**KARNATAK UNIVERSITY, DHARWAD**

****

## Janata Shikshana Samiti

Shri Manjunatheshwara Institute of

UG & PG Studies, Vidyagiri, Dharwad-580004.



**A PROJECT REPORT ON**

“**BOMBAY NOVELTY E-STORE**”

**BACHELOR OF SCIENCE (Computer Science)**

**OF**

**KARNATAK UNIVERSITY, DHARWAD**

**PROJECT GUIDED BY:**

**Mrs. Renuka I**

**Submitted by**

**Aishwarya A Rokhade Priyanka V Rokhade**

**16M10103 16M10152**

**BSc (CS) VI Semester BSc (CS) VI Semester**

**DEPARTMENT OF COMPUTER SCIENCE 2018-2019**

**Janata Shikshana Samiti**

**Shri Manjunatheshwara Institute of UG & PG Studies,**

**Vidyagiri, Dharwad. 580004**



**CERTIFICATE**

**This is to certify that Ms.Aishwarya A Rokhade and Ms. Priyanka V Rokhade has satisfactorily completed Project work entitled “Bombay Novelty E-store” for the partial fulfillment of BSC(CS) prescribed by Karnataka University, Dharwad during the academic year 2018-2019**

**Mrs.Renuka I Prof.Vivek.M.Laxmeshwar Dr.Ajith Prasad**

**Project Guide HOD Principal**

**Examination Register No: Examiner’s**

**1. 16M10103 1.**

**2. 16M10152 2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ACKNOWLEDGEMENT**

The successful presentation of this project is an acknowledgement of the immense support extended by **JSS SHRI MANJUNATHESHWARA INSTITUTE OF UG & PG STUDIES VIDYAGIRI, DHARWAD.** Which has provided an opportunity to fulfill our goal.

We would like to express our heartfelt thanks to our **President SHRI SHRI VISHWESHATEERTHA SRIPADANGALAVARU SWAMIJI OF PEJAVARMATH, UDUPI & PADMABHUSHANA PUJYA Dr. D. VEERENDRA HEGGADE, Chairman of JANATA SHIKSHANA SAMITI & Dharmadhikari of DHARMASTHALA.** For their kind blessing and our beloved **Secretary** **of JANATA SHIKSHANA SAMITI’SDr. N.VAJRAKUMAR.**

We express our thanks to our beloved **Principal Dr. AJITH PRASAD** for extending allthe facilities to carry out this project. We express our thanks to our beloved **UG HOD Dept. Of Computer Science Prof. VIVEK M. LAXMESHWAR** who has been a constant source of inspiration and motivation. We would also like to express out special thanks to our **Project Guide** **Smt. RENUKA I** who has spent her precious time by guiding and encouraging us throughout the development of the project.

And we are grateful to our **Parents**, our **Friends** and all the people who have helped us directly or indirectly to make this project a success.

**Ms. AISHWARYA A ROKHADE (16M10103)**

**Ms. PRIYANKA V ROKHADE (16M10152)**

**CONTENTS**

**Topic Page No**

**1. PROJECT SYNOPSIS - - - - - - - - - - - - - - - - - 06  
 1.1 Introduction.  
 1.2 Existing System   
 1.3 Proposed system   
 1.4 Input of the project/System.  
 1.5 Output of the project/System.  
1.6 Process Logic.**

**1.7 Tools/Platforms, Languages to be used.**

**1.8 Limitations of the project.**

**1.9 Scope or future application.**  
**2. FRAMEWORK - - - - - - - - - - - - - - - - - - - - - - - - 10  
2.1 Python and Framework  
2.1.2 Django  
 2.1.2 Usage  
 2.1.3 Speed Optimization  
2.2 HTML**

**2.2.2 Features of HTML**

**2.3 CSS**

**2.4 JAVA SCRIPT  
2.5 MY SQL  
2.5.1 Introduction To MySQL  
2.5.2 Features Of MySQL  
2.6 Create A Connection To a Database  
2.7 MySQL Benefits  
  
3. Project subject - - - - - - - - - - - - - - - - - - - - - - - - - - - 19  
 3.1 Modules**

**4. SystemRequirementsAndAnalysis - - - - 20**  
**4.1 System Requirements  
 4.2 System Analysis  
 4.3 System Design  
 4.4 Data Tables**

**5. Data Flow Diagrams - - - - - - - - - - - - - - - - - - 28  
5.1 Data Flow Diagram**

**5.2 DFD Notations**

**5.3 Steps Required In DFD**

**5.4 Context Level Diagram**

**5.5 Level 1 DFD**

**6. ER Diagrams - - - - - - - - - - - - - - - - - - - - - - - -31**

**7. Tables used in the database- - - - - - - - - - - - -33**

**8. Sample outputs /Snapshots - - - - - - - - - - - - - 39**

**9. Source Code - - - - - - - - - - - - - - - - - - - - - - - - - 43**

**10. Testing - - - - - - - - - - - - - - - - - - - - - - - - - - - - 75  
10.1 Scope Of Testing.  
 10.2 Testing Objective  
10.3 About Testing  
10.4 Software Quality Assurances.   
 10.5 Testing Strategies  
 10.6 Test Cases  
 10.7 Limitations**

**11. Bibliography - - - - - - - - - - - - - - - - - - - - - - - -77**

**1.Project Synopsis**

**1.1 Introduction:**

“Bombay Novelty Stores” shop was established by Kalappa V Rokhade in 1954 which is situated at Subhash road, opposite old bus stand Dharwad. Earlier he had two shops in which they use to sell varieties of stationary items in one shop and another shop had stainless steel utensils, wedding cards. Further they brought a scheme from Bombay by introducing “Mutual Benefit Scheme” for customers that set an era of rise to the public demand. At the initial he uses to sell flasks of Eagle Company, Aluminum products, VIP bags and Iron box. Further he combined his two shops and made a showcase on newly introduced items such as Prestige Cooker, Water filter, Mixer Grinders, Induction stove, Induction cooker etc which gained the growth of business, trust of the customer and maintained its dignity. Today the same dignity and business achievements is been continued by his three of his sons by selling varieties of kitchen and home appliances at reasonable rates and best discounts.

**1.2 Existing System and Its Disadvantages**

* The currently running system is manual.
* Calculations and maintenance of records takes lots of time.
* Customer and shopkeeper finds difficult to handle during rush hour.
* All transactions and cash flows are recorded manually on paper or register.
* Stock maintenance is done manually.

**1.3 Proposed System And Its Advantages:**

* The application provides a computerized data storage facility.
* This system is used to maintain the data about stock, purchase, transactions etc.
* This system provides better transactions use such as debit & credit cards.
* We can track our products easily in mobile.
* New system provides security against unauthorized access.
* Also provides better services to the customers.

**1.4 Input of the Project/System:**

* Administrator adds product information.
* Updating the stock.
* Providing customer details.
* Options for payment method like debit and credit.
* Provides supplier information to update our stock.
* Product services for the customers.

**1.5 Output of the Project/System:**

* Provides customer account details.
* It provides product details to the customers.
* Generates bills for customer.
* Delivery and tracking details to the customer.
* Provides invoice notifications through email.
* Exchange offers to the customers.

**1.6 Process Logic**

**INPUT OUTPUT**

* Product details
* Account details
* Billing details
* Order details
* Delivery details
* Product information
* Customer information
* Stock information
* Order information
* Supplier information

**1.7 TOOLS/PLATFORMS, LANGUAGES TO BE USED**

**SOFTWARE REQUIREMENTS**

* Platform :Web Application
* Operating system :Windows 7 and others
* Database :My SQL
* Front end :HTML, CSS, java script
* Language :Python
* Framework :Django
* Browser :Any of Chrome, Opera, Mozilla Firefox

**Hardware Requirements:**

* Processor : Intel Core i3
* Hard disk space : 40GBmin
* RAM : 2GBmin

**1.8 LIMITATIONS OF THE PROJECT**

* Although our software presents a broad range of options to its users some intricate options could not be covered into it.
* Single language facility can be given presently.
* Presently we are adding limited stocks.
* Invoice messages like SMS cannot be provided currently.

**1.9 SCOPE OR FUTURE APPLICATION**

* Multiple language facility can be given.
* If any new category is introduced can be added.
* It can also be part of android application.
* Barcode reading can be provided.
* Net banking and other facilities for transactions can be provided.

**2.Framework**

**2.1 Python and Framework**

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

Python provides lots of features that are listed below.

* Easy to Learn and Use
* Python is easy to learn and use. It is developer-friendly and high level programming language.
* Expressive Language
* Python language is more expressive means that it is more understandable and readable.
* Interpreted Language
* Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.
* Cross-platform Language
* Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.
* Free and Open Source
* Python language is freely available at official web address. The source-code is also available. Therefore it is open source.
* Object-Oriented Language
* Python supports object oriented language and concepts of classes and objects come into existence.
* Extensible
* It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.
* Large Standard Library
* Python has a large and broad library and provides rich set of module and functions for rapid application development.
* GUI Programming Support
* Graphical user interfaces can be developed using Python.
* Integrated
* It can be easily integrated with languages like C, C++, JAVA etc.

**Django**

A web framework is a collection of modular tools that abstracts away much of the difficulty, repetition and inherent in web development. For example, most websites need the same basic functionality: the ability to connect to a database, set URL routes, display content on a page, handle security properly, and so on.

Rather than recreate all of this from scratch, programmers over the years have created web frameworks in all the major programming languages: Django and Flask in Python, Rails in Ruby, and Express in JavaScript among many, many others.

Django is a free, open source web framework written in the Python programming language.Its popularity is due to its friendliness to both beginners and advanced programmers.

Django is robust enough to be used by the largest websites in the world–but also flexible enough to be a good choice for early-stage startups and prototyping personal projects.

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.

This framework uses a famous tag line: “The web framework for perfectionists with deadlines”.

By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

**Django helps you write software that is:**

Complete:--

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation.

Versatile:--

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

Secure:--

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.

A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and click jacking (see Website security for more details of such attacks).

Scalable:--

Django uses a component-based “shared-nothing” architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed).Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands

e.g.Instagram

Maintainable:--

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

Portable:--

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavors of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

**2.2 HTML**

HTML means Hypertext Markup Language. HTML is a method of describing the format of documents which allows them to be viewed on computer screens. HTML documents are displayed by web browsers, programs which can navigate across networks and display a wide variety of types of information. HTML pages can be developed to be simple text or to be complex multimedia extravaganzas containing sound, moving images, virtual reality, and Java applets.

The global publishing format of the Internet is HTML. It allows authors to use not only text but also format that text with headings, lists, and tables, and to include still images, video, and sound within text. Readers can access pages of information from anywhere in the world at the click of a mouse-button. Information can be downloaded to the reader’s own PC or workstation. HTML pages can also be used for entering data and as the front-end for commercial transactions.

**2.2.2 Features of HTML**

* It is not a programming language.
* It is not a data description language.
* It is simple to understand and implement.
* HTML constructs a very easy to comprehend, and can be used effectively by anybody.
* The methodology used by HTML to mark up information is independent of its representation on a particular hardware or software architecture.
* HTML syntax is a worldwide standard.

**2.3 CSS**

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

Cascading Style Sheets, fondly referred to as CSS, is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.

CSS is easy to learn and understood but it provides powerful control over the presentation of an HTML document.

WHY CSS?

* CSS saves time: You can write CSS once and reuse same sheet in multiple HTML pages.
* Easy Maintenance: To make a global change simply change the style, and all elements in all the webpages will be updated automatically.
* Search Engines: CSS is considered as clean coding technique, which means search engines won’t have to struggle to “read” its content.
* Superior styles to HTML: CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* Offline Browsing: CSS can store web applications locally with the help of offline cache.Using of this we can view offline websites.

CSS Syntax:--

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document.

A style rule set consists of a selector and declaration block.

Selector => h1

Declaration => {color:blue;font size:12px;}

The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

For Example:

->color is property and blue is value.

->font size is property and 12px is value.

