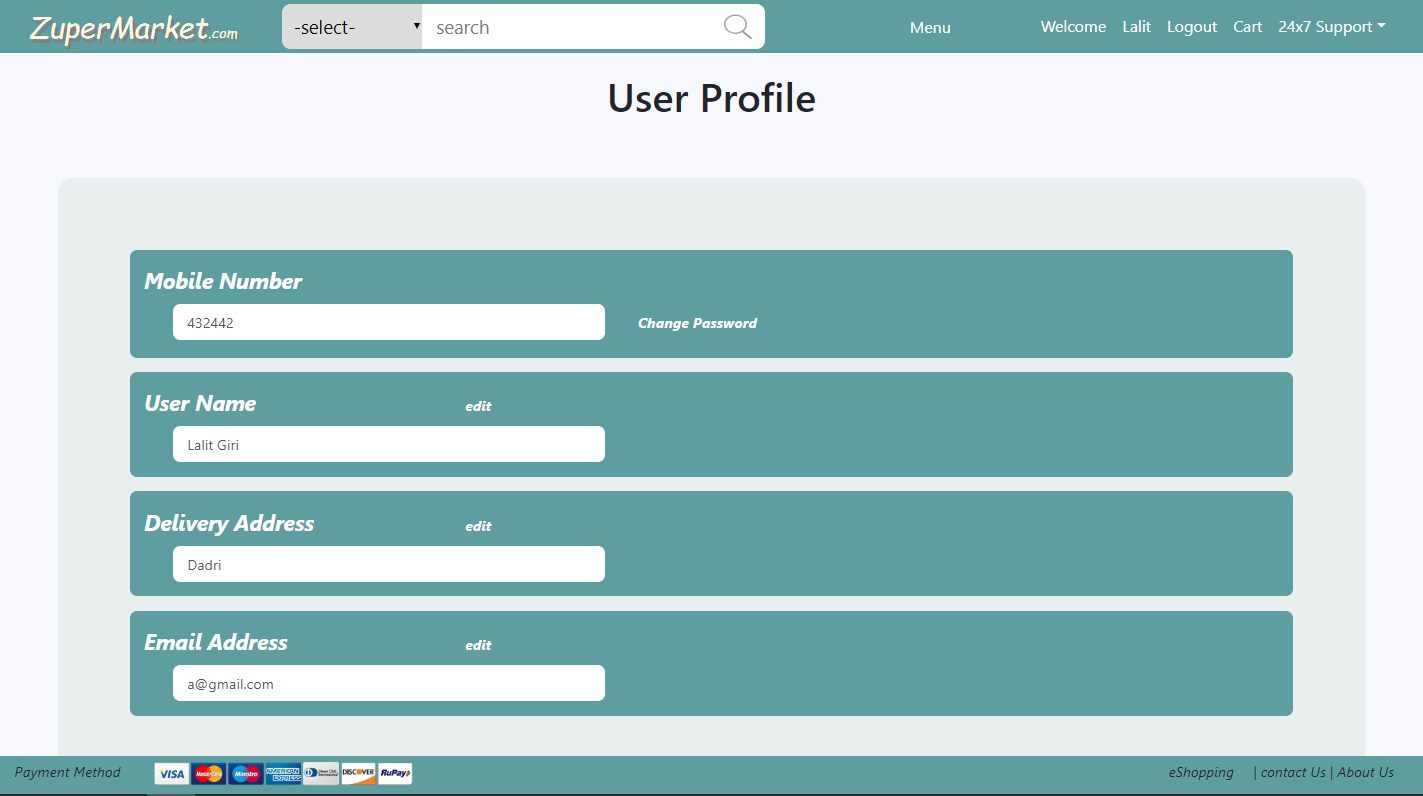
******

Figure 10 User Profile

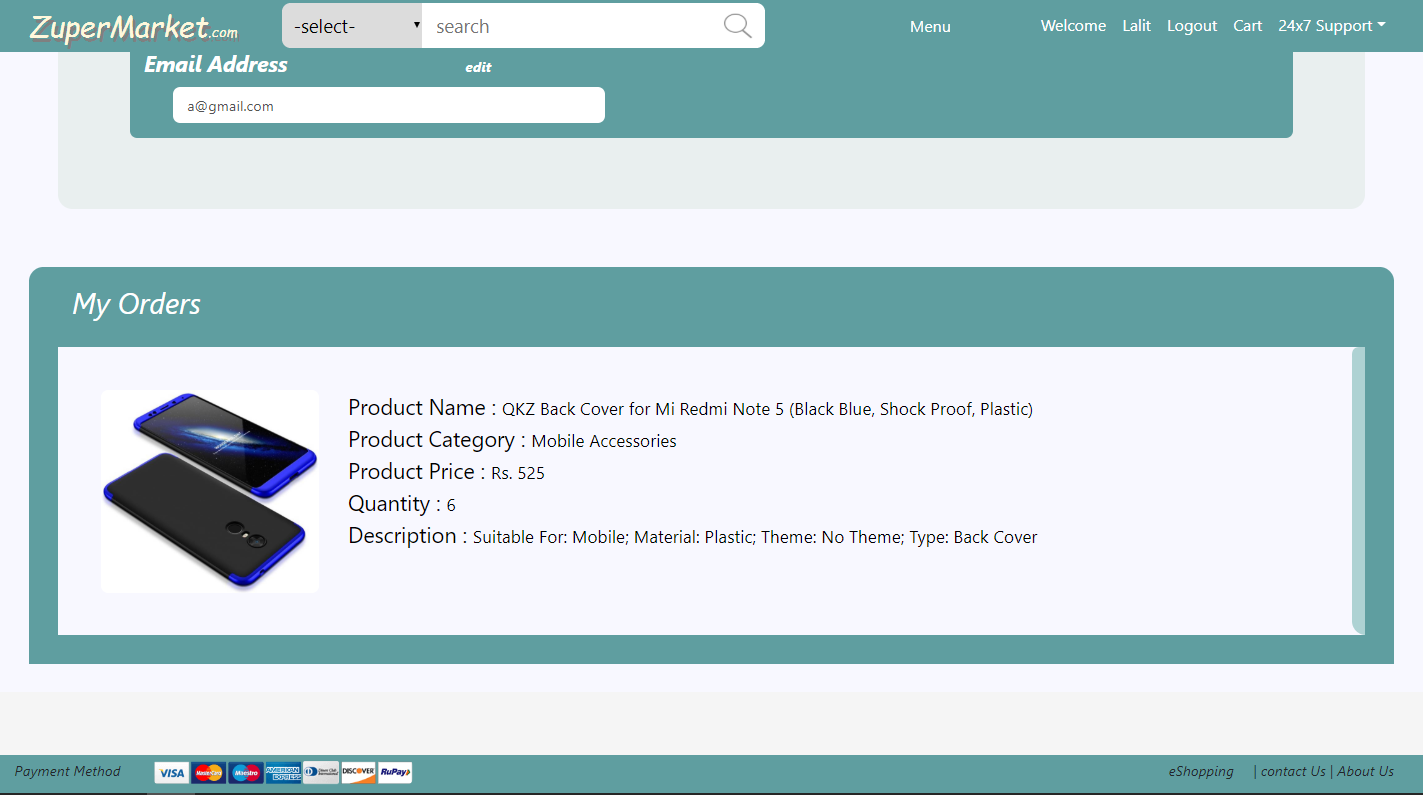
******

Figure 11 User Orders

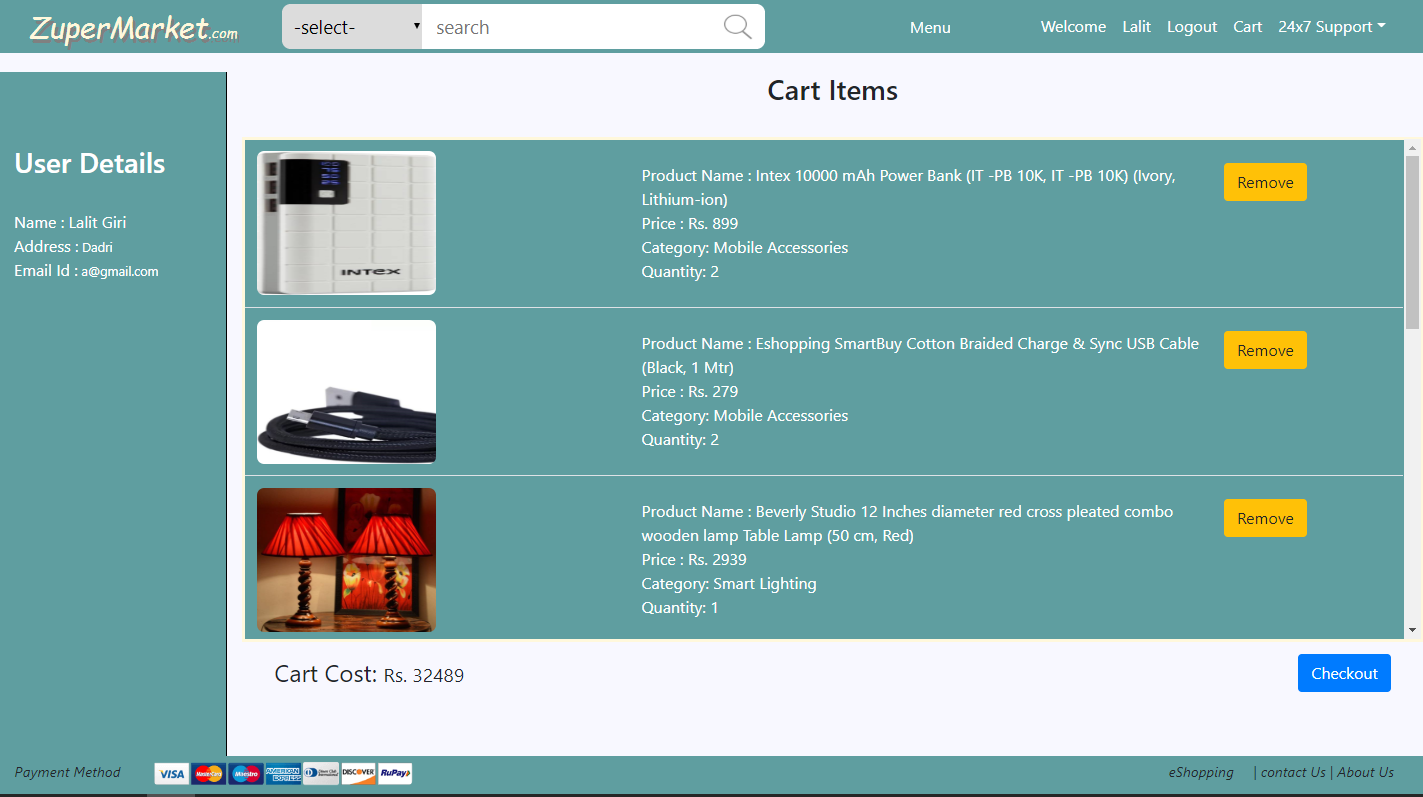
******

Figure 12 User Cart Items

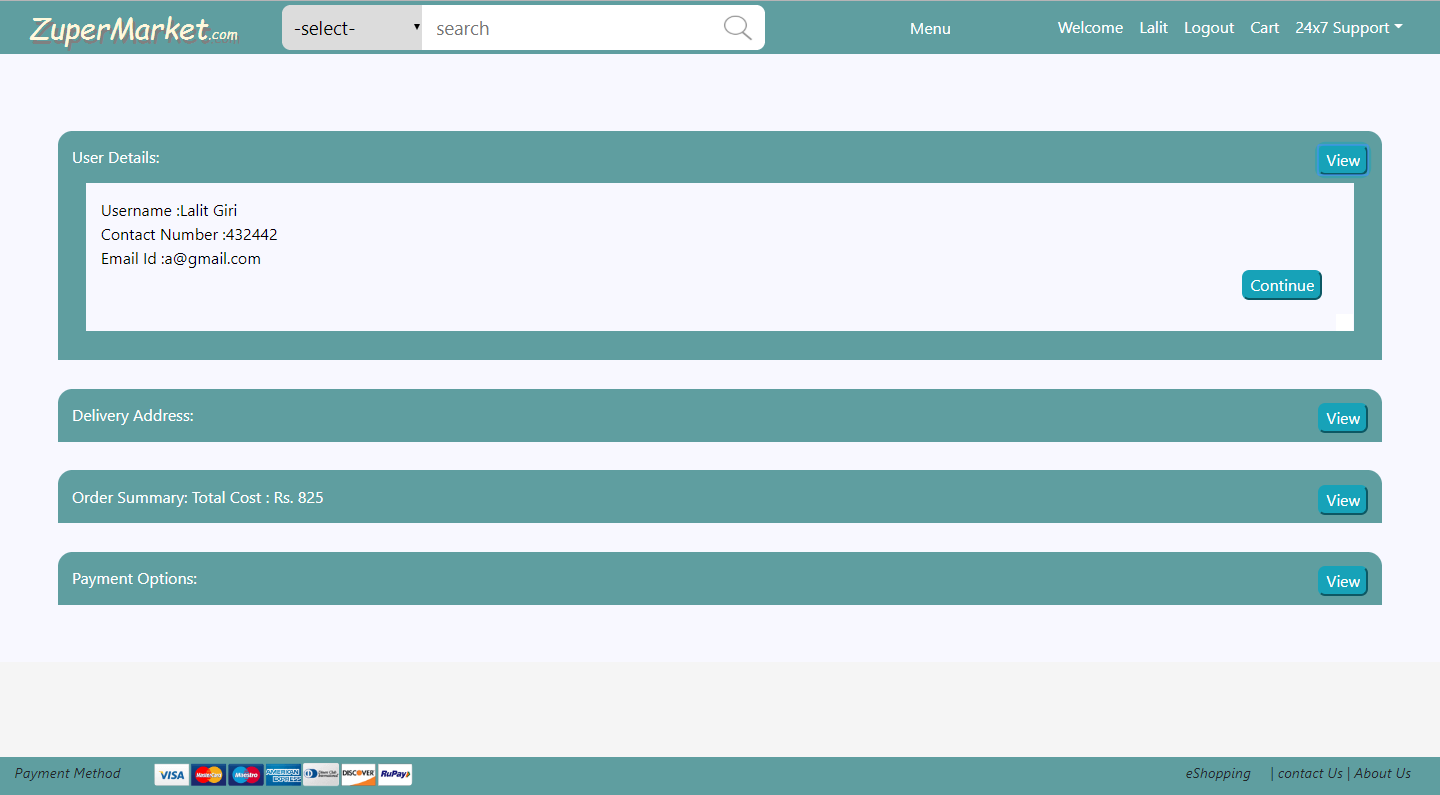
******

Figure 13 Order Booking

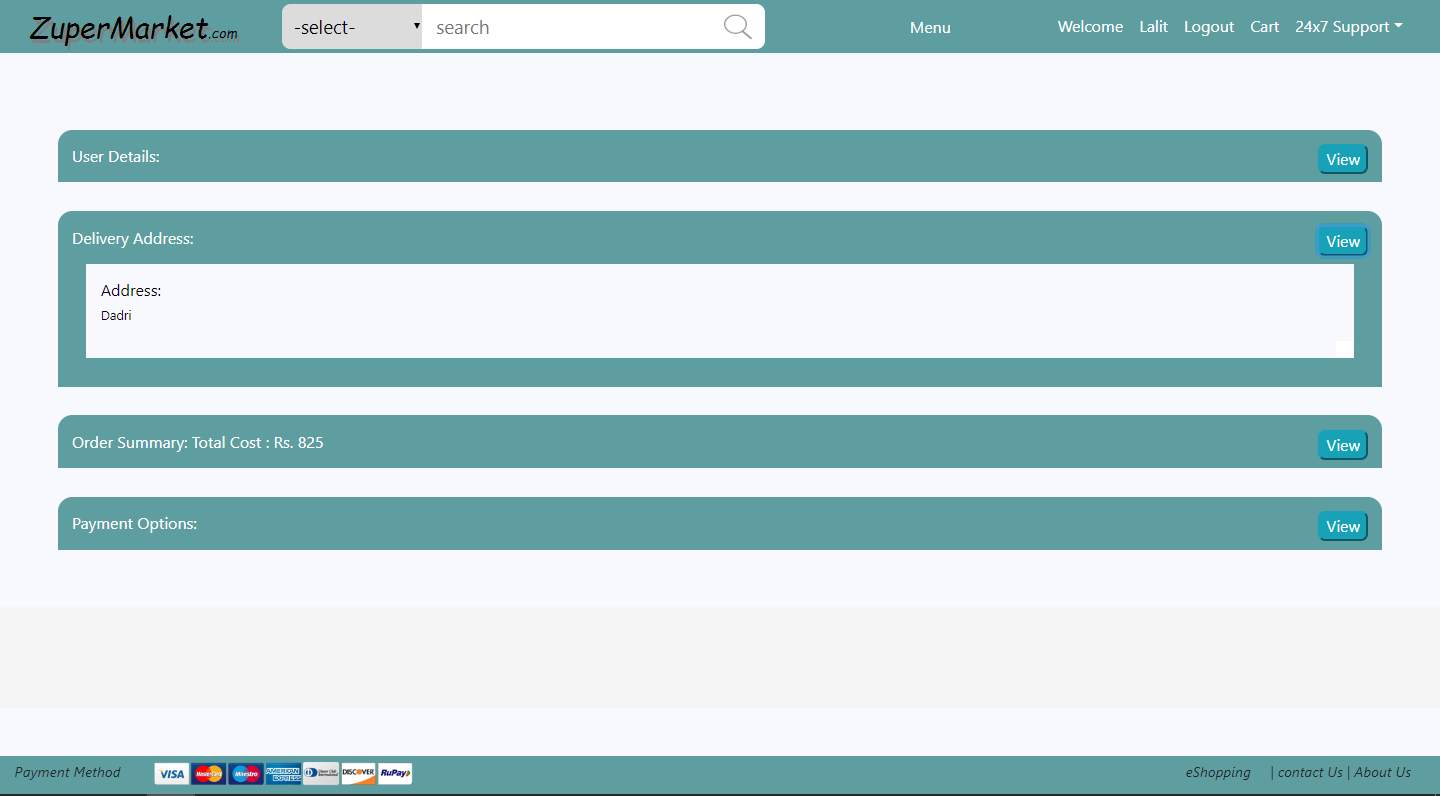
