## CHAPTER 1

**INTRODUCTION**

**1.1 OVERVIEW OF THEPROJECT**

Here we shared bakery Shop Management System Project Report which is developed using eclipse. This is software application and is used maintain control most of the activities happening in big shop. This maintains and controls the stock details and does billing and generates various reports.

The bakery Shop Management System will allow more than one shop owner, to sell various products less than one roof. “Details of products in the stock with product code, product name, quantity” bakery Shop Management System is inventory software that designed specially to meet the requirements of small and medium sized enterprises, such as bakery stores whole sale or retail. It is easy to create invoice, do inventory control like invoice management, stock balance management, goods item management, goods category management, staff sales records management and staff permission management, backup and restore stock by Inventory’s user friendly and graphical interface and functionalities. And Inventory supports full customizable Company info, tax code and value, invoice number etc.

## CHAPTER 2

**SYSTEM ANALYSIS**

**EXISTINGSYSTEM**

Here the shop has been handled by manually, so it’s very hard to managing the bakery shop. Each and every purchase and sales can be handling into the paper and managing it. Here there’s an no records have been provided to the owner or manager. These kinds of issues may be happened in the existing system

**DISADVANTAGES**

* Very hard to find the stock details
* No records will be provided by the system

## PROPOSEDSYSTEM

Here in this system provide lot of features, like purchase, sales and stock details. Once the admin registered these records can be followed by the system. Here we can provide the details in less minute time.

**ADVANTAGES**

* We can easily identify the product details in a systematic
* Generate billing and stock details in a system way

## FEASIBILITYSTUDY

The feasibility of the system is analyzed in this phase and business proposal is put forth with general plan for the project and cost estimates. During the system analysis of the project, the feasibility study of proposed system is to be carried out. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in feasibility analysis are

* + - Technical Feasibility
    - Economic Feasibility
    - Operational Feasibility

## Technical Feasibility

Technical feasibility assesses the current resources (such as hardware and software) technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, whether the certain current resources and technology can be upgraded or added in the software to accomplish specified user requirements.

The technical requirements of the application are simple and basic. Python is used for the developers of the application and the framework is largely used by many, thus there will be enough support for future enhancements. The framework is stable and the support from the developers is constantly updated. The devices which have internet connectivity are enough for the application.

## Economic Feasibility

Economic feasibility determines whether the required software is capable of generating financial gains for an organization. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on.

The cost of application development is very less and the cost of implementation is also less. It can be developed with the system with minimum requirements and can also be operated with the system with some basic requirements that are available the existing systems. For this, it is essential to consider expenses made on purchases and activities required to carry out software development.

## Operational Feasibility

Operational feasibility assesses the extent to which the required software performs a series of steps to solve user requirements. This feasibility is dependent on developer and involves visualizing whether the software will operate after it has been developed and be operative.

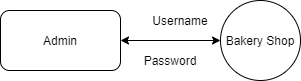
The application is developed based on the user requirements and is developed on the priority of the user requirements such as an integrated service and reviewing platform.