A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces

**2.4 JAVA SCRIPT**

* JavaScript is a very powerful client-side scripting language.
* JavaScript is used mainly for enhancing the interaction of a user with the webpage.
* In other words, you can make your webpage more lively and interactive, with the help of JavaScript.
* JavaScript is also being used widely in game development and Mobile application development
* The language was initially called Live Script and was later renamed JavaScript.
* There are many programmers who think that JavaScript and Java are the same.
* In fact, JavaScript and Java are very much unrelated.
* Java is a very complex programming language whereas JavaScript is only a scripting language.
* The syntax of JavaScript is mostly influenced by the programming language C.
* JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.g. having complex animations, clickable buttons, popup menus, etc.).
* There are also more advanced server side versions of JavaScript such as Node.Js which allow you to add more functionality to a website than simply downloading files (such as real time collaboration between multiple computers). Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them.
* JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements.
* Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:
* Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model (DOM).
* For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
* Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server.
* For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.
* This means that in the browser, JavaScript can change the way the webpage (DOM) looks. And, likewise, Node.js JavaScript on the server can respond to custom requests from code written in the browser.

**2.5 MY SQL**

MySQL is an open source relational database management system (RDBMS). Its name is a combination of "My”, the name of co-founder Michael Widenius's daughter and "SQL", the abbreviation for Structured Query Language.

MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation).[8] In 2010, when Oracle acquired Sun, Widenius forked the open-source MySQL project to create MariaDB.

MySQL is a component of the LAMP web application software stack (and others), which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and Word Press. MySQL is also used by many popular websites, including Google (though not for searches), Facebook, Twitter, Flickr, and YouTube.

MySQL is written in C and C++. Its SQL parser is written in yacc, but it uses a home-brewed lexical analyzer. MySQL works on many system platforms, including AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, macOS, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris, OS/2 Warp, QNX, Oracle Solaris, Symbian, SunOS, SCO OpenServer, SCO UnixWare, Sanos and Tru64. A port of MySQL to OpenVMS also exists.

The MySQL server software itself and the client libraries use dual-licensing distribution. They are offered under GPL version 2, or a proprietary license.

Support can be obtained from the official manual. Free support additionally is available in different IRC channels and forums. Oracle offers paid support via its MySQL Enterprise products. They differ in the scope of services and in price. Additionally, a number of third party organizations exist to provide support and services, including MariaDB and Percona.

MySQL has received positive reviews, and reviewers noticed it "performs extremely well in the average case" and that the "developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good". It has also been tested to be a "fast, stable and true multi-user, multi-threaded sql database server".

**3. Project subject**

**3.1 Modules**

**Administrator**- person is responsible for monitoring functions and procedures of the platform. Administrator is responsible to provide valid information of the product and purchase.

**Customers**-he or she is a verified user for website who is intended to buy a product from the website. The customer must have valid username and password to make a purchase. The person is updated and fed with the latest offers and discounts.

**Supplier-** the person who supplies products to the admin whenever the stock is empty. The records of the stock is maintained by the admin.

**4. Software Requirements and Analysis**

Table of Contents

**Table of contents--------------------------------------------------------------**

**Revision History---------------------------------------------------------------**

1. **Introduction---------------------------------------------------------------**

1.1 Purpose ------------------------------------------------------------------

1.2 Document Convention ------------------------------------------------

1.3 Product Scope -----------------------------------------------------------

1.4 References ---------------------------------------------------------------

2. **Overall Description-------------------------------------------------------**

2.1 Product Perspective ----------------------------------------------------

2.2 Product Functions- -----------------------------------------------------

2.3 User Classes and Characteristics -------------------------------------

2.4 Operating Environment ------------------------------------------------

2.5 Design and Implementation Constraints ----------------------------

2.6 User Documentation ----------------------------------------------------

2.7 Assumptions and Dependencies --------------------------------------

3. **External Interface Requirements ---------------------------------------**

3.1 User Interfaces -----------------------------------------------------------

3.2 Hardware Interfaces ----------------------------------------------------

3.3 Software Interfaces------------------------------------------------------

3.4 Communications Interfaces --------------------------------------------

4. **System Features ------------------------------------------------------------**

4.1 System Feature 1 ---------------------------------------------------------

4.2 System Feature 2 (and so on) ------------------------------------------

5. **Other Nonfunctional Requirements -------------------------------------**

5.1 Performance Requirements ---------------------------------------------

5.2 Safety Requirements ------------------------------------------------------

5.3 Security Requirements ----------------------------------------------------

5.4 Software Quality Attributes -----------------------------------------

**1) INTRODUCTION**

* 1. Purpose

The “online shopping system” has been developed to override the problems prevailing in the practicing manual system. This system is designed for the particular need of the client to carry out operations in a smooth and effective manner. Our project has challenges to overcome and managing the information of product, customer, shopping, payment, delivery, services. The purpose of online shopping system is to automate the existing manual system by the help of computerized equipment and full-fledged computer software, fulfilling their requirements so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. Basically the project describes how to manage for good performance and better services for the clients.

* 1. Document Conventions

|  |  |
| --- | --- |
| SRS | System Requirement Specification |
| DFD | Data Flow Diagram |
| DB | Database |
| ER | Entity Relationship |
| FAQ | Frequently Asked Questions |

* 1. Product Scope

The application provides a computerized data storage facility. This system is used to maintain the data about stock, purchase, transactions etc. This system provides better transactions use. We can track our products easily. New system provides security against unauthorized access. Also provides better services to the customers.

* 1. References

Fundamentals of database systems by Ramez.Elmarsi and Shamkant.B.Navathe

**2) Overall Description**

2.1 Product Perspective

Bombay Novelty E-Store website is a outstanding way of bringing sellers and customers on a online platform to make purchase in a secures and efficient manner. This website offers recommendations for discounts and offers on products based on their category.

2.2 Product Functions

* Provide a simple interface and platform to ease process of buying as well as selling of products online.
* Include smooth functionality and efficiency that adds to buyers confidence.
* Appropriate space is dedicated to Offers that fetches adequate attention of users.
* Feature of MBScheme develop inquisitiveness towards users to buy new product.
* Tracking feature of orders and pending deliveries for both buyers and sellers.
* Notifying orders and delivery reports to customers and sellers.

2.3 Users Classes And Charactertics

**Administrator**- person is responsible for monitoring functions and procedures of the platform.Administrator are responsible to provide valid information of the product and purchase.

**Customers**-he or she is a verified user for website who is intended to buy a product from the website. The customer must have valid username and password to make a purchase. The person is updated and fed with the latest offers and discounts.

**Supplier-** the person who supplies products to the admin whenever the stock is empty.The records of the stock is maintained by the admin.

2.4 Operating Environment

Recommended browsers are Chrome, Firefox and Internet Explorer or higher.

|  |  |
| --- | --- |
| Particulars | Client System |
| Operating System | Windows 7 and others |
| Processor | Intel |
| Hard disk | 500GB |
| RAM | 2GB |

2.5 Design and Implementation Constraints

|  |  |
| --- | --- |
| Client-side programming language | |
| HTML | HTML is a standard markup language for creating web pages and web applications. |
| CSS | Cascading Style Sheets define style rules in a separate CSS file |
| JavaScript | JavaScript is a lightweight, object-oriented, cross-platform scripting language, mainly used within web pages |
| Python | Python is a interpreted, high-level, general-purpose programming language. |

2.6 User Documentation

* Notifications
* Contact Us
* Make a payment
* Submit account details
* Payment security

2.7 Assumptions and Dependencies

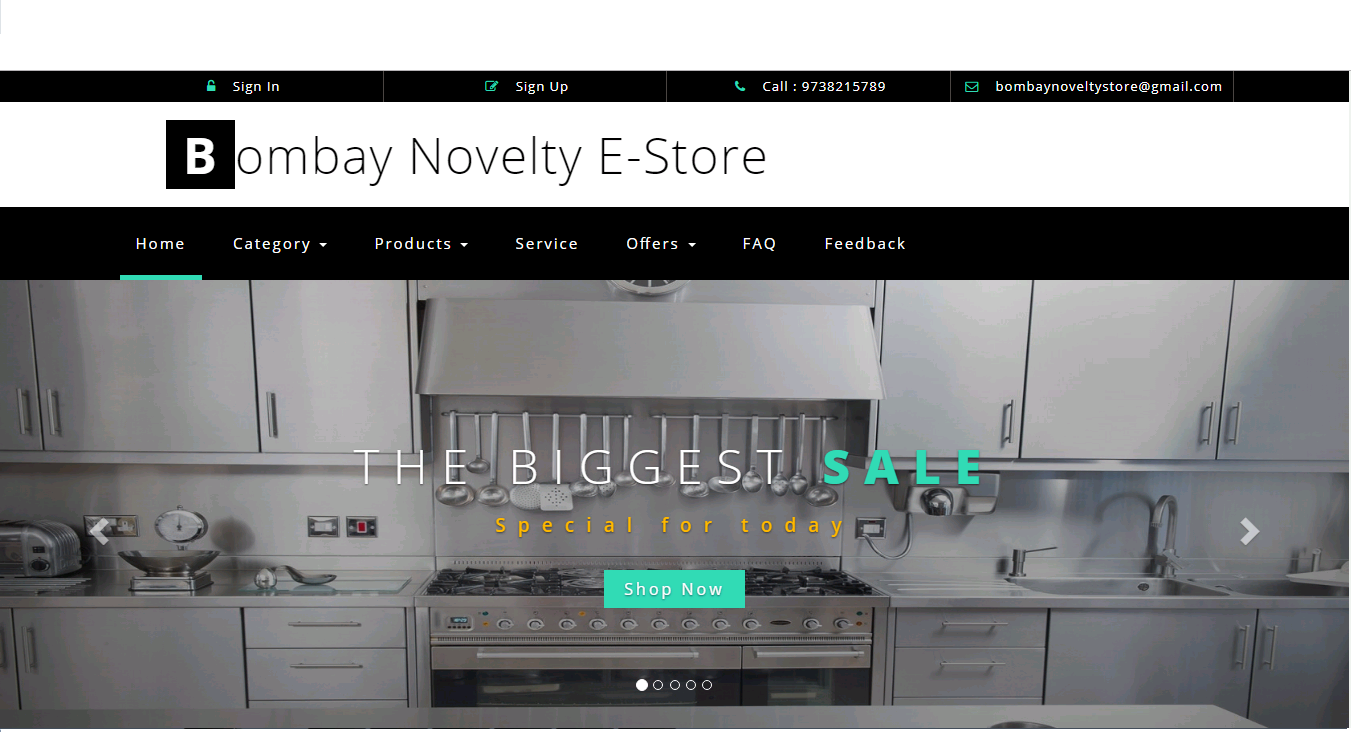
* Although our software presents a broad range of options to its users some intricate options could not be covered into it.
* Single language facility can be given presently.
* Presently limited stocks are available.
* Cannot do online transactions.
* Invoice messages like SMS cannot be provided currently.