******

Figure 14 Order Booking

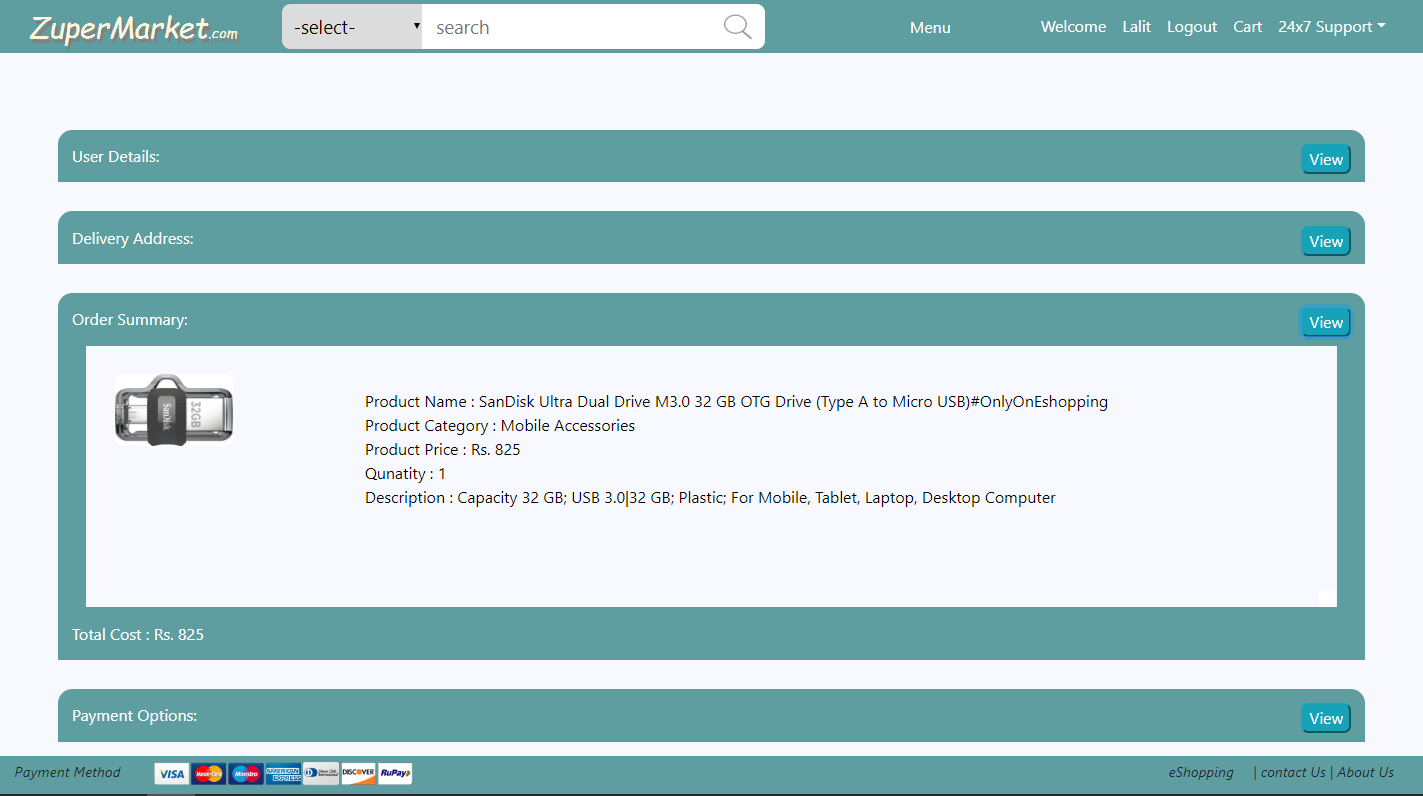
******

Figure 15 Order Booking Product Description

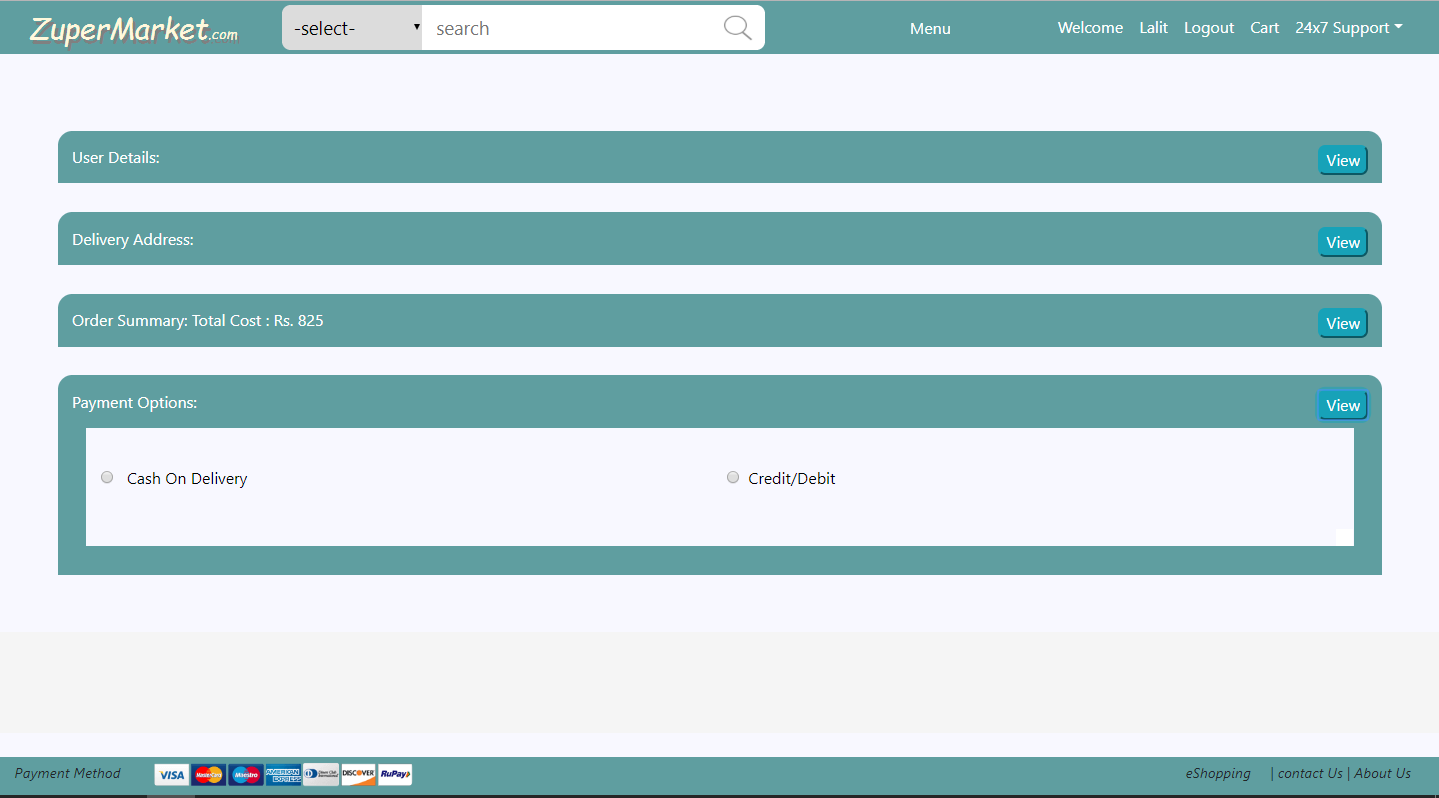
******

Figure 16 Order Booking Payment Option

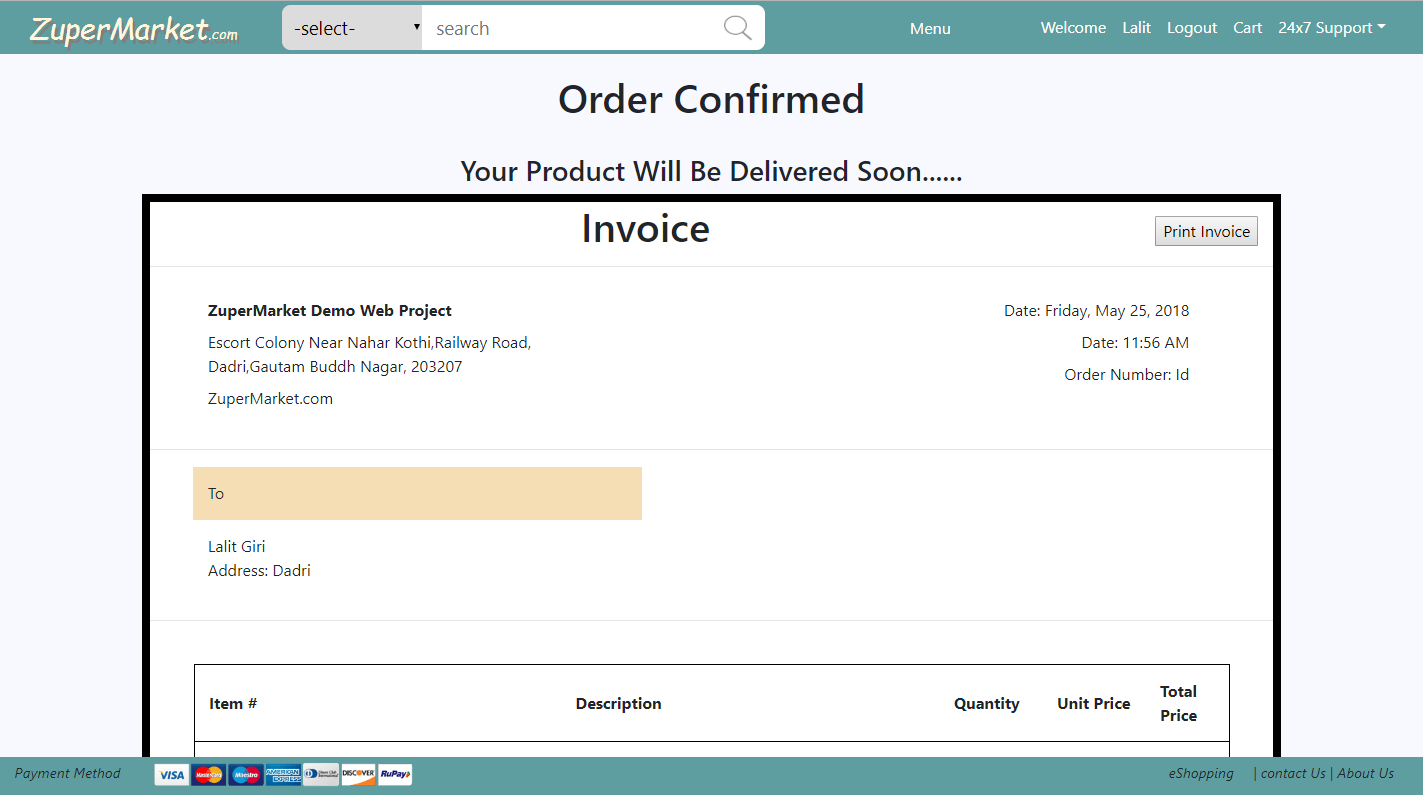
******

Figure 17 Order Confirmed With Invoice