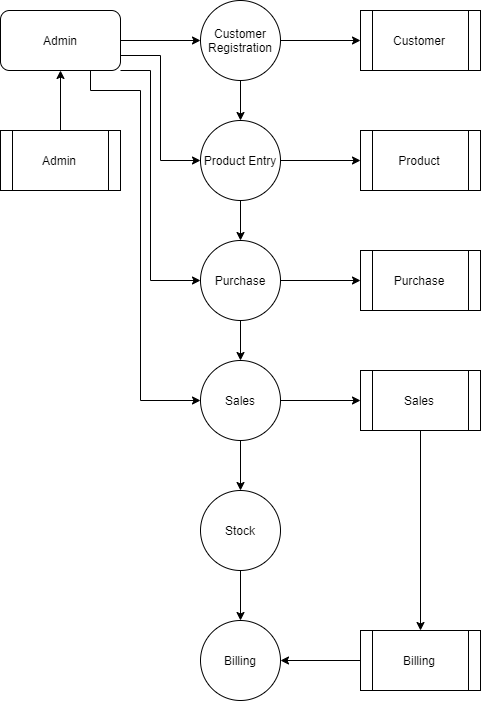
## PROBLEMANALYSIS

The orders from public are submitted through email, letter or form to the corresponding authority. The registered orders are forwarded to corresponding authority and report about the orders is submitted. The status of orders addressed is not recorded and maintained properly. The maps are not implemented. So, the customer does not know live location of the package. There is need for the software which receives the orders through online, forwarded to respective peoples for rectification and post the status of orders.

## CONTEXT AND DATA FLOWDIAGRAM

A data-flow diagram (DFD)is a way of representing a flow of a data of a process or system. The DFD also provides information about the outputs and inputs of each entity and process itself. A data-flow diagram is a part of structured-analysis modelling tools.





**SYSTEMCONFIGURATION**

### Hardware Requirements

Processor : P 4 700 GHz

RAM Capacity : 4GB

Hard Disk : 180GB

### Software Requirements

Operating System : Windows 8,10

Front End : JAVA

Back End : SQL

### Software Descriptions

### Java

Java is a high-level programming language developed by Sun Microsystems. It was originally designed for developing programs for set-top boxes and handheld devices, but later became a popular choice for creating web applications.

The Java syntax is similar to C++, but is strictly an object-oriented programming language. For example, most Java programs contain classes, which are used to define objects, and methods, which are assigned to individual classes. Java is also known for being stricter than C++, meaning variables and functions must be explicitly defined. This means Java source code may produce errors or "exceptions" more easily than other languages, but it also limits other types of errors that may be caused by undefined variables or unassigned types.

Unlike Windows executables (.EXE files) or Macintosh applications (.APP files), Java programs are not run directly by the operating system. Instead, Java programs are interpreted by the Java Virtual Machine, or JVM, which runs on multiple platforms. This means all Java programs are multiplatform and can run on different platforms, including Macintosh, Windows, and Unix computers. However, the JVM must be installed for Java applications or applets to run at all. Fortunately, the JVM is included as part of the Java Runtime Environment (JRE),

### MySQL

MySQL is the popular Open-Source Relational SQL Database Management System. MySQL is being used for developing various web-based software applications. The MySQL development project has made its [source code](https://en.wikipedia.org/wiki/Source_code) available under the terms of the [General Public](https://en.wikipedia.org/wiki/GNU_General_Public_License) [License.](https://en.wikipedia.org/wiki/GNU_General_Public_License) It is used to store the information.

MySQL was owned and sponsored by the single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available, and offer additional functionality.

## CHAPTER 3

**SYSTEM DESIGN**

**3.1 INPUT DESIGN**

Input Design is the process of converting a user-oriented description of the input into a computer-based system. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. Input Design is the first phase in the system design. Input designing is to converting the user-oriented information to the computer- oriented form. The input data items are grouped and analyzed to find out whether the proposed system can be developed from the user input. The system is developed using various processes screens formats.

The main objective of input design is to

* + - It should serve specific purpose effectively such as storing, recording, and retrieving the information.
    - It ensures proper completion with accuracy.
    - It should be easy to fill and straightforward.
    - It should focus on user’s attention, consistency and simplicity.

### DATABASE DESIGN

The most important consideration in designing the database is how the information will be used. The main objective of designing a database is Data Integration, Data Integrity and Data Independence.

### Data Integration

In a database, information from several files is coordinated, accessed and operated upon as through it is in a single file. Logically, the information is centralized, physically, the data may be located on different devices, connected through data communication facilities.

### Data Integrity

Data integrity means storing all data in one place only and how each application access it. This approach results in more consistent information, one update being sufficient to achieve a new record status for all applications. This leads to less data redundancy, that is data items need not be duplicated. A reduction in the direct access storage requirement.