**3) External Interface Requirements**

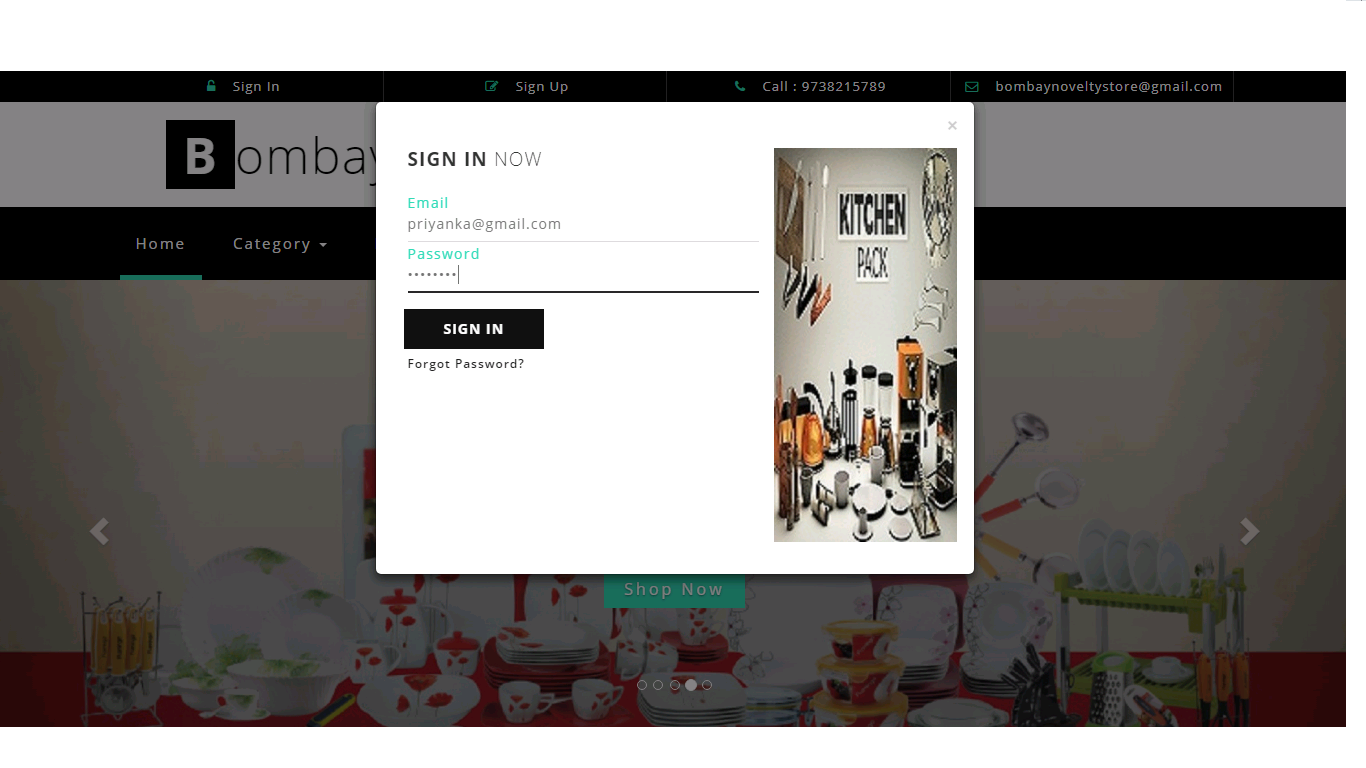
3.1 User Interfaces

* Login or Signup page
* Homepage containing products according to customer’s interest
* If the customer selects particular product then the information about that product is displayed in another tab.
* After all the transactions the system makes the selling report as portable document file(pdf) and sent to the customer E-mail address.

**Homepage**



**Login page**



3.2 Hardware Interfaces

Since the application must run over the internet, all the hardware shall require to connect internet will be hardware interface for the system. As for example-Modem, WAN-LAN, Ethernet cross cable. The system require database to store any transactions of the system like MySql, system also require DNS(domain name server) for naming on the internet. At the last user need web browser for interact with the system.

|  |  |
| --- | --- |
| Web Server Deployment And Technologies | |
| MySQL | MySQL database for storage of data |

3.3 Software Interfaces

Following are the software used for the application.

|  |  |
| --- | --- |
| Software used | Description |
| Operating system | We have chosen Windows operating system for its best support and user-friendliness. |
| Database | To save the customer records, product records we have chosen MySQL database |
| Python | To implement the project we have chosenPython as web programming. |

3.4 Communication Interfaces

The E-store system shall use the HTTPS protocol for communication over the internet and for the intranet communication through TCP/IP protocol suite.

**4) System Features**

The online shopping system manages the information of product, customer, order, payment, delivery and services. The following are some system features:

a) Registration/Login

Description:-

If a user wants to buy a product then he/she must be registered.

Only the registered user can login.

1) If the user logins with the invalid username and password

Stimulus/Response sequences:-

“Invalid password” or “Invalid username”

Functional requirements:-

User should login to the system by entering valid user id and password.

2) Order

Description:-

Through logining in, the customer can view and order the desired product.

Stimulus/Response sequences:-

“1 product ordered”

Functional requirements:-

If the customer needs to buy a product then the customer should login. Later he/she can view and order the product.

3) Payment

Description:-

For customers, there are many type of secure billing will be prepaid as credit or debit card and postpaid as after the shipping

Stimulus/Response sequence:-

“Thank you for the order”

Functional requirements:-

Customer orders the product and later moves to the payment details.

4) Administrator

Description:-

Administrator manages all the activities present in the system.

Functional requirements:-

Admin holds the information about product and customers of the system.

5)Notifications

Description:-

Customer buys a product, pays the payment and receive the notifications

Stimulus/Response sequence:-

“Message has been sent”

Functional requirements:-

Product is been bought by the customer by the process of order and payment. Inorder to assure the transactions notification is provided.

**5. Data Flow Diagrams**

**5.1 Data Flow Diagram**

The data flow diagram (DFD) is one of the important modeling tools. It shows the user of the data pictorially. DFD represents the flow of the data between the transformation and processes in the system. The data flow diagram shows the logical flow of the data. It represents the functional dependencies within a system. It shows the output values in a computation are derived from input values.

**5.2 DFD Notations**

* In the DFD, there are four symbols
* A square defines a source(originator) or destination of system data
* An arrow identifies data flow. It is the pipeline through which the information flows
* A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
* An open rectangle is a data store, data at rest or a temporary repository of data

Source or Destination of data

Data flow

Data Store

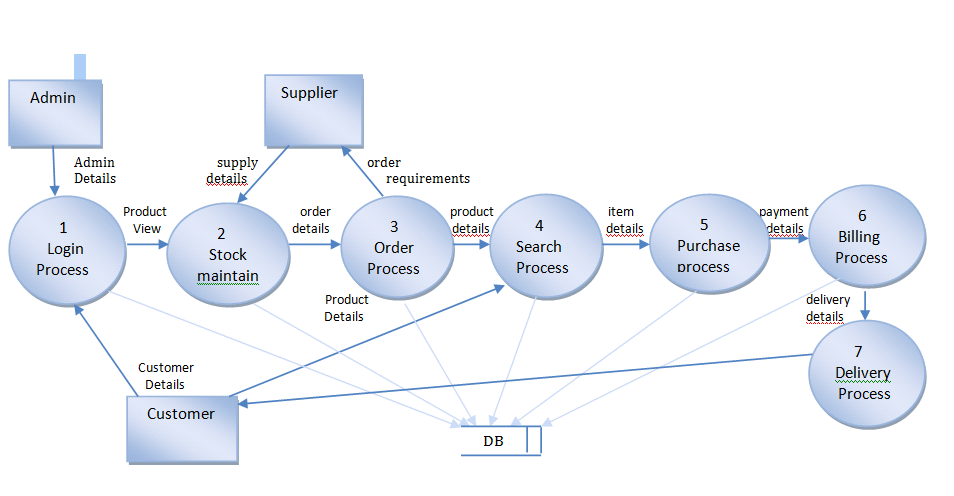
Process that transforms data flow

**5.3 Steps Required In DFD**

Several rules of thumb are used in drawing DFD’S:

* Process should be named and numbered for an easy reference. Each name should be representative of the process.
* The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
* When a process is exploded into lower level details, they are numbered.
* The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each work capitalized

**5.5 Level 1 DFD**

****

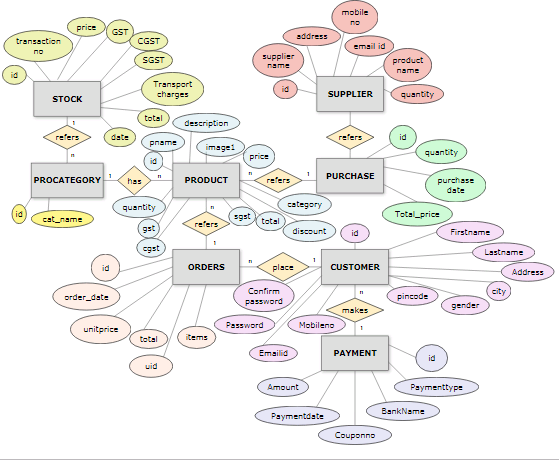
**6. ER Diagrams**

The entity relationship model is based on a perception of the world as consisting of a collection of basic objects (entities) and relationships among these objects.

* An entity is a distinguishable object that exists.
* Each entity has associated with it a set of attributes describing it.
* A relationship is an association among several entities.
* The set of entities or relationships of the same type is called the entity set relation set.
* Another important element of the E-R diagram is the mapping the cardinalities, which express the number of the entities to which another entity can be associated via a relation set.

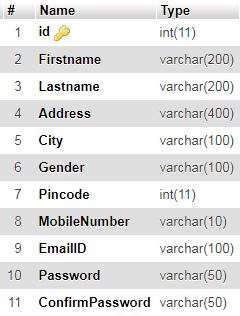
The overall logical structure of a database can be expressed graphically by E-R diagram:

* Rectangle: represent entity sets.
* Ellipses: represent attributes.
* Diamonds: represent relationships among entity sets.
* Lines: link attribute to entity sets and entity sets to relationships.