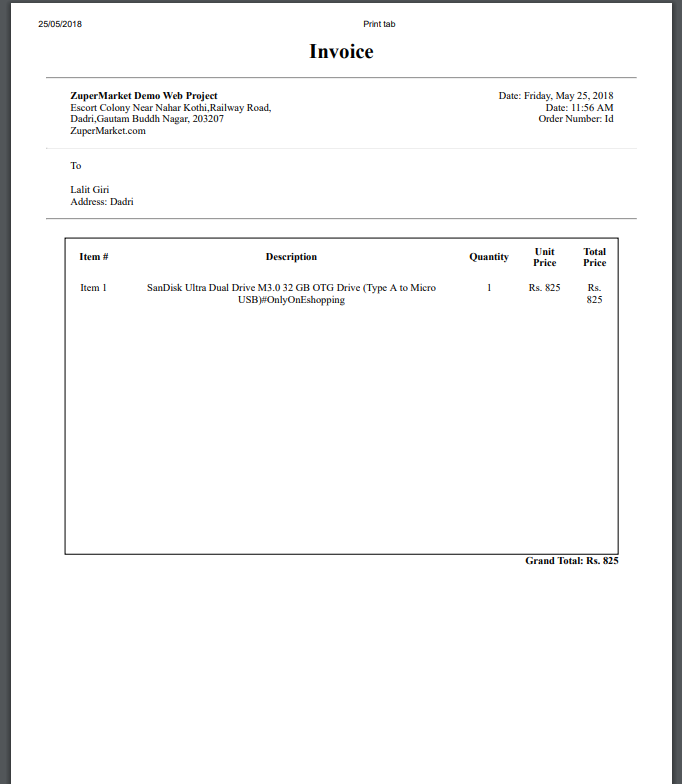
******

Figure 18 Invoice Copy

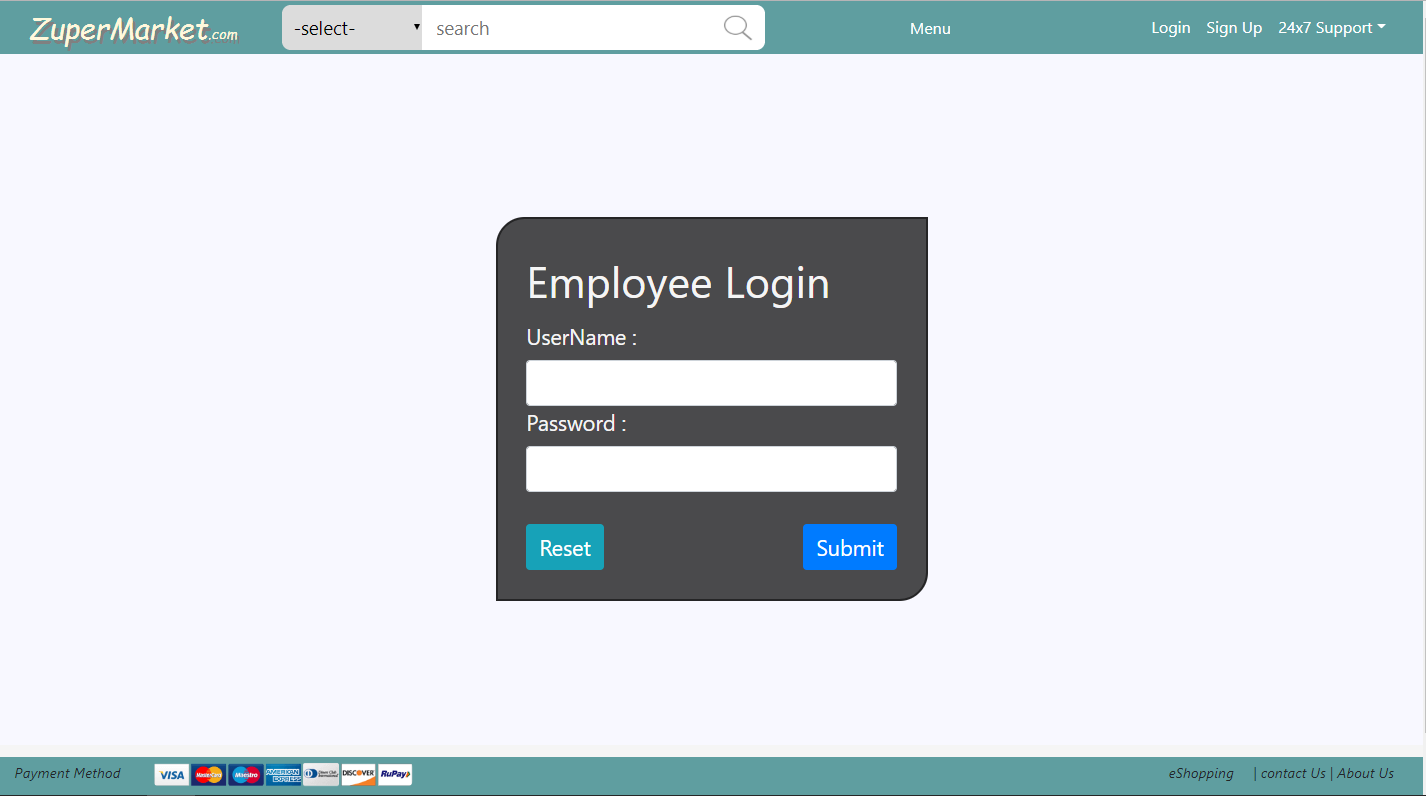
******

Figure 19 Employee Login

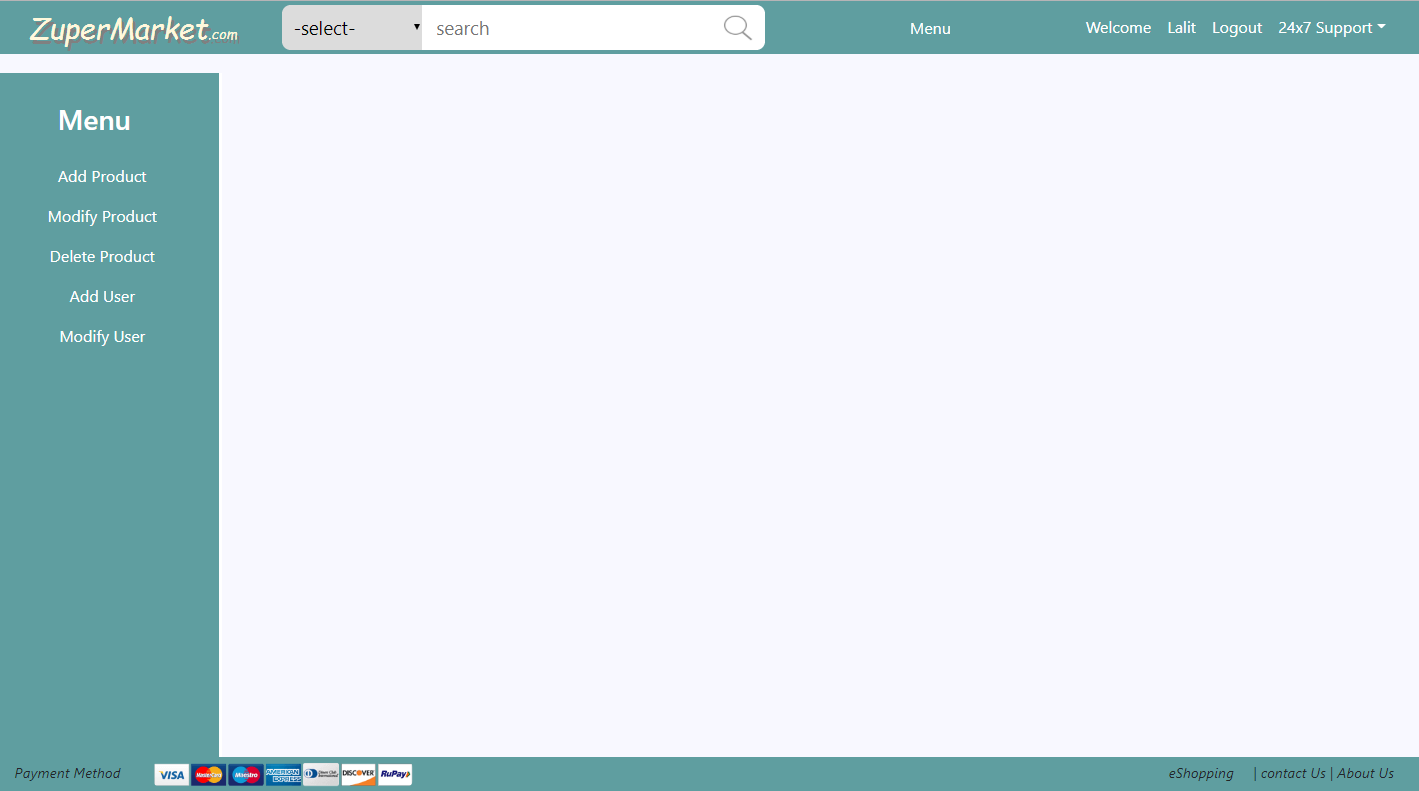
******

Figure 20 Employee Home Page

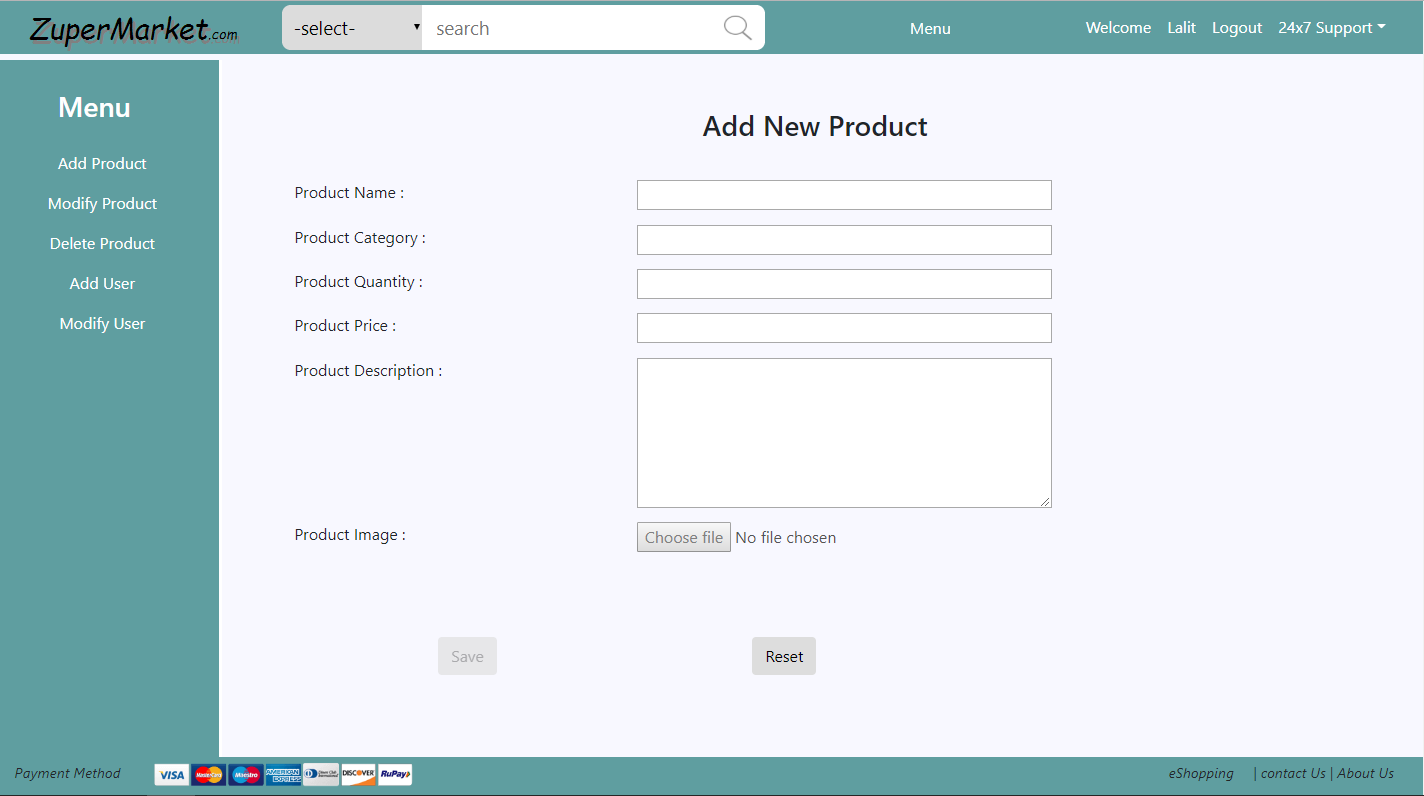
******

Figure 21 Add New Product

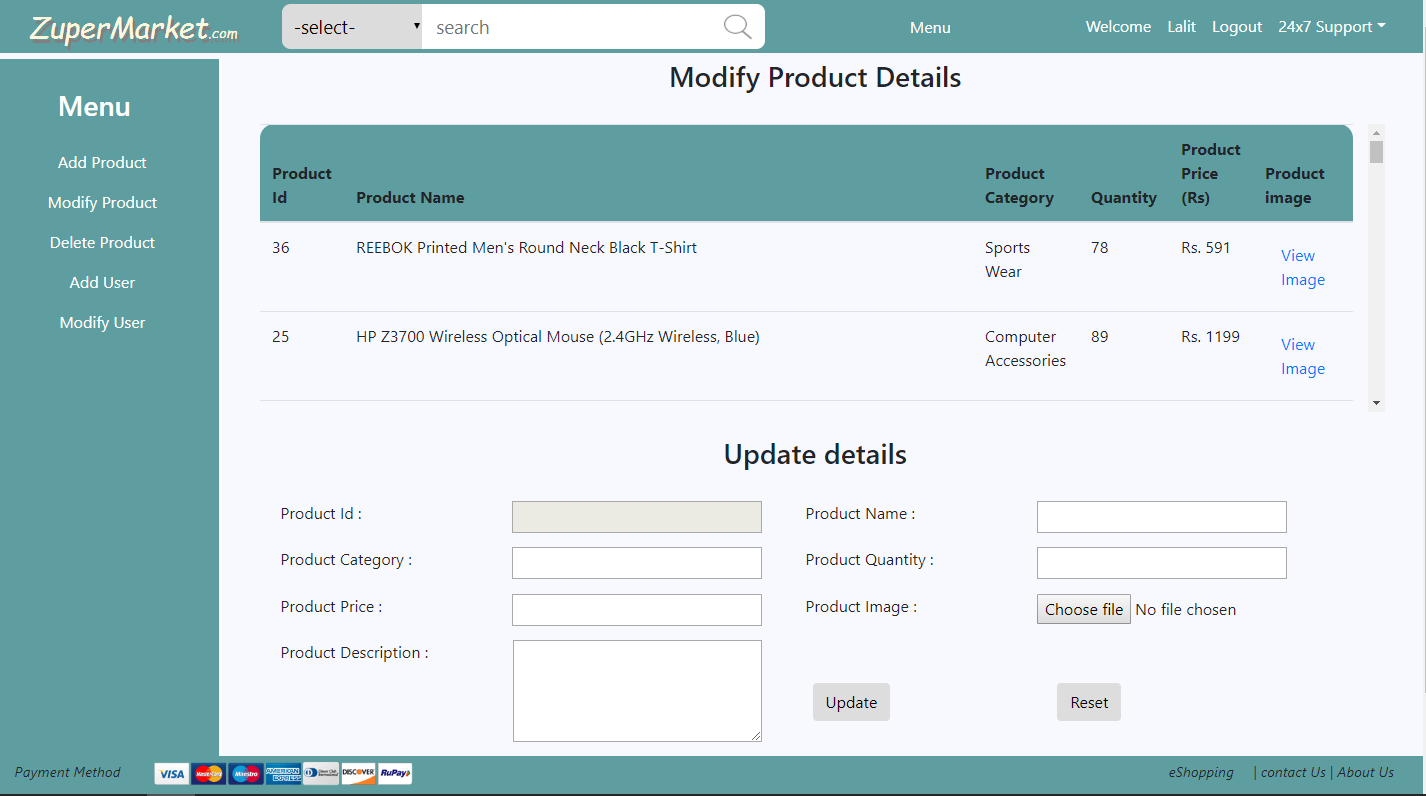