### Data Independence

Dataindependenceistheinsulationofapplicationprogramsfromchangingaspects of physical data organization. This objective seeks to allow changes in the content and organizationofphysicaldatawithoutreprogrammingofapplicationandallowmodifications to application programs without reorganizing the physical data.

### 3.2.1 TABLE DESIGN

The table needed for each module were designed and the specification of each and every column was given based on the records and details collected during record specification of the system study.

**TABLE NAME: admin**

|  |  |  |  |
| --- | --- | --- | --- |
| **FIELD** | **DATA TYPE** | **SIZE** | **CONSTRAINT** |
| Id | Int | 10 | Primary key |
| Username | Varchar | 30 | Not null |
| Password | Varchar | 30 | Not null |

**TABLE NAME: Customer**

|  |  |  |  |
| --- | --- | --- | --- |
| **FIELD** | **DATA TYPE** | **SIZE** | **CONSTRAINT** |
| Id | Int | 10 | Primary key |
| Name | Varchar | 30 | Not null |
| Address | Varchar | 30 | Not null |
| Contact no | Varchar | 10 | Not null |

**TABLE NAME: Product**

|  |  |  |  |
| --- | --- | --- | --- |
| **FIELD** | **DATA TYPE** | **SIZE** | **CONSTRAINT** |
| Id | Int | 10 | Primary key |
| Product name | Varchar | 30 | Not null |
| Price | Varchar | 10 | Not null |
| Brand | Varchar | 30 | Not null |

**TABLE NAME: Purchase**

|  |  |  |  |
| --- | --- | --- | --- |
| **FIELD** | **DATA TYPE** | **SIZE** | **CONSTRAINT** |
| Id | Int | 10 | Primary key |
| Product id | Int | 10 | Foreign key |
| Total count | Int | 10 | Not null |

**TABLE NAME: Sales**

|  |  |  |  |
| --- | --- | --- | --- |
| **FIELD** | **DATA TYPE** | **SIZE** | **CONSTRAINT** |
| Id | Int | 10 | Primary key |
| Product id | Int | 10 | Foreign key |
| Customer id | Int | 10 | Foreign key |
| Total count | Int | 10 | Not null |

**MODULEDESCRIPTION**

The main module in this project are listed below

* + - **Customer Registration**
    - **Product Registration**
    - **Purchase**
    - **Sales**
    - **Stock**
    - **Billing**

1. **Customer Registration**

This module will be collecting the all the information about the customer, when customer visit or purchase any product admin has collect the customer details and register here,

1. **Product Registration**

This module admin registers the product details, what are the products are available into the bakery shop, once it’s added it will be stored into the product table.

1. **Purchase**

When owner purchasing the product for a shop, it should manage by this module. each and every purchase detail are should be managing here, then only we can maintain the stock details.

1. **Sales**

This module is act as an when customer purchasing the any product admin collect the purchase details and sales it. Then it will show the total cost rupees in a sales screen.

1. **Stock**

This module helps to shown the what are the stocks are available in the shop, we can easily identify the overall stock in the single screen. Also, can able to find the what are the stocks will be close find then can purchase the order.

1. **Billing**

This module has to show the billing details for the bakery shop. We can see the billing detail as a date wise and customer wise. Which can provide the report to the shop

## CHAPTER 4

## SYSTEM TESTING

Testingisanintegralpartofanysystemdevelopmentlifecycle.Insufficient and untested applications may tend to crash and the result is loss of economic and manpower investment besides user's dissatisfaction and downfall of reputation. Software testing can be looked upon as one among many processes, an organization performs, and that provides the lost opportunity to correct any flaws in the developed system. Software testing includes selecting test data that have more probability of giving errors.

The first step in system testing is to develop a plan that tests all aspects of the system. Completeness, correctness, reliability and maintainability of the software aretobetestedforthebestqualityassurancethatthesystemmeetsthespecificationand requirements for its intended use and performance. System testing is the most useful practical process of executing a program with the implicit intention of finding errors that make the program fails. System testing is done in three phases.