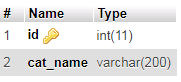


**7. Tables used in the database**

**Userregistration**

****

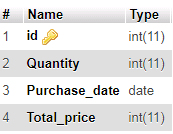
**Procategory**

****

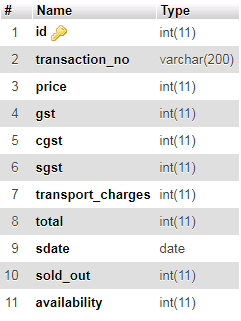
**Product**

****

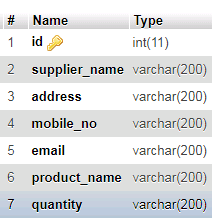
**Purchase Items**

****

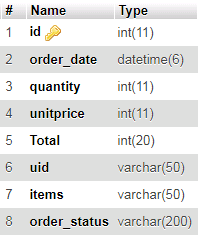
**Stockitems**

****

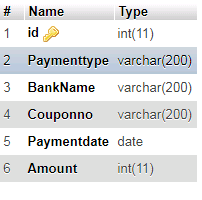
**Supplier**

****

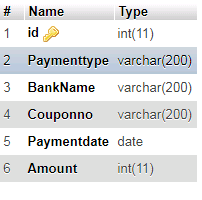
**Order\_details**

****

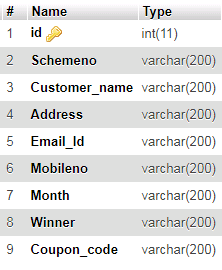
**Payment**

****

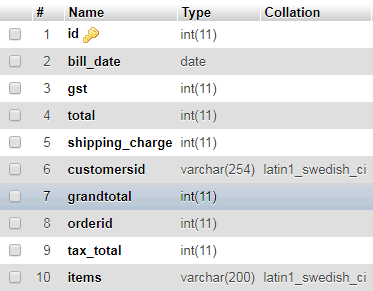
**Exchange offer**

****

**MBScheme**

****

**Billing**

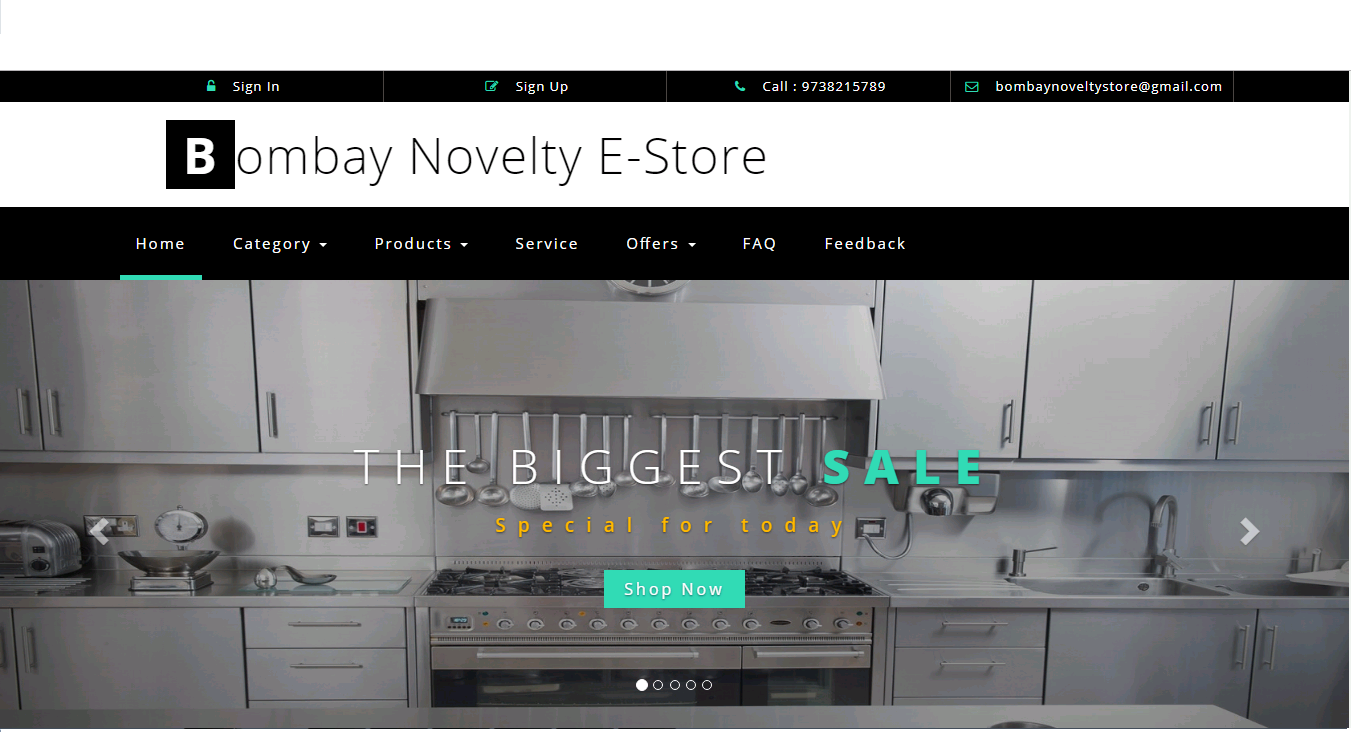
****

**Feedback**

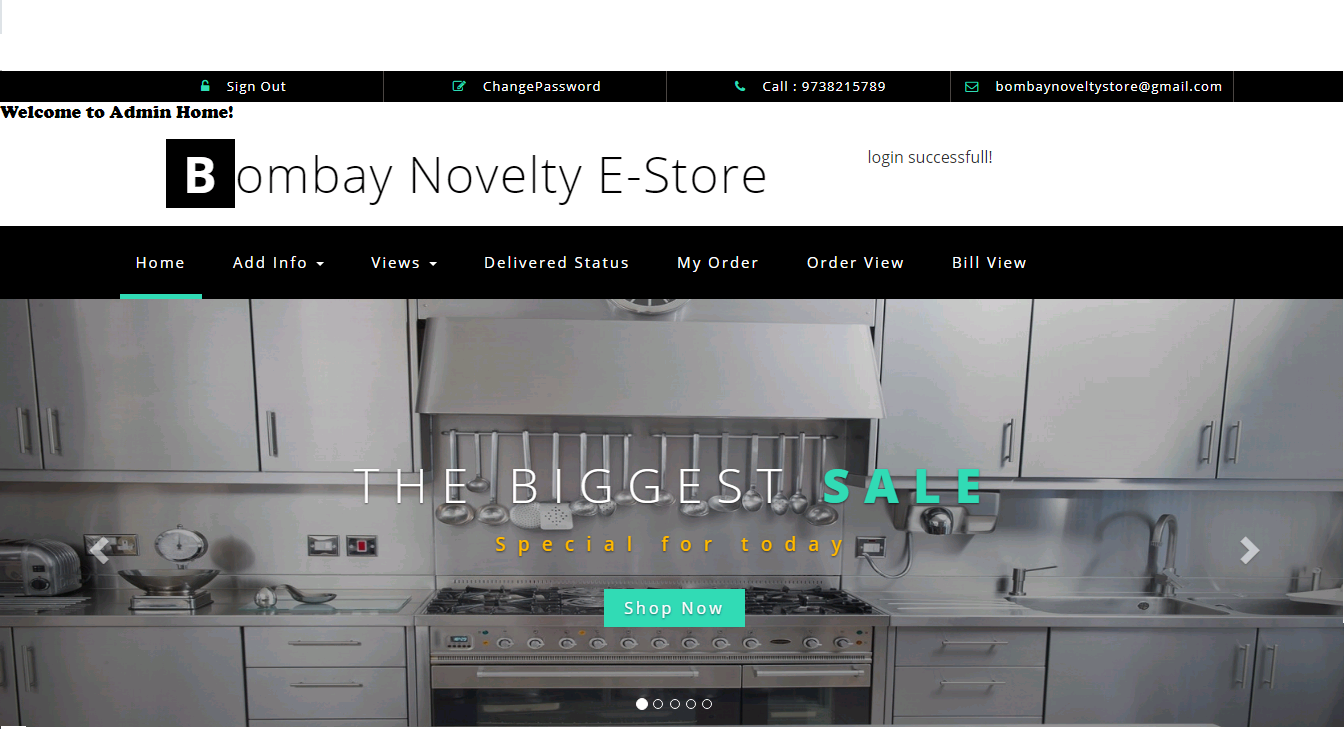
****

**8. Sample outputs /Snapshots**

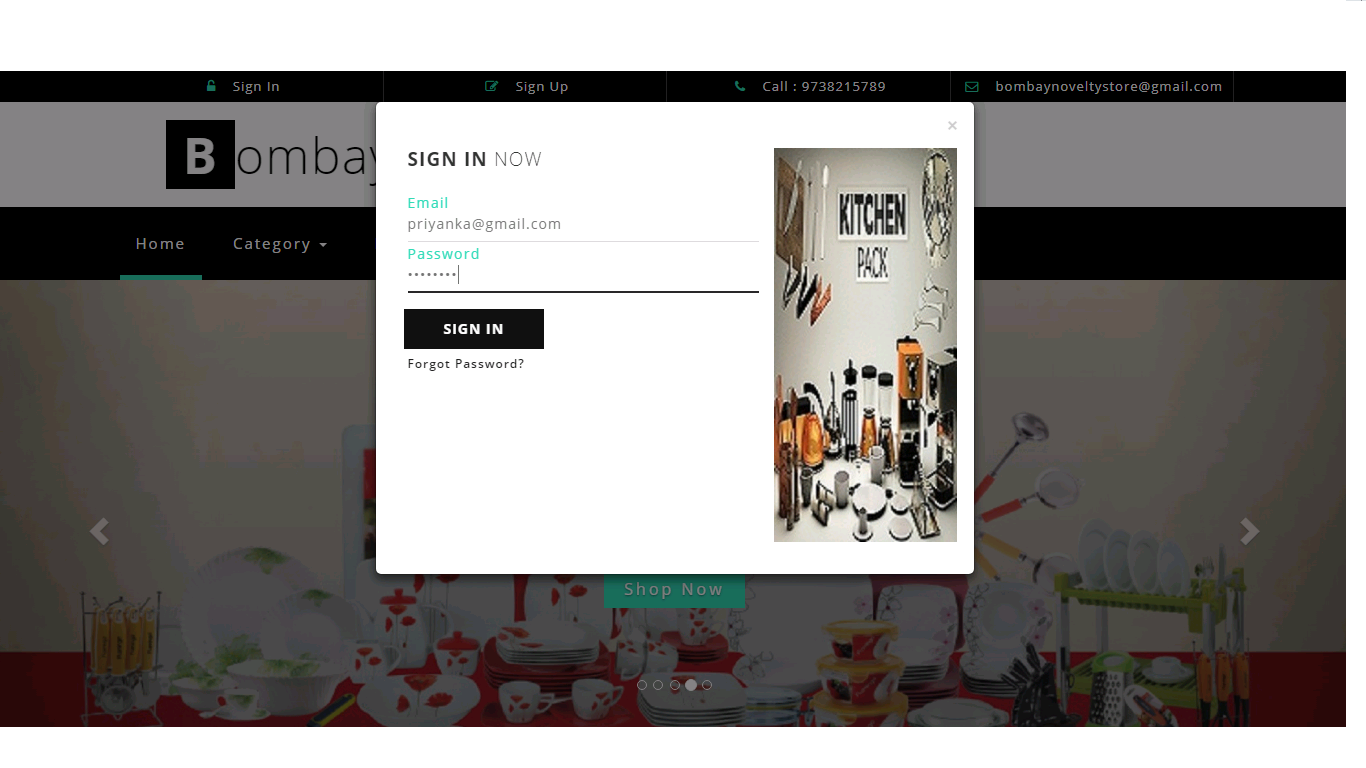
**Home Page**

****

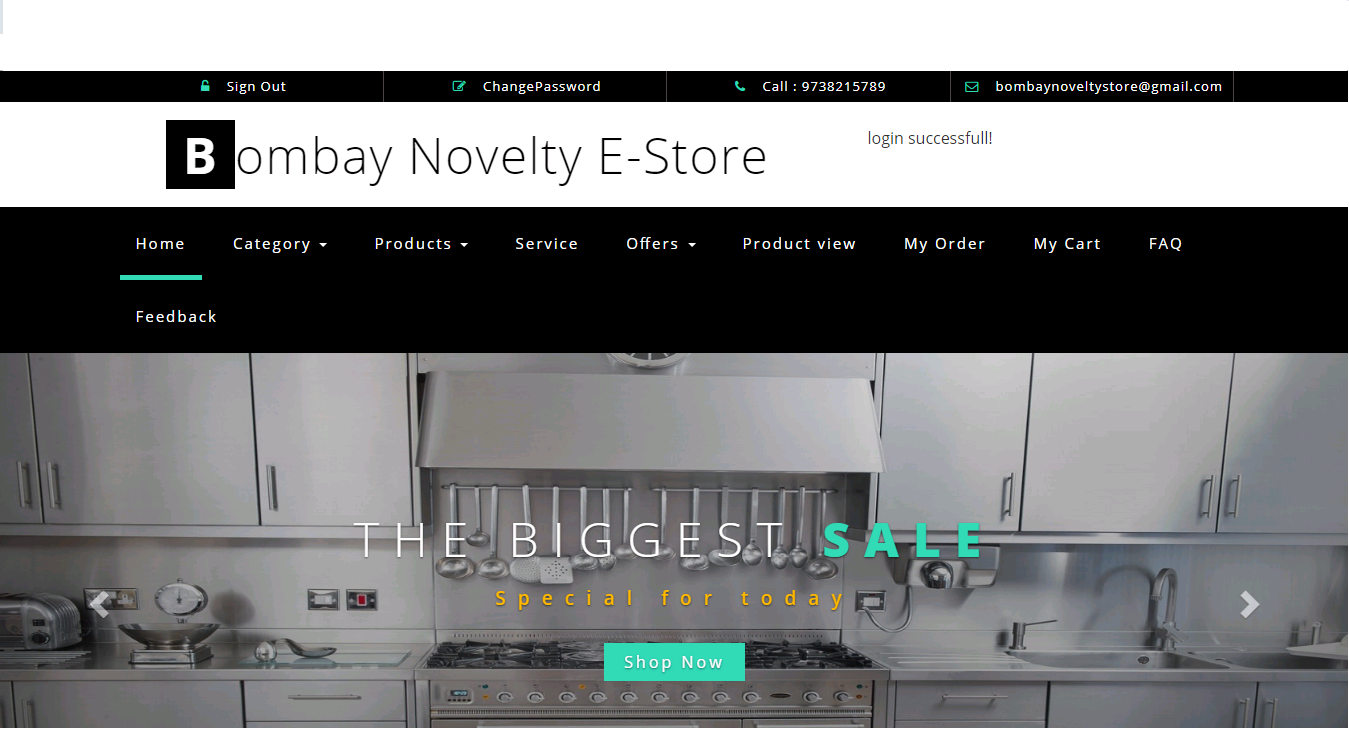
**Admin Page**

****

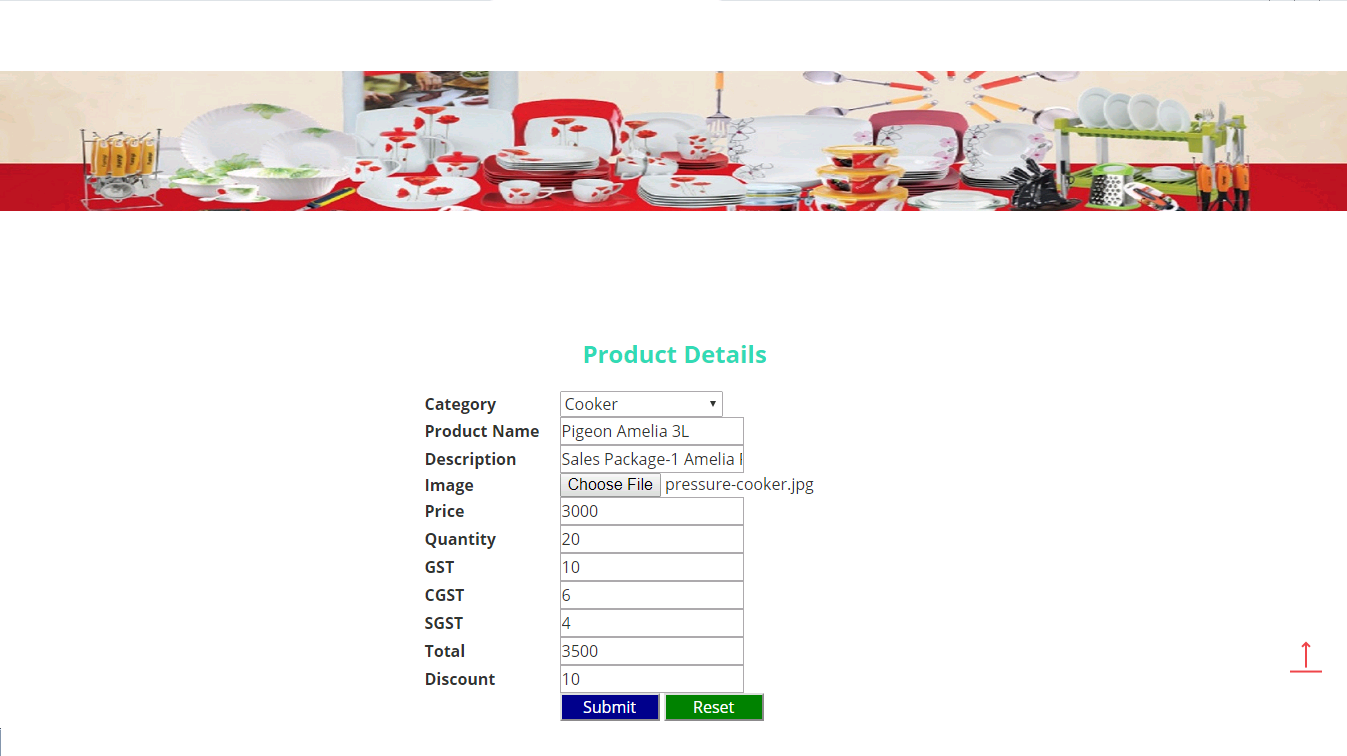
**Login**

****

**Login Successful**

****

**Product insert**

****

**Product View**

****

**Product Update**

****

**9. Source Code**

**Python code for creating models**

**from django.db import models**

**class SignUpForm(models.Model):**

**uname = models.EmailField(default='')**

**class UserRegistration(models. Model):**

**Firstname = models.CharField(default='', max\_length=200)**

**Lastname = models.CharField(default='', max\_length=200)**

**Gender = models.CharField(max\_length=100)**

**Address = models.CharField(max\_length=400)**

**City = models.CharField(max\_length=100)**

**Pincode = models.IntegerField(null=True,blank=True)**

**MobileNumber = models.CharField(max\_length=10)**

**EmailID = models.EmailField(max\_length=100)**

**Password = models.CharField(max\_length=50)**

**ConfirmPassword = models.CharField(max\_length=50)**

**class UserLogin(models.Model):**

**emailid = models.CharField(max\_length=200)**

**password = models.CharField(max\_length=50)**

**utype = models.CharField(max\_length=50)**

**class ProCategory(models.Model):**

**cat\_name = models.CharField(default='', max\_length=200)**

**class Product(models.Model):**

**pname = models.CharField(default='', max\_length=200)**

**category = models.CharField(max\_length=200)**

**description = models.CharField(max\_length=500)**

**image1 = models.FileField(upload\_to='documents/')**

**price = models.IntegerField()**

**quantity = models.IntegerField()**

**gst = models.IntegerField(default='')**

**cgst = models.IntegerField(default='')**

**sgst = models.IntegerField(default='')**

**total = models.IntegerField(default='')**

**discount = models.IntegerField()**

**class StockItems(models.Model):**

**transaction\_no = models.CharField(max\_length=200)**

**price = models.IntegerField()**

**gst = models.IntegerField()**

**cgst = models.IntegerField()**

**sgst = models.IntegerField()**

**transport\_charges = models.IntegerField()**

**sdate = models.DateField()**

**total = models.IntegerField()**

**sold\_out = models.IntegerField()**

**availability = models.IntegerField()**

**class Supplier(models.Model):**

**supplier\_name = models.CharField(default='', max\_length=200)**

**address = models.CharField(max\_length=200)**

**mobile\_no = models.CharField(max\_length=200)**

**email = models.EmailField(max\_length=200)**

**product\_name = models.CharField(max\_length=200)**

**quantity = models.IntegerField()**

**class Billing(models.Model):**

**orderid = models.IntegerField(null=True, blank=True)**

**customersid = models.EmailField(null=True, blank=True)**

**bill\_date = models.DateField(auto\_now\_add=True)**

**items = models.CharField(default='', max\_length=200)**

**total = models.IntegerField(null=True, blank=True)**

**gst = models.IntegerField(null=True, blank=True)**

**tax\_total = models.IntegerField(null=True, blank=True)**

**shipping\_charge = models.IntegerField(null=True, blank=True)**

**grandtotal = models.IntegerField(null=True, blank=True)**

**class Payment(models.Model):**

**Paymenttype = models.CharField(default='', max\_length=200)**

**BankName = models.CharField(default='', max\_length=200)**

**Couponno = models.CharField(max\_length=200)**

**Amount = models.IntegerField(null=True, blank=True)**

**Paymentdate = models.DateField(auto\_now\_add=True)**

**class Deliverydtails(models.Model):**

**Deliverydate = models.DateField()**

**Trackingno = models.CharField(max\_length=200)**

**Packingstatus = models.CharField(max\_length=200)**

**Shippingstatus = models.CharField(max\_length=200)**

**Delivery\_status = models.CharField(max\_length=200)**

**class Purchase\_Items(models.Model):**

**Quantity = models.IntegerField()**

**Purchase\_date = models.DateField()**

**Total\_price = models.IntegerField()**

**class order\_details(models.Model):**

**order\_date = models.DateTimeField(auto\_now\_add=True)**

**quantity = models.IntegerField(null=True, blank=True)**

**unitprice = models.IntegerField(null=True, blank=True)**

**Total = models.IntegerField(null=True)**

**uid = models.CharField(max\_length=50, null=True)**

**items = models.CharField(max\_length=50, null=True)**

**order\_status = models.CharField(default='', max\_length=200)**

**class MB\_scheme(models.Model):**

**id = models.AutoField(primary\_key=True)**

**Schemeno = models.CharField(max\_length=200)**

**Customer\_name = models.CharField(max\_length=200)**

**Address = models.CharField(max\_length=200)**

**Email\_Id = models.EmailField(max\_length=200)**

**Mobileno = models.CharField(max\_length=200)**

**Month = models.CharField(max\_length=200)**

**Winner = models.CharField(max\_length=200)**

**Coupon\_code = models.CharField(max\_length=200)**

**class ExchangeOffer(models.Model):**

**exchngname = models.CharField(default='', max\_length=200)**

**dscrptn = models.CharField(max\_length=500)**

**discount = models.IntegerField()**

**Start\_date = models.DateField()**

**finish\_date = models.DateField()**

**coupon\_code = models.CharField(max\_length=200)**

**class Service(models.Model):**

**Product\_name = models.CharField(default='', max\_length=200)**

**Description = models.CharField(max\_length=500)**

**Price = models.IntegerField()**

**Service\_men\_name = models.CharField(default='', max\_length=200)**

**Service\_men\_Mbno = models.CharField(max\_length=200)**

**class FeedBackForm(models.Model):**

**userid = models.EmailField()**

**servicefeedback = models.CharField(max\_length=200)**

**comments = models.CharField(max\_length=200)**

**Views.py**

**from django.shortcuts import render**

**#from appliance\_app.models import SignUpForm**

**from appliance\_app.models import UserRegistration**

**from appliance\_app.models import UserLogin**

**from appliance\_app.models import ProCategory**

**from appliance\_app.models import ExchangeOffer**

**from appliance\_app.models import Supplier**

**from appliance\_app.models import Purchase\_Items**

**from appliance\_app.models import StockItems**

**from appliance\_app.models import Product**

**from appliance\_app.models import Service**

**from appliance\_app.models import MB\_scheme**

**from appliance\_app.models import order\_details**

**from appliance\_app.models import FeedBackForm**

**from appliance\_app.models import Billing**

**from appliance\_app.models import Payment**

**from datetime import datetime**

**import os**

**import smtplib**

**from django.conf import settings**

**from django.contrib import messages**

**from django.core.mail import send\_mail**

**from django.core.files.storage import FileSystemStorage**

**from appliance.settings import BASE\_DIR**

**#Create your views here.**

**# def thankyou(request):**

**# form = SignUpForm(request.POST or None)**

**# if form.is\_valid():**

**# save\_it = form.save(commit=False)**

**# save\_it.save()**

**# subject = 'Thankyou for ur order'**

**# message = 'welcome '**

**# from\_email = settings.EMAIL\_HOST\_USER**

**# to\_list = [save\_it.email, settings.EMAIL\_HOST\_USER]**

**# send\_mail(subject, message, from\_email, to\_list, fail\_silently=True)**

**# messages.success(request, 'thankyou')**

**# return HttpResponseRedirect('/thankyou/')**

**# return render\_to\_response("thankyou.html", locals(), context\_instance=RequestContext(request))**

**def index(request):**

**user\_dict = Product.objects.all()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'index.html', {'user\_dict': user\_dict, 'prod\_cat': prod\_cat, 'prod\_list': prod\_list })**

**def admins(request):**

**user\_dict = ''**

**return render(request, 'admin\_home.html', {'user\_dict': user\_dict})**

**def customers(request):**

**user\_dict = ()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'customer\_home.html', {'user\_dict': user\_dict, 'prod\_cat': prod\_cat, 'prod\_list': prod\_list, 'msg':'payment has been done successfully!'})**

**def customers\_next(request):**

**content = "From Bombay Novelt....Keep shopping:)! "**

**mail = smtplib.SMTP('smtp.gmail.com', 587)**

**mail.ehlo()**

**mail.starttls()**

**mail.login('priyankavrokhade@gmail.com', 'pinku@yo@1997')**

**mail.sendmail('priyankavrokhade@gmail.com', 'rokhadeaishu@gmail.com', content)**

**mail.close()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'customer\_home.html', {'prod\_cat': prod\_cat, 'prod\_list': prod\_list, 'msg':'payment has been done successfully!'})**

**return render(request, 'send\_email.html')**

**def signup(request):**

**user\_dict = {'msg': 'login successfull!'}**

**if request.method == "POST":**

**Firstname = request.POST.get('t1','')**

**Lastname = request.POST.get('t2','')**

**Gender= request.POST.get('t3','')**

**Address = request.POST.get('t4','')**

**City = request.POST.get('t5','')**

**Pincode = request.POST.get('t6','')**

**MobileNumber = request.POST.get('t7','')**

**EmailID = request.POST.get('t8','')**

**Password = request.POST.get('t9','')**

**ConfirmPassword = request.POST.get('t10','')**

**UserRegistration.objects.create(Firstname=Firstname, Lastname=Lastname, Gender=Gender, Address=Address, City=City, Pincode=Pincode, MobileNumber=MobileNumber, EmailID=EmailID, Password=Password, ConfirmPassword=ConfirmPassword)**