******

Figure 22 Modify Products

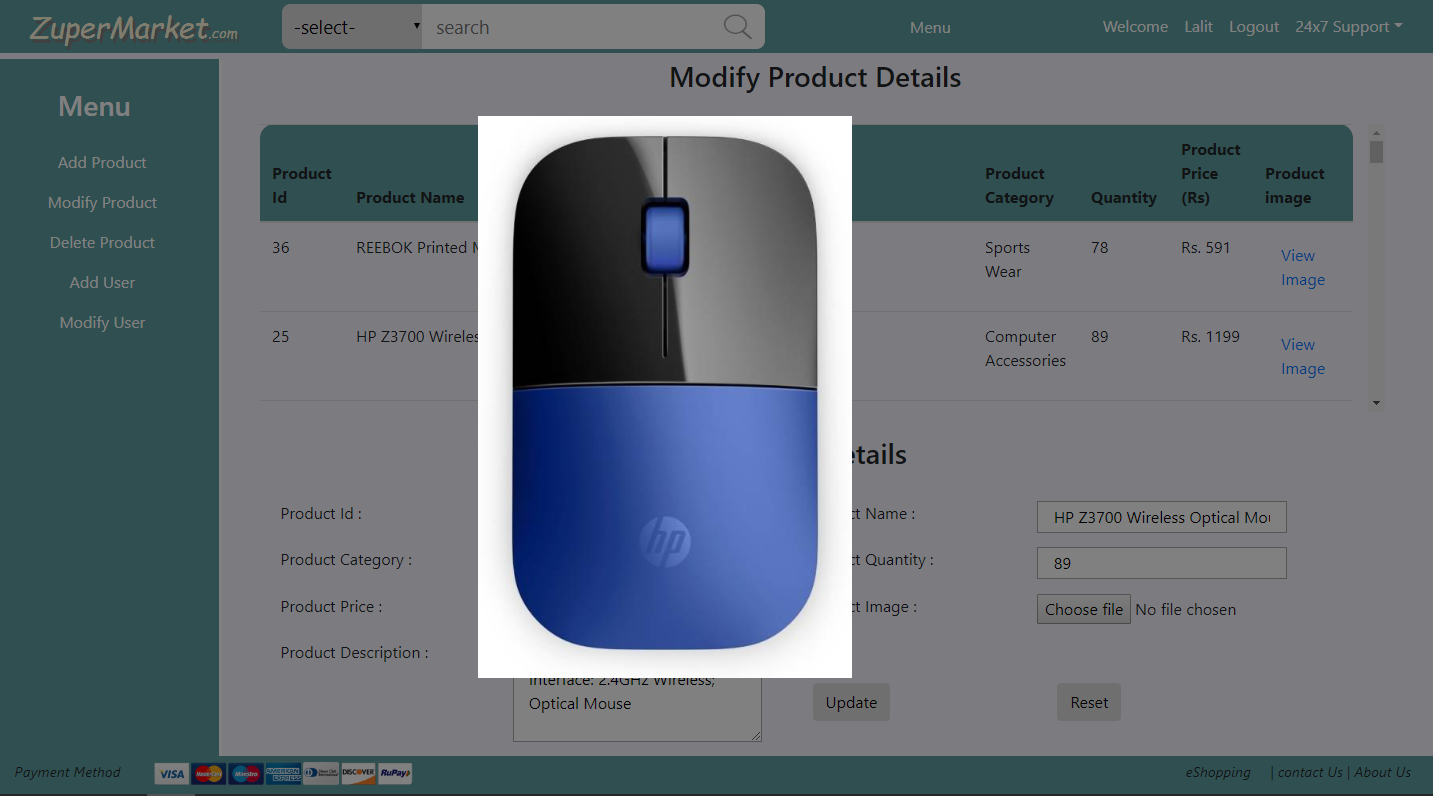
******

Figure 23 Image View When Modifying Products

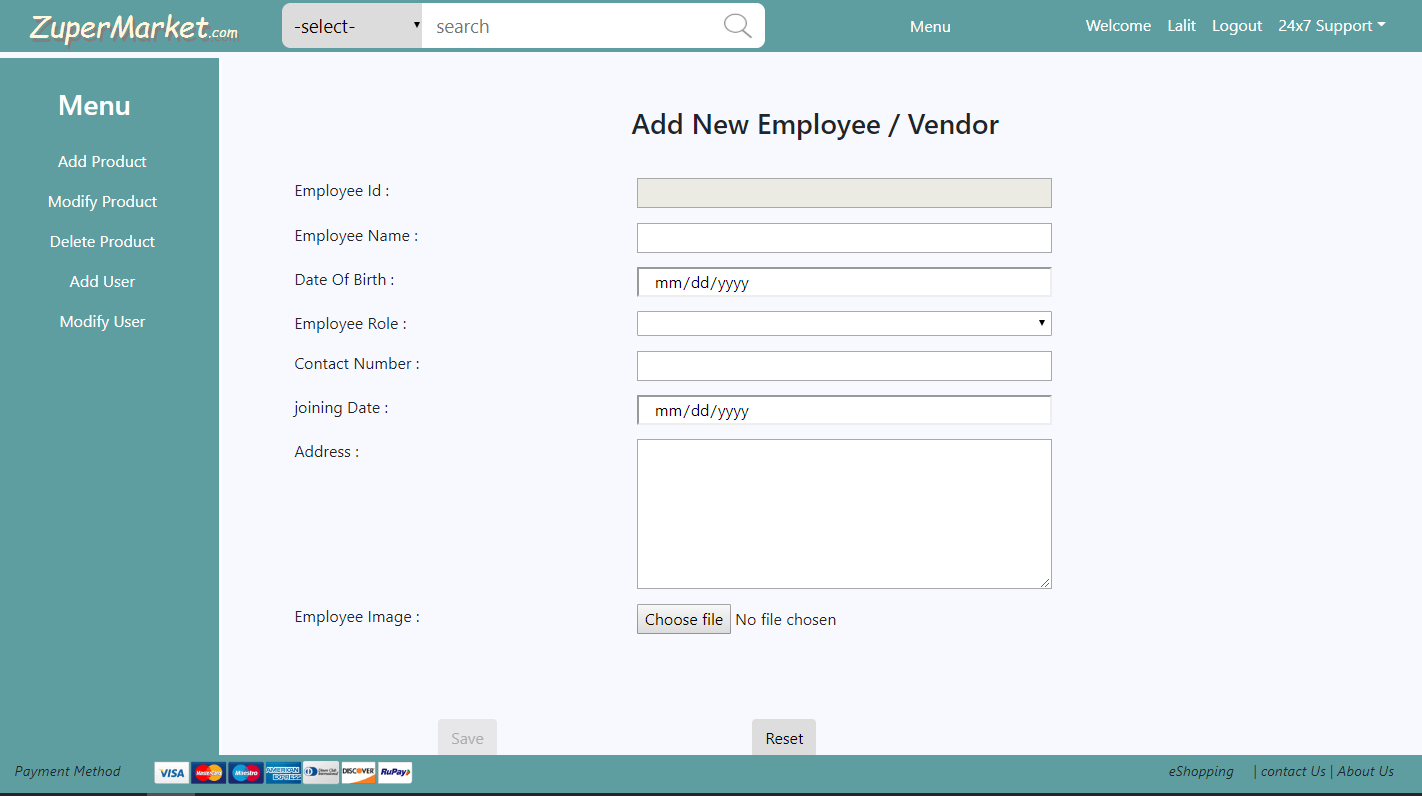
******

Figure 24 Add New Employee

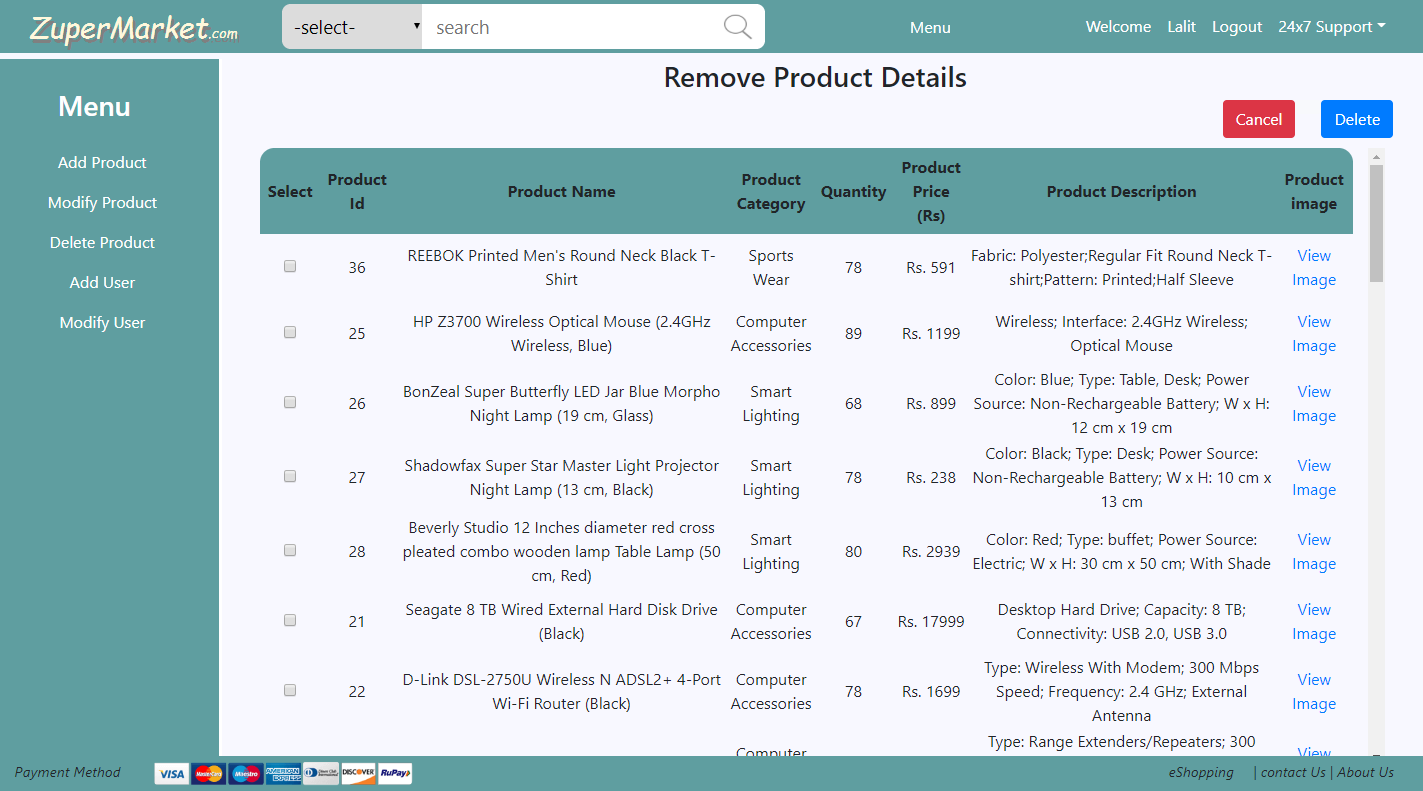
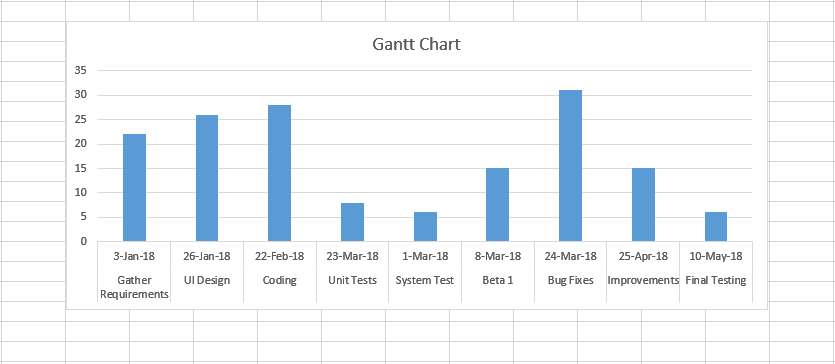
******

Figure 25 Remove Or Delete Products

***Gantt Chart***

******

******

***FUTURE SCOPE OF THE PROJECT***

“On-Line Shopping” is a web-based project which is made for remote-shopping or shopping through Internet. As the technology is being advanced the way of life is changing accordance. Now a day’s we can place the order for anything from our home. There is no need to go the shop of the things we want. The order can be placed online through Internet. The payment, the confirmation of purchasing; we can do everything we want. Now we can think that how the days have been changed with time. People had to stand in rows to wait there terms to buy a particular thing from a popular shop. But what is happening now a day’s; we can extremely surprise that those things can be available on the door-step in few hours. In future we will try to make this website which work so flexible and beneficial for the customer and also try to make smooth service.

***BIBLIOGRAPHY***

**WEBSITES REFFERED:**

* [www.journaldev.com/Blog/Java](http://www.journaldev.com/Blog/Java)
* [www.baeldung.com](http://www.baeldung.com)
* [www.javatpoint.com/java-tutorial](http://www.javatpoint.com/java-tutorial)
* [www.tutorialspoint.com](http://www.tutorialspoint.com)
* [www.angular.io](http://www.angular.io)
* www.javabrains.io