* + - * Unit Testing
      * Integration Testing
      * Validation Testing

### UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software the module. Using the detailed design and the process specification testing is done to registration by the user with in the boundary of the Login module. The login form receives the username and password details and validates the value with the database. If valid, the home page is displayed.

### INTEGRATION TESTING

Integration Testing is the process of this activity can be considered as testing the design and hence module interaction. The primary objective of integration testing is to discover errors in the interfaces between the components. Login form and registration form are integrated and tested together. If the user is newly registered, the received details will be stored in the registration table. While logging in, the application will check for valid user name and password in the registration table and if valid the user is prompted for submitting complaints.

### VALIDATION TESTING

Validation are independent procedures that are used together for checking that a product, service, or system meets [requirements](https://en.wikipedia.org/wiki/Requirement) and [specifications](https://en.wikipedia.org/wiki/Specification_(technical_standard)) and that it fulfills its in purpose the actual result from the expected result for the complaint process. Select the complaint category of the complaint by user. The input given to various forms fields are validated effectively. Each module is tested independently. It is tested that the complaint module fields receive the correct input for the necessary details such as complaint category, complaint id, reference name, complaint description, email for further process.

## CHAPTER 5

* 1. **CONCLUSION**

The purpose of this project as an maintaining the customer details as well as shop purchase, sales and stock details. Which can be maintain any where at any situations. User can handle very user friendly to run this application. Every one can manually understand and work the respective app.

## APPENDICES APPENDIX I SAMPLE CODE

package com.example.demo.controller;

import java.util.List;

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

import org.springframework.http.ResponseEntity;

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

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

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

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

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

import com.example.demo.response.GetBillingResponse;

import com.example.demo.response.GetCustomerResponse;

import com.example.demo.response.GetProductResponse;

import com.example.demo.response.GetStockResponse;

import com.example.demo.service.ApiService;

@RestController

@RequestMapping(value = { "/api" })

public class ApiController {

@Autowired

ApiService service;

@GetMapping("/login/{username}/{password}")

public Boolean login(@PathVariable String username,@PathVariable String password) {

return service.login(username,password);

}

@PostMapping("/add\_customer/{name}/{mobile}/{alternate}/{address}/{gender}")

public String add\_customer(@PathVariable String name,

@PathVariable String mobile,

@PathVariable String alternate,

@PathVariable String address,

@PathVariable String gender) {

service.add\_customer(name,mobile,alternate,address,gender);

return "Customer Saved Sucessfully";

}

@GetMapping("/get\_customer")

public ResponseEntity<List<GetCustomerResponse>> get\_customer() {

return ResponseEntity.ok().body(service.get\_customer());

}

@PostMapping("/add\_product/{company}/{model}/{price}")

public String add\_product(@PathVariable String company,

@PathVariable String model,

@PathVariable Integer price) {

service.add\_product(company,model,price);

return "Product Saved Sucessfully";

}

@GetMapping("/get\_product")

public ResponseEntity<List<GetProductResponse>> get\_product() {

return ResponseEntity.ok().body(service.get\_product());

}

@PostMapping("/add\_purchase/{product\_id}/{quantity}/{details}")

public String add\_purchase(@PathVariable Integer product\_id,

@PathVariable Integer quantity,@PathVariable String details) {

service.add\_purchase(product\_id,quantity,details);

return "Purchase Saved Sucessfully";

}

@PostMapping("/add\_sales/{customer\_id}/{product\_id}/{quantity}")

public String add\_sales(@PathVariable Integer customer\_id,@PathVariable Integer product\_id,

@PathVariable Integer quantity) {

service.add\_sales(customer\_id,product\_id,quantity);

return "Sales Saved Sucessfully";

}

@GetMapping("/get\_customer/{mobile}")

public Integer get\_mobile(@PathVariable String mobile) {

return service.get\_mobile(mobile);