**UserLogin.objects.create(emailid=EmailID, password=Password, utype='customer')**

**return render(request, 'customer\_home.html', context=user\_dict)**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'index.html', {'prod\_cat':prod\_cat, 'prod\_list':prod\_list})**

**def sampleview(request):**

**user\_dict = UserRegistration.objects.all()**

**return render(request, 'sampleview.html', {'user\_dict': user\_dict})**

**def remove\_items(request):**

**if request.method == 'POST':**

**userdata = UserRegistration.objects.all()**

**id = request.POST.get('id')**

**useritem = UserRegistration.objects.get(id=id)**

**useritem.delete()**

**useritem = UserRegistration.objects.all()**

**return render(request, 'sampleview.html', context={'user\_dict': useritem})**

**def update\_items(request):**

**if request.method == 'POST':**

**userdata = UserRegistration.objects.all()**

**id = request.POST.get('id')**

**useritem = UserRegistration.objects.filter(id=id).values()**

**return render(request, 'sampleedit.html', {'useritem': useritem})**

**def reg\_db(request):**

**if request.method == "POST":**

**userdata = UserRegistration.objects.all()**

**id = request.POST.get('id')**

**useritem = UserRegistration.objects.filter(id=id).values()**

**Firstname = request.POST.get('t1', '')**

**Lastname = request.POST.get('t2', '')**

**Gender = request.POST.get('t3', '')**

**Address = request.POST.get('t4', '')**

**City = request.POST.get('t5', '')**

**Pincode = request.POST.get('t6', '')**

**MobileNumber = request.POST.get('t7', '')**

**EmailID = request.POST.get('t8', '')**

**Password = request.POST.get('t9', '')**

**ConfirmPassword = request.POST.get('t10', '')**

**UserRegistration.objects.filter(id=id).update(Firstname=Firstname, Lastname=Lastname, Gender=Gender, Address=Address,**

**City=City, Pincode=Pincode, MobileNumber=MobileNumber, EmailID=EmailID,**

**Password=Password, ConfirmPassword=ConfirmPassword)**

**useritem = UserRegistration.objects.all()**

**return render(request, 'sampleview.html', context={'user\_dict': useritem})**

**def signin(request):**

**user\_dict = {'msg': 'login successfull!'}**

**if request.method == "POST":**

**emailid = request.POST.get('Email', '')**

**password = request.POST.get('ps', '')**

**utype = request.POST.get('user', '')**

**checklogin = UserLogin.objects.filter(emailid=emailid).values()**

**for a in checklogin:**

**type = a['utype']**

**upass = a['password']**

**request.session['uid'] = emailid**

**if upass == password:**

**if type == "customer":**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'customer\_home.html', {'prod\_cat': prod\_cat, 'prod\_list':prod\_list, 'msg':'login successfull!'})**

**if type == "admin":**

**prod\_list = ProCategory.objects.all()**

**return render(request, 'admin\_home.html', {'prod\_list': prod\_list, 'msg':'login successfull!'})**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'index.html', {'prod\_cat': prod\_cat, 'prod\_list': prod\_list})**

**def Category(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**cname = request.POST.get('p2','')**

**ProCategory.objects.create(cat\_name=cname)**

**return render(request, 'admin\_home.html', context=user\_dict)**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'Category.html', {'prod\_cat': prod\_cat, 'prod\_list':prod\_list, 'user\_dict': user\_dict})**

**def categoryview(request):**

**user\_dict = ProCategory.objects.all()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'categoryview.html', {'prod\_cat': prod\_cat, 'prod\_list': prod\_list, 'user\_dict': user\_dict})**

**def catremove\_items(request):**

**if request.method == 'POST':**

**userdatas = ProCategory.objects.all()**

**id = request.POST.get('id')**

**useritems = ProCategory.objects.get(id=id)**

**useritems.delete()**

**useritems = ProCategory.objects.all()**

**return render(request,'categoryview.html',context={'user\_dict': useritems})**

**def catupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = ProCategory.objects.all()**

**id = request.POST.get('id')**

**useritems = ProCategory.objects.filter(id=id).values()**

**return render(request, 'catedit.html', {'useritems': useritems})**

**def cat\_db(request):**

**if request.method == "POST":**

**userdatas = ProCategory.objects.all()**

**id = request.POST.get('id')**

**useritems = ProCategory.objects.filter(id=id).values()**

**cname = request.POST.get('p2', '')**

**ProCategory.objects.filter(id=id).update(cat\_name=cname)**

**useritems = ProCategory.objects.all()**

**return render(request, 'categoryview.html', context={'user\_dict': useritems})**

**def supplier(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**p1 = request.POST.get('r1')**

**p2 = request.POST.get('r2')**

**p3 = request.POST.get('r3')**

**p4 = request.POST.get('r4')**

**p5 = request.POST.get('r5')**

**p6 = request.POST.get('r6')**

**print(p1)**

**Supplier.objects.get\_or\_create(supplier\_name=p1, address=p2, mobile\_no=p3, email=p4, product\_name=p5, quantity=p6)**

**return render(request, 'admin\_home.html', context=user\_dict)**

**return render(request,'supplier.html')**

**def supplierview(request):**

**user\_dict = Supplier.objects.all()**

**return render(request, 'supplierview.html', {'user\_dict': user\_dict})**

**def supremove\_items(request):**

**if request.method == 'POST':**

**userdatas = Supplier.objects.all()**

**id = request.POST.get('id')**

**useritems = Supplier.objects.get(id=id)**

**useritems.delete()**

**useritems = Supplier.objects.all()**

**return render(request,'supplierview.html',context={'user\_dict': useritems})**

**def supupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = Supplier.objects.all()**

**id = request.POST.get('id')**

**useritems = Supplier.objects.filter(id=id).values()**

**return render(request, 'supplieredit.html', {'useritems': useritems})**

**def sup\_db(request):**

**if request.method == "POST":**

**userdatas = Supplier.objects.all()**

**id = request.POST.get('id')**

**useritems = Supplier.objects.filter(id=id).values()**

**supplier\_name = request.POST.get('r1', '')**

**address = request.POST.get('r2', '')**

**mobile\_no = request.POST.get('r3', '')**

**email = request.POST.get('r4', '')**

**product\_name = request.POST.get('r5', '')**

**quantity = request.POST.get('r6', '')**

**Supplier.objects.filter(id=id).update(supplier\_name=supplier\_name, address=address, mobile\_no=mobile\_no, email=email, product\_name=product\_name, quantity=quantity )**

**useritems = Supplier.objects.all()**

**return render(request, 'supplierview.html', context={'user\_dict': useritems})**

**def purchase(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**Quantity = request.POST.get('o1','')**

**Purchase\_date = request.POST.get('o2','')**

**Total\_price = request.POST.get('o3','')**

**Purchase\_Items.objects.create(Quantity=Quantity, Purchase\_date=Purchase\_date, Total\_price=Total\_price )**

**return render(request, 'admin\_home.html', context=user\_dict)**

**return render(request, 'Purchase\_Items.html')**

**def purchaseview(request):**

**user\_dict = Purchase\_Items.objects.all()**

**return render(request, 'purchaseview.html', {'user\_dict': user\_dict})**

**def purremove\_items(request):**

**if request.method == 'POST':**

**userdatas = Purchase\_Items.objects.all()**

**id = request.POST.get('id')**

**user\_items = Purchase\_Items.objects.get(id=id)**

**user\_items.delete()**

**user\_items = Purchase\_Items.objects.all()**

**return render(request,'purchaseview.html', context={'user\_dict': user\_items})**

**def purupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = Purchase\_Items.objects.all()**

**id = request.POST.get('id')**

**user\_items = Purchase\_Items.objects.filter(id=id).values()**

**return render(request, 'purchaseedit.html', {'user\_items': user\_items})**

**def pur\_db(request):**

**if request.method == "POST":**

**userdatas = Purchase\_Items.objects.all()**

**id = request.POST.get('id')**

**user\_items = Purchase\_Items.objects.filter(id=id).values()**

**Quantity = request.POST.get('o1', '')**

**Purchase\_date = request.POST.get('o2', '')**

**Total\_price = request.POST.get('o3', '')**

**Purchase\_Items.objects.filter(id=id).update(Quantity=Quantity, Purchase\_date=Purchase\_date, Total\_price=Total\_price)**

**user\_items = Purchase\_Items.objects.all()**

**return render(request, 'purchaseview.html', context={'user\_dict': user\_items})**

**def stock(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**transaction\_no = request.POST.get('q1', '')**

**price = request.POST.get('q2', '')**

**gst = request.POST.get('q3', '')**

**cgst = request.POST.get('q4', '')**

**sgst = request.POST.get('q5', '')**

**transport\_charges = request.POST.get('q6', '')**

**sdate = request.POST.get('q7', '')**

**total = request.POST.get('q8', '')**

**sold\_out = request.POST.get('q9', '')**

**availability = request.POST.get('q10', '')**

**StockItems.objects.create(transaction\_no=transaction\_no, price=price, gst=gst, cgst=cgst, sgst=sgst, transport\_charges=transport\_charges, total=total, sdate=sdate, sold\_out=sold\_out, availability=availability)**

**return render(request, 'admin\_home.html', context=user\_dict)**

**return render(request,'Stock\_Items.html')**

**def stockview(request):**

**user\_dict = StockItems.objects.all()**

**return render(request, 'stockview.html', {'user\_dict': user\_dict})**

**def stkremove\_items(request):**

**if request.method == 'POST':**

**userdatas = StockItems.objects.all()**

**id = request.POST.get('id')**

**useritems = StockItems.objects.get(id=id)**

**useritems.delete()**

**user\_items = StockItems.objects.all()**

**return render(request,'stockview.html',context={'user\_dict': user\_items})**

**def stkupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = StockItems.objects.all()**

**id = request.POST.get('id')**

**useritems = StockItems.objects.filter(id=id).values()**

**return render(request, 'stockedit.html', {'useritems': useritems})**

**def stk\_db(request):**

**if request.method == "POST":**

**userdatas = StockItems.objects.all()**

**id = request.POST.get('id')**

**useritems = StockItems.objects.filter(id=id).values()**

**transaction\_no = request.POST.get('q1', '')**

**price = request.POST.get('q2', '')**

**gst = request.POST.get('q3', '')**

**cgst = request.POST.get('q4', '')**

**sgst = request.POST.get('q5', '')**

**transport\_charges = request.POST.get('q6', '')**

**sdate = request.POST.get('q7', '')**

**total = request.POST.get('q8', '')**

**sold\_out = request.POST.get('q9', '')**

**availability = request.POST.get('q10', '')**

**StockItems.objects.filter(id=id).update(transaction\_no=transaction\_no, price=price, gst=gst, cgst=cgst, sgst=sgst, transport\_charges=transport\_charges, sdate=sdate, total=total, sold\_out=sold\_out, availability=availability)**

**user\_items = StockItems.objects.all()**

**return render(request, 'stockview.html', context={'user\_dict': user\_items})**

**def exchange(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**pname = request.POST.get('s1')**

**p2 = request.POST.get('n2')**

**p3 = request.POST.get('n3')**

**p4 = request.POST.get('n4')**

**p5 = request.POST.get('n5')**

**p6 = request.POST.get('n6')**

**ExchangeOffer.objects.create(exchngname=pname, dscrptn=p2, discount=p3, Start\_date=p4, finish\_date=p5, coupon\_code=p6)**

**eoffer\_data = ExchangeOffer.objects.all()**

**return render(request, 'admin\_home.html', {'user\_dict':eoffer\_data})**

**ex\_offer = Product.objects.all()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'Exchange\_Offer.html', {'prod\_list':ex\_offer, 'prod\_cat': prod\_cat, 'prod\_list':prod\_list })**