}

@GetMapping("/get\_stock")

public ResponseEntity<List<GetStockResponse>> get\_stock() {

return ResponseEntity.ok().body(service.get\_stock());

}

@GetMapping("/get\_billing")

public ResponseEntity<List<GetBillingResponse>> get\_billing() {

return ResponseEntity.ok().body(service.get\_billing());

}

}

package com.example.demo.dao;

import java.math.BigInteger;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.query.NativeQuery;

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

import org.springframework.stereotype.Repository;

@Repository

public class ApiDao {

@Autowired

SessionFactory sf;

public void add\_customer(String name, String mobile, String alternate,String address, String gender) {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "INSERT INTO `customer` (`id`, `name`, `mobile`,`alternate`, `address`, `gender`) VALUES (NULL, '"

+ name + "', '" + mobile + "', '" + alternate + "', '" + address + "', '" + gender + "');";

session.createSQLQuery(sql).executeUpdate();

}

public List<Object[]> get\_customer() {

Session session = sf.getCurrentSession();

String sql = "Select \* from customer";

NativeQuery nq = session.createNativeQuery(sql);

return nq.list();

}

public void add\_product(String company, String model, Integer price) {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "INSERT INTO `product` (`id`, `company`, `model`, `price`) VALUES (NULL, '" + company + "', '"

+ model + "', '" + price + "');";

session.createSQLQuery(sql).executeUpdate();

}

public List<Object[]> get\_product() {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "Select \* from product";

NativeQuery nq = session.createNativeQuery(sql);

return nq.list();

}

public void add\_purchase(Integer product\_id, Integer quantity, String details) {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "INSERT INTO `purchase` (`id`, `product\_id`, `quantity`, `details`, `date`) VALUES (NULL, '" + product\_id

+ "', '" + quantity + "', '" + details + "', current\_timestamp());";

session.createSQLQuery(sql).executeUpdate();

}

public void add\_sales(Integer customer\_id, Integer product\_id, Integer quantity) {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "INSERT INTO `sales` (`id`, `customer\_id`, `product\_id`, `quantity`) VALUES (NULL, '" + customer\_id

+ "', '" + product\_id + "', '" + quantity + "');";

session.createSQLQuery(sql).executeUpdate();

}

public List<Object[]> get\_stock() {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "select p.company,p.model,COALESCE(sum(pqty),0) - COALESCE(sum(sqty),0) qty from product p \r\n"

+ "LEFT JOIN (select product\_id,COALESCE(SUM(quantity),0) pqty from purchase GROUP by product\_id) as a on a.product\_id = p.id\r\n"

+ "LEFT JOIN (select product\_id,COALESCE(SUM(quantity),0) sqty from sales GROUP by product\_id) as b on b.product\_id = p.id\r\n"

+ "GROUP BY p.company,p.model";

System.out.println(sql);

NativeQuery nq = session.createNativeQuery(sql);

return nq.list();

}

public Integer get\_mobile(String mobile) {

Session session = sf.getCurrentSession();

String sql = "Select id,name from customer where mobile='" + mobile + "'";

NativeQuery nq = session.createNativeQuery(sql);

List<Object[]> list = nq.getResultList();

if (list.size() != 0) {

return (Integer) list.get(0)[0];

} else {

return null;

}

}

public List<Object[]> get\_billing() {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "select customer.name,customer.mobile,product.company,product.model,sales.quantity,product.price from sales LEFT JOIN customer on(customer.id=sales.customer\_id) LEFT JOIN product on(product.id=sales.product\_id) ";

NativeQuery nq = session.createNativeQuery(sql);

return nq.list();

}

public Boolean login(String username, String password) {

// TODO Auto-generated method stub

Session session = sf.getCurrentSession();

String sql = "select \* from admin where username='"+username+"' and password='"+password+"'";;

NativeQuery nq = session.createNativeQuery(sql);

if (nq.list().size() != 0) {

return true;

} else {

return false;

}

}

}package com.example.demo.configuration;

import java.util.Properties;

import javax.sql.DataSource;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.orm.hibernate5.HibernateTransactionManager;

import org.springframework.orm.hibernate5.LocalSessionFactoryBean;

import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration

@EnableTransactionManagement

public class HibernateConfiguration {

@Value("${db.driver}")

private String DB\_DRIVER;

@Value("${db.password}")

private String DB\_PASSWORD;

@Value("${db.url}")

private String DB\_URL;

@Value("${db.username}")

private String DB\_USERNAME;

@Value("${hibernate.dialect}")

private String HIBERNATE\_DIALECT;

@Value("${hibernate.show\_sql}")

private String HIBERNATE\_SHOW\_SQL;

@Value("${hibernate.hbm2ddl.auto}")

private String HIBERNATE\_HBM2DDL\_AUTO;

@Value("${entitymanager.packagesToScan}")

private String ENTITYMANAGER\_PACKAGES\_TO\_SCAN;

@Bean

public LocalSessionFactoryBean sessionFactory() {

LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();

sessionFactory.setDataSource(dataSource());

sessionFactory.setPackagesToScan(ENTITYMANAGER\_PACKAGES\_TO\_SCAN);

Properties hibernateProperties = new Properties();

hibernateProperties.put("hibernate.dialect", HIBERNATE\_DIALECT);

hibernateProperties.put("hibernate.show\_sql", HIBERNATE\_SHOW\_SQL);

hibernateProperties.put("hibernate.hbm2ddl.auto", HIBERNATE\_HBM2DDL\_AUTO);

sessionFactory.setHibernateProperties(hibernateProperties);

return sessionFactory;

}

@Bean

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(DB\_DRIVER);

dataSource.setUrl(DB\_URL);

dataSource.setUsername(DB\_USERNAME);

dataSource.setPassword(DB\_PASSWORD);

return dataSource;

}

@Bean

public HibernateTransactionManager transactionManager() {

HibernateTransactionManager txManager = new HibernateTransactionManager();

txManager.setSessionFactory(sessionFactory().getObject());

return txManager;

}

}package com.example.demo.configuration;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration

@EnableWebMvc

public class WebConfig extends WebMvcConfigurerAdapter {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*");

}

}package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration;

@SpringBootApplication

@EnableAutoConfiguration(exclude = HibernateJpaAutoConfiguration.class)

public class SampleApplication {

public static void main(String[] args) {

SpringApplication.run(SampleApplication.class, args);

}

}package com.example.demo.service;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.List;

import javax.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import com.example.demo.dao.ApiDao;

import com.example.demo.response.GetBillingResponse;

import com.example.demo.response.GetCustomerResponse;

import com.example.demo.response.GetProductResponse;

import com.example.demo.response.GetStockResponse;

@Service

@Transactional

public class ApiService {

@Autowired

ApiDao dao;

public void add\_customer(String name, String mobile,String alternate, String address, String gender ) {

// TODO Auto-generated method stub

dao.add\_customer(name,mobile,alternate,address,gender);

}

public List<GetCustomerResponse> get\_customer() {

// TODO Auto-generated method stub

List<Object[]> result =dao.get\_customer();

List<GetCustomerResponse> response = new ArrayList<GetCustomerResponse>();

for(int i=0;i<result.size();i++) {

Object[] row = result.get(i);

GetCustomerResponse obj = new GetCustomerResponse();

obj.setId((Integer)row[0]);

obj.setName((String)row[1]);

obj.setMobile((String)row[2]);

obj.setAddress((String)row[4]);

obj.setGender((String)row[5]);

obj.setAlternate((String)row[3]);

response.add(obj);

}

return response;

}

public void add\_product(String company, String model, Integer price) {

// TODO Auto-generated method stub

dao.add\_product(company,model,price);