**def exchangeview(request):**

**user\_dict = ExchangeOffer.objects.all()**

**return render(request, 'exchangeview.html', {'user\_dict': user\_dict})**

**def exchgremove\_items(request):**

**if request.method == 'POST':**

**userdata = ExchangeOffer.objects.all()**

**id = request.POST.get('id')**

**useritem = ExchangeOffer.objects.get(id=id)**

**useritem.delete()**

**user\_item = ExchangeOffer.objects.all()**

**return render(request,'exchangeview.html',context={'user\_dict': user\_item})**

**def exchgupdate\_items(request):**

**if request.method == 'POST':**

**userdata = ExchangeOffer.objects.all()**

**id = request.POST.get('id')**

**useritem = ExchangeOffer.objects.filter(id=id).values()**

**return render(request, 'exchangeedit.html', {'useritem': useritem})**

**def ex\_db(request):**

**if request.method == "POST":**

**userdata = ExchangeOffer.objects.all()**

**id = request.POST.get('id')**

**useritem = ExchangeOffer.objects.filter(id=id).values()**

**exchngname = request.POST.get('n1', '')**

**dscrptn = request.POST.get('n2', '')**

**discount = request.POST.get('n3', '')**

**Start\_date = request.POST.get('n4', '')**

**finish\_date = request.POST.get('n5', '')**

**coupon\_code = request.POST.get('n6', '')**

**ExchangeOffer.objects.filter(id=id).update(exchngname=exchngname, dscrptn=dscrptn, discount=discount, Start\_date=Start\_date, finish\_date=finish\_date, coupon\_code=coupon\_code )**

**user\_item = ExchangeOffer.objects.all()**

**return render(request, 'exchangeview.html', context={'user\_dict': user\_item})**

**def product(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**cat\_name = request.POST.get('p2','')**

**pname = request.POST.get('s1', '')**

**description = request.POST.get('s2','')**

**image1 = request.FILES['s3']**

**fs = FileSystemStorage()**

**filename = fs.save(image1.name, image1)**

**uploaded\_file\_url = fs.url(filename)**

**pat = os.path.join(BASE\_DIR, '/media/'+filename)**

**price = request.POST.get('s4', '')**

**quantity = request.POST.get('s5', '')**

**gst = request.POST.get('s6', '')**

**cgst = request.POST.get('s7', '')**

**sgst = request.POST.get('s8', '')**

**total = request.POST.get('s9', '')**

**discount = request.POST.get('s10', '')**

**Product.objects.create(pname=pname, description=description, image1=image1, price=price, quantity=quantity, gst=gst, cgst=cgst, sgst=sgst, total=total, discount=discount,category=cat\_name)**

**return render(request, 'admin\_home.html', context=user\_dict)**

**prod\_list = ProCategory.objects.all()**

**return render(request,'product.html', {'prod\_list': prod\_list})**

**def productview(request):**

**user\_dict = Product.objects.all()**

**return render(request, 'productview.html', {'user\_dict': user\_dict})**

**def proremove\_items(request):**

**if request.method == 'POST':**

**userdatas = Product.objects.all()**

**id = request.POST.get('id')**

**user\_items = Product.objects.get(id=id)**

**user\_items.delete()**

**user\_items = Product.objects.all()**

**return render(request,'productview.html',context={'user\_dict': user\_items})**

**def proupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = Product.objects.all()**

**id = request.POST.get('id')**

**user\_items = Product.objects.filter(id=id).values()**

**return render(request, 'productedit.html', {'user\_items': user\_items})**

**def pro\_db(request):**

**if request.method == "POST":**

**userdatas = Supplier.objects.all()**

**id = request.POST.get('id')**

**user\_items = Product.objects.filter(id=id).values()**

**pname = request.POST.get('s1', '')**

**description = request.POST.get('s2', '')**

**image1 = request.POST.get('s3', '')**

**price = request.POST.get('s4', '')**

**quantity = request.POST.get('s5', '')**

**gst = request.POST.get('s6', '')**

**cgst = request.POST.get('s7', '')**

**sgst = request.POST.get('s8', '')**

**total = request.POST.get('s9', '')**

**discount = request.POST.get('s10', '')**

**Product.objects.filter(id=id).update(pname=pname, description=description, image1=image1,**

**price=price, quantity=quantity, gst=gst, cgst=cgst, sgst=sgst, total=total, discount=discount )**

**user\_items = Product.objects.all()**

**return render(request, 'productview.html', context={'user\_dict': user\_items})**

**def service(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**Product\_name = request.POST.get('m1', '')**

**Description = request.POST.get('m2', '')**

**Price = request.POST.get('m3', '')**

**Service\_men\_name = request.POST.get('m4', '')**

**Service\_men\_Mbno = request.POST.get('m5', '')**

**Service.objects.create(Product\_name=Product\_name, Description=Description, Price=Price,**

**Service\_men\_name=+Service\_men\_name, Service\_men\_Mbno=Service\_men\_Mbno)**

**return render(request, 'admin\_home.html', context=user\_dict)**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'Service.html', {'user\_dict': user\_dict, 'prod\_cat': prod\_cat, 'prod\_list': prod\_list})**

**def serviceview(request):**

**user\_dict = Service.objects.all()**

**return render(request, 'serviceview.html', {'user\_dict': user\_dict})**

**def serremove\_items(request):**

**if request.method == 'POST':**

**userdatas = Service.objects.all()**

**id = request.POST.get('id')**

**useritems = Service.objects.get(id=id)**

**useritems.delete()**

**user\_items = Service.objects.all()**

**return render(request,'serviceview.html',context={'user\_dict': user\_items})**

**def serupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = Service.objects.all()**

**id = request.POST.get('id')**

**useritems = Service.objects.filter(id=id).values()**

**return render(request, 'serviceedit.html', {'useritems': useritems})**

**def ser\_db(request):**

**if request.method == "POST":**

**userdatas = Service.objects.all()**

**id = request.POST.get('id')**

**useritems = Service.objects.filter(id=id).values()**

**Product\_name = request.POST.get('m1', '')**

**Description = request.POST.get('m2', '')**

**Price = request.POST.get('m3', '')**

**Service\_men\_name = request.POST.get('m4', '')**

**Service\_men\_Mbno = request.POST.get('m5', '')**

**Service.objects.filter(id=id).update(Product\_name=Product\_name, Description=Description, Price=Price, Service\_men\_name=Service\_men\_name, Service\_men\_Mbno=Service\_men\_Mbno)**

**user\_items = Service.objects.all()**

**return render(request, 'serviceview.html', context={'user\_dict': user\_items})**

**def mbscheme(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**Schemeno = request.POST.get('l1', '')**

**Customer\_name = request.POST.get('l2', '')**

**Address = request.POST.get('l3', '')**

**Email\_Id = request.POST.get('l4', '')**

**Mobileno = request.POST.get('l5', '')**

**Month = request.POST.get('l6', '')**

**Winner = request.POST.get('l7', '')**

**Coupon\_code = request.POST.get('l8', '')**

**MB\_scheme.objects.create(Schemeno=Schemeno, Customer\_name=Customer\_name, Address=Address, Email\_Id=Email\_Id, Mobileno=Mobileno, Month=Month, Winner=Winner, Coupon\_code=Coupon\_code)**

**return render(request, 'index.html', context=user\_dict)**

**return render(request,'mbscheme.html')**

**def mbschemeview(request):**

**user\_dict = MB\_scheme.objects.all()**

**return render(request, 'mbschemeview.html', {'user\_dict': user\_dict})**

**def mbsremove\_items(request):**

**if request.method == 'POST':**

**userdatas = MB\_scheme.objects.all()**

**id = request.POST.get('id')**

**useritems = MB\_scheme.objects.get(id=id)**

**useritems.delete()**

**user\_items = MB\_scheme.objects.all()**

**return render(request,'mbschemeview.html',context={'user\_dict': user\_items})**

**def mbsupdate\_items(request):**

**if request.method == 'POST':**

**userdatas = MB\_scheme.objects.all()**

**id = request.POST.get('id')**

**useritems = MB\_scheme.objects.filter(id=id).values()**

**return render(request, 'mbschemeedit.html', {'useritems': useritems})**

**def mbs\_db(request):**

**if request.method == "POST":**

**userdatas = MB\_scheme.objects.all()**

**id = request.POST.get('id')**

**useritems = MB\_scheme.objects.filter(id=id).values()**

**Schemeno = request.POST.get('l1', '')**

**Customer\_name = request.POST.get('l2', '')**

**Address = request.POST.get('l3', '')**

**Email\_Id = request.POST.get('l4', '')**

**Mobileno = request.POST.get('l5', '')**

**Month = request.POST.get('l6', '')**

**Winner = request.POST.get('l7', '')**

**Coupon\_code = request.POST.get('l8', '')**

**MB\_scheme.objects.filter(id=id).update(Schemeno=Schemeno, Customer\_name=Customer\_name, Address=Address, Email\_Id=Email\_Id, Mobileno=Mobileno, Month=Month, Winner=Winner, Coupon\_code=Coupon\_code)**

**user\_items = MB\_scheme.objects.all()**

**return render(request, 'mbschemeview.html', context={'user\_dict': user\_items})**

**def changepassword(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**uname = request.POST.get('t1','')**

**upass = request.POST.get('t2','')**

**newpass = request.POST.get('t3','')**

**confirmpass = request.POST.get('t4','')**

**ucheck = UserLogin.objects.filter(emailid=uname).values()**

**for a in ucheck:**

**u = a['emailid']**

**p = a['password']**

**if u == uname and upass == p:**

**if newpass == confirmpass:**

**UserLogin.objects.filter(emailid=uname).update(password=newpass)**

**else:**

**return render(request,'changepassword.html', context={'msg':'both must be same'})**

**else:**

**return render(request, 'changepassword.html', context={'msg':'invalid username or password'})**

**return render(request, 'changepassword.html')**

**def forgotpassword(request):**

**if request.method=="POST":**

**u=request.POST.get('a1')**

**if u=='priyanka@gmail.com':**

**return render(request,'index.html',{'msg':'12345'})**

**else:**

**return render(request,'index.html',{'msg':'invalid'})**

**return render(request,'forgotpassword.html')**

**def procustomerview(request):**

**user\_dict = Product.objects.all()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'product\_view\_customer.html', context={'prod\_cat':prod\_cat, 'prod\_list':prod\_list, 'user\_dict': user\_dict})**

**def porder(request):**

**emailid = request.session['uid']**

**today = ""**

**quantity = ""**

**unitprice = ""**

**pname = ""**

**if request.method == "POST":**

**pid = request.POST.get('pid')**

**quantity = request.POST.get('c3')**

**today = datetime.today()**

**uuprice = Product.objects.filter(id=pid).values()**

**for p in uuprice:**

**unitprice = p['price']**

**print(unitprice)**

**pname = p['pname']**

**return render(request, 'order.html', {'use': today, 'qtty': quantity, 'ss': unitprice, 'user\_dict': emailid, 'pp': pname})**

**def Orderview(request):**

**user\_dict = order\_details.objects.all()**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'orderview.html', {'prod\_cat': prod\_cat, 'prod\_list':prod\_list, 'user\_dict': user\_dict})**

**def odelete(request):**

**if request.method == 'POST':**

**userdatas = order\_details.objects.all()**

**id = request.POST.get('id')**

**useritems = order\_details.objects.get(id=id)**

**useritems.delete()**

**user\_items = order\_details.objects.all()**

**return render(request,'orderview.html',context={'user\_dict': user\_items})**

**def order\_next(request):**

**if request.method=="POST":**

**order\_date = request.POST.get('c2')**

**quantity = request.POST.get('c3')**

**unitprice = request.POST.get('c4')**

**Total = request.POST.get('c5')**

**today = datetime.today()**

**uid = request.POST.get('c6', '')**

**items = request.POST.get('c7', '')**

**pname = request.POST.get('pname')**

**order\_details.objects.create(order\_date=order\_date, quantity=quantity, unitprice=unitprice, Total=Total,uid=uid,items=items)**