}

public List<GetProductResponse> get\_product() {

// TODO Auto-generated method stub

List<Object[]> result =dao.get\_product();

List<GetProductResponse> response = new ArrayList<GetProductResponse>();

for(int i=0;i<result.size();i++) {

Object[] row = result.get(i);

GetProductResponse obj = new GetProductResponse();

obj.setId((Integer)row[0]);

obj.setCompany((String)row[1]);

obj.setModel((String)row[2]);

obj.setPrice((Integer)row[3]);

response.add(obj);

}

return response;

}

public void add\_purchase(Integer product\_id, Integer quantity, String details) {

// TODO Auto-generated method stub

dao.add\_purchase(product\_id,quantity,details);

}

public void add\_sales(Integer customer\_id, Integer product\_id, Integer quantity) {

// TODO Auto-generated method stub

dao.add\_sales(customer\_id,product\_id,quantity);

}

public List<GetStockResponse> get\_stock() {

List<Object[]> result =dao.get\_stock();

List<GetStockResponse> response = new ArrayList<GetStockResponse>();

for(int i=0;i<result.size();i++) {

Object[] row = result.get(i);

GetStockResponse obj = new GetStockResponse();

obj.setCompany\_name((String)row[0]);

obj.setQuantity((BigDecimal)row[2]);

obj.setDetails((String)row[1]);

response.add(obj);

}

return response;

}

public Integer get\_mobile(String mobile) {

// TODO Auto-generated method stub

return dao.get\_mobile(mobile);

}

public List<GetBillingResponse> get\_billing() {

List<Object[]> result =dao.get\_billing();

List<GetBillingResponse> response = new ArrayList<GetBillingResponse>();

for(int i=0;i<result.size();i++) {

Object[] row = result.get(i);

GetBillingResponse obj = new GetBillingResponse();

obj.setCustomer\_name((String)row[0]);

obj.setMobile((String)row[1]);

obj.setCompany((String)row[2]);

obj.setModel((String)row[3]);

obj.setQuantity((Integer)row[4]);

obj.setPrice((Integer)row[5]);

response.add(obj);

}

return response;

}

public Boolean login(String username, String password) {

return dao.login(username,password);