**payment = order\_details.objects.latest('id')**

**return render(request, 'Payment.html', {'pay\_amount': payment, 'dd':today})**

**def cart(request):**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**pid = request.POST.get('pid')**

**pname = request.POST.get('pname')**

**unitprice = request.POST.get('unitprice')**

**today = datetime.today()**

**order\_details.objects.create(pid=pid, pname=pname, unitprice=unitprice)**

**return render(request, 'customers.html', context=user\_dict)**

**return render(request, 'Cart.html')**

**def billing(request):**

**items = ""**

**today = ""**

**total = ""**

**customersid = ""**

**user\_dict = {'msg': 'one record inserted successfully'}**

**if request.method == "POST":**

**id = request.POST.get('id')**

**orderid = request.POST.get('f2')**

**customersid = request.POST.get('f3')**

**bill\_date = request.POST.get('f4')**

**today = datetime.today()**

**items = request.POST.get('f5')**

**total = request.POST.get('f6')**

**order\_dict = order\_details.objects.filter(id=id).values()**

**for s in order\_dict:**

**customersid = s['uid']**

**print(customersid)**

**items = s['items']**

**print(items)**

**total = s['unitprice']**

**print('total')**

**gst = request.POST.get('f7')**

**tax\_total = request.POST.get('f8')**

**shipping\_charge = request.POST.get('f9')**

**grandtotal = request.POST.get('f10')**

**Billing.objects.create(orderid=orderid, customersid=customersid, items=items, bill\_date=bill\_date, total=total, gst=gst, tax\_total=tax\_total, shipping\_charge=shipping\_charge, grandtotal=grandtotal)**

**return render(request, 'Bill.html', {'oid': id, 'tt': today, 'cc': customersid, 'zz': items, 'yy': total})**

**def billview(request):**

**user\_dict = Billing.objects.all()**

**return render(request, 'billingview.html', {'user\_dict': user\_dict})**

**def bill\_delete(request):**

**if request.method == 'POST':**

**userdatas = Billing.objects.all()**

**id = request.POST.get('id')**

**useritems = Billing.objects.get(id=id)**

**useritems.delete()**

**user\_items = Billing.objects.all()**

**return render(request,'billingview.html',context={'user\_dict': user\_items})**

**def payment(request):**

**today = ""**

**user\_dict = ''**

**if request.method == "POST":**

**Paymenttype = request.POST.get('f2','')**

**BankName = request.POST.get('f3','')**

**Couponno = request.POST.get('f4','')**

**Amount = request.POST.get('f5')**

**Paymentdate = request.POST.get('f6')**

**today = datetime.today()**

**print(Amount)**

**Payment.objects.create(Paymenttype=Paymenttype, BankName=BankName, Couponno=Couponno, Amount=Amount, Paymentdate=Paymentdate)**

**amt = int(Amount)**

**return render(request, 'payment2.html',{'amount': amt})**

**return render(request, 'Payment.html', {'pid': id, 'dd': today})**

**def paymentview(request):**

**user\_dict = Payment.objects.all()**

**return render(request, 'payview.html', {'user\_dict': user\_dict})**

**def paymentremove(request):**

**useritems = ''**

**if request.method == 'POST':**

**userdatas = Payment.objects.all()**

**id = request.POST.get('id')**

**useritems = Payment.objects.get(id=id)**

**useritems.delete()**

**useritems = Payment.objects.all()**

**return render(request,'payview.html', context={'user\_dict': useritems})**

**def faq(request):**

**user\_dict = {'msg': 'sent'}**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'faqs.html', context={'user\_dict': user\_dict, 'prod\_cat':prod\_cat, 'prod\_list':prod\_list})**

**def catwise(request):**

**return render(request,'catwise\_view.html')**

**def prodwise(request):**

**return render(request, 'prodwise\_view.html')**

**def feedback(request):**

**user\_dict = {'msg': ''}**

**if request.method == "POST":**

**userid = request.POST.get('e2', '')**

**servicefeedback = request.POST.get('e3', '')**

**comments = request.POST.get('e4', '')**

**FeedBackForm.objects.create(userid=userid, servicefeedback=servicefeedback, comments=comments)**

**return render(request, 'index.html', context=user\_dict)**

**prod\_cat = ProCategory.objects.all()**

**prod\_list = Product.objects.all()**

**return render(request, 'Feedback.html', context={'user\_dict': user\_dict, 'prod\_cat':prod\_cat, 'prod\_list':prod\_list})**

**def fbackview(request):**

**user\_dict = FeedBackForm.objects.all()**

**return render(request, 'feedbackview.html', {'user\_dict': user\_dict})**

**def servicecustomer(request):**

**user\_dict = Service.objects.all()**

**return render(request, 'servicecustomerview.html', context={'user\_dict':user\_dict})**

**URLS.py**

**from django.conf.urls import url**

**from . import views**

**urlpatterns=[**

**url('^$', views.index, name='index'),**

**url('^admins/$', views.admins, name='admins'),**

**url('^customers/$', views.customers, name='customers'),**

**#url('^sample/$',views.sample, name='sample'),**

**url('^signup/$', views.signup, name='signup'),**

**url('^sampleview/$', views.sampleview, name='sampleview'),**

**url('^remove\_items/$', views.remove\_items, name='remove\_items'),**

**url('^update\_items/$', views.update\_items, name='update\_items'),**

**url('^reg\_db/$', views.reg\_db, name='reg\_db'),**

**url('^changepassword/$',views.changepassword, name='changepassword'),**

**url('^forgotpassword/$',views.forgotpassword, name='forgotpassword'),**

**url('^signin/$', views.signin, name='signin'),**

**#url('^connection/$', views.signin, name='signin'),**

**url('^Category/$',views.Category, name='Category'),**

**url('^categoryview/$',views.categoryview, name='categoryview'),**

**url('^catremove\_items/$',views.catremove\_items, name='catremove\_items'),**

**url('^catupdate\_items/$',views.catupdate\_items, name='catupdate\_items'),**

**url('^cat\_db/$',views.cat\_db, name='cat\_db'),**

**url('^purchase/$',views.purchase, name='purchase'),**

**url('^purchaseview/$',views.purchaseview, name='purchaseview'),**

**url('^purremove\_items/$',views.purremove\_items, name='purremove\_items'),**

**url('^purupdate\_items/$',views.purupdate\_items, name='purupdate\_items'),**

**url('^pur\_db/$', views.pur\_db, name='pur\_db'),**

**url('^supplier/$',views.supplier, name='supplier'),**

**url('^supplierview/$',views.supplierview, name='supplierview'),**

**url('^supremove\_items/$',views.supremove\_items, name='supremove\_items'),**

**url('^supupdate\_items/$',views.supupdate\_items, name='supupdate\_items'),**

**url('^sup\_db/$',views.sup\_db, name='sup\_db'),**

**url('^stock/$',views.stock, name='stock'),**

**url('^stockview/$',views.stockview, name='stockview'),**

**url('^stkremove\_items/$',views.stkremove\_items, name='stkremove\_items'),**

**url('^stkupdate\_items/$',views.stkupdate\_items, name='stkupdate\_items'),**

**url('^stk\_db/$',views.stk\_db, name='stk\_db'),**

**url('^exchange/$',views.exchange, name='exchange'),**

**url('^exchangeview/$',views.exchangeview, name='exchangeview'),**

**url('^exchgremove\_items/$',views.exchgremove\_items, name='exchgremove\_items'),**

**url('^exchgupdate\_items/$',views.exchgupdate\_items, name='exchgupdate\_items'),**

**url('^ex\_db/$',views.ex\_db, name='ex\_db'),**

**url('^product/$',views.product, name='product'),**

**url('^productview/$',views.productview, name='productview'),**

**url('^proremove\_items/$',views.proremove\_items, name='proremove\_items'),**

**url('^proupdate\_items/$',views.proupdate\_items, name='proupdate\_items'),**

**url('^pro\_db/$',views.pro\_db, name='pro\_db'),**

**url('^service/$', views.service, name='service'),**

**url('^serviceview/$', views.serviceview, name='serviceview'),**

**url('^serremove\_items/$', views.serremove\_items, name='serremove\_items'),**

**url('^serupdate\_items/$', views.serupdate\_items, name='serupdate\_items'),**

**url('^ser\_db/$', views.ser\_db, name='ser\_db'),**

**url('^servicecustomer/$', views.servicecustomer, name='servicecustomer'),**

**url('^mbscheme/$', views.mbscheme, name='mbscheme'),**

**url('^mbschemeview/$', views.mbschemeview, name='mbschemeview'),**

**url('^mbsremove\_items/$', views.mbsremove\_items, name='mbsremove\_items'),**

**url('^mbsupdate\_items/$', views.mbsupdate\_items, name='mbsupdate\_items'),**

**url('^mbs\_db/$', views.mbs\_db, name='mbs\_db'),**

**url('^procustomerview/$',views.procustomerview, name='procustomerview'),**

**url('^porder/$', views.porder, name='porder'),**

**url('^Orderview/$', views.Orderview, name='Orderview'),**

**url('^odelete/$', views.odelete, name='odelete'),**

**url('^billing/$', views.billing, name='billing'),**

**url('^billview/$', views.billview, name='billview'),**

**url('^bill\_delete/$', views.bill\_delete, name='bill\_delete'),**

**url('^payment/$', views.payment, name='payment'),**

**url('^paymentview/$', views.paymentview, name='paymentview'),**

**url('^paymentremove/$', views.paymentremove, name='paymentremove'),**

**url('^cart/$', views.cart, name='cart'),**

**url('^faq/$', views.faq, name='faq'),**

**url('^feedback/$', views.feedback, name='feedback'),**

**url('^fbackview/$', views.fbackview, name='fbackview'),**

**url('^catwise/$', views.catwise, name='catwise'),**

**url('^prodwise/$', views.prodwise, name='prodwise'),**

**url('^order\_next/$', views.order\_next, name='order\_next'),**

**url('^send\_mail/$', views.send\_mail, name='send\_mail'),**

**url('^customers\_next/$', views.customers\_next, name='customers\_next'),**

**#url('^thankyou/$', views.thankyou, name='thankyou'),**

**]**

**10. Testing**

**10.1 Scope of Testing**

The code is tested at various levels in software testing. Unit system and user acceptance testing are often performed. This is grey area as many different opinions. Exist as to what the stage of testing is and how much if any iteration occurs. Iteration is generally part of the waterfall model, but usually occurs at this stage.

Software testing is the execution of program to find its faults. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested and on the functional externals, that is conducting test to uncover errors and ensure that defined inputs will produce actual results agreed with required with required results. The following test strategies were adopted to test the system.

**10.2 Testing Objective**

The system is tested with variety of inputs. The System is tested for accuracy and correctness of the results obtained. Finally the system is tested for inter-operability.

**Test Approaches:**

1. **Black Box Testing:**

* This method focuses on the functional requirements of the software.
* Thistesting enables to derive set input conditions that will fully exercise all functional requirements of the program.
* Black Box testing attempts to find errors in the following category.
* Incorrect or missing functions.
* Interface errors.
* Performance errors.
* Initialization and Termination errors.

1. **White Box Testing:**

* This is performed early in the testing process, while Black Box testing is applied during the last stage of testing. In this test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases.
* It has been used to generate the test case in the following test cases:
* Guarantee that all independent paths have been executed.
* Execute all logical decisions from their True and False side.
* Execute all loops at their boundaries and within their operational bounds.

**10.3 About Testing**

**10.4 Software Quality Assurances.**

**10.5 Testing Strategies**

There are two general strategies for testing software. These are as follows:

* **Code Testing:** This examines the logic of the program. To follow this test, cases are developed such that every path of program is tested.
* **Specification Testing:** Specification Testing examines the specification starting what the program should do and how it should perform under various conditions. Then test cases are developed for each condition and combinations of conditions and to be submitted for processing.

**The stages in testing process are:**

* **Unit Testing:**Unit testing is done on individual modules as they are completed and become executable. It is confined only to the designer’s requirements.
* **Integration Testing:**Integration testing ensures that software and subsystems work together as a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.
* **System Testing:**System testing involves in-house testing of the entire system before delivery to the user. Its aim is to satisfy the user. The system meets all requirements of the client’s specifications.
* **Performance Testing:**It was a good idea to do our stress testing early on, because it gave us time to fix some of the unexpected deadlocks and stability problems that only occurred when components were exposed to very high transaction volumes.
* **Validation Testing:**The system has been tested and implemented successfully and thus ensured that all requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed.
* **Acceptance Testing:**It is a pre-delivery testing in which entire system is tested at client’s site on real world data to find errors.

**12. Bibliography**

**Web sites:**

**W3schools.com**

**django.com**