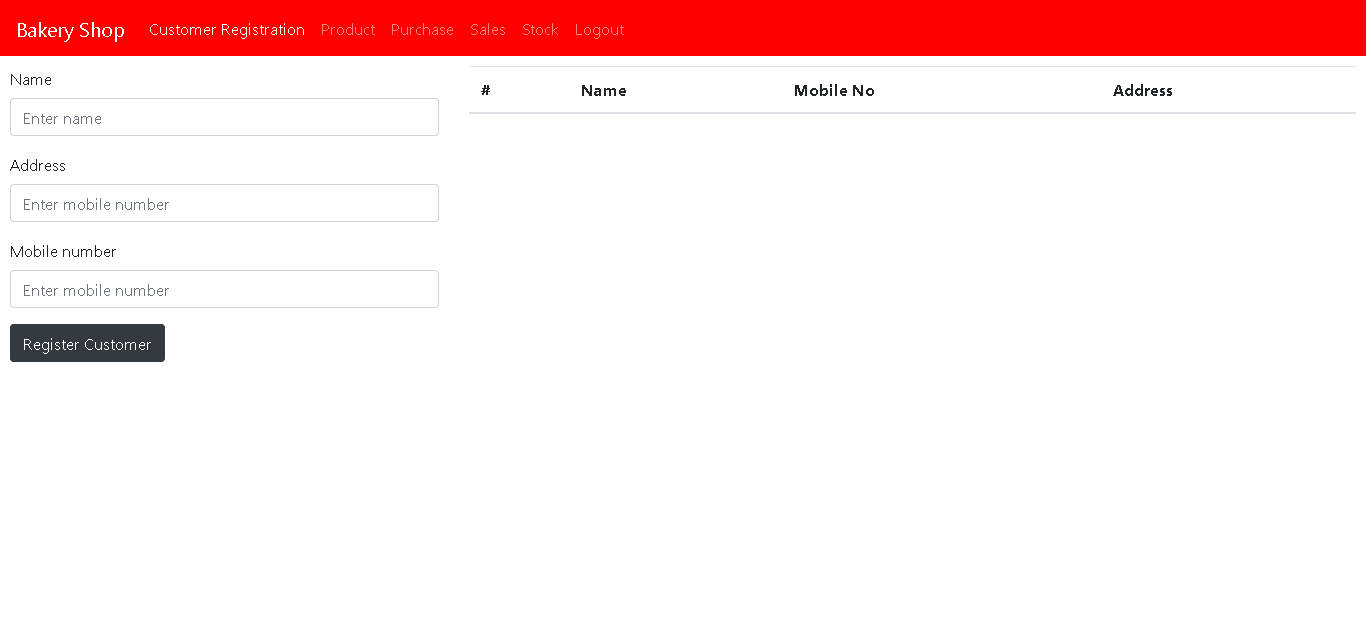
}

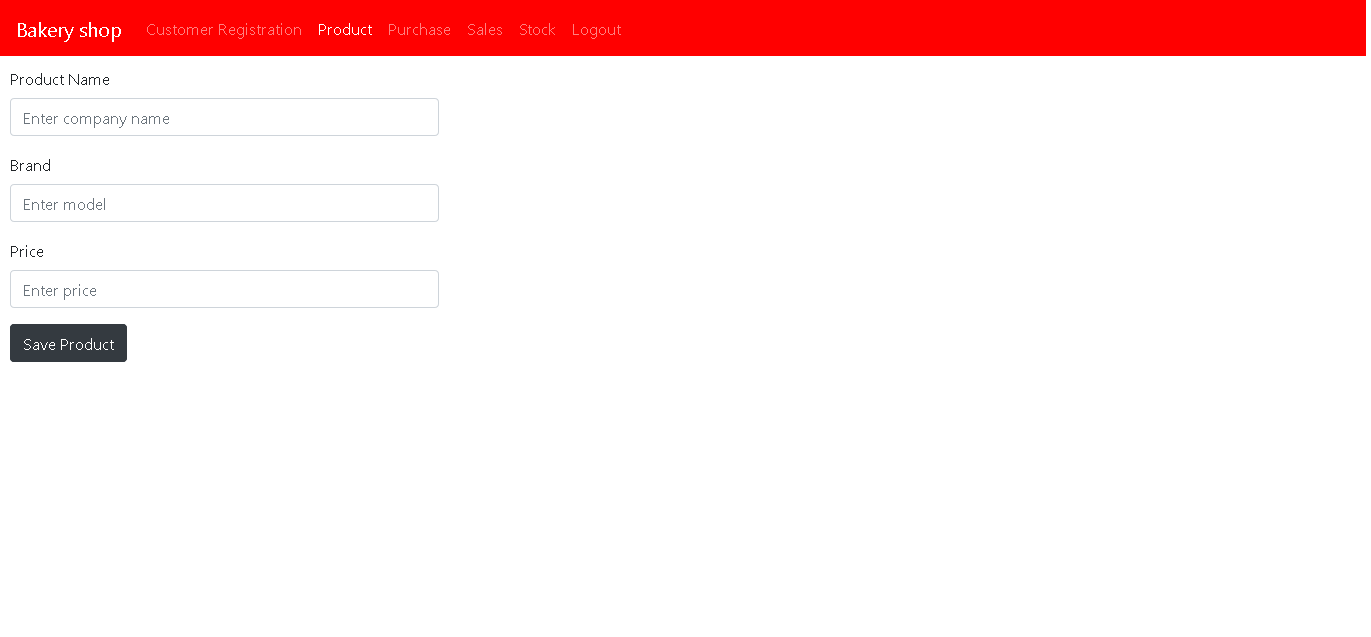
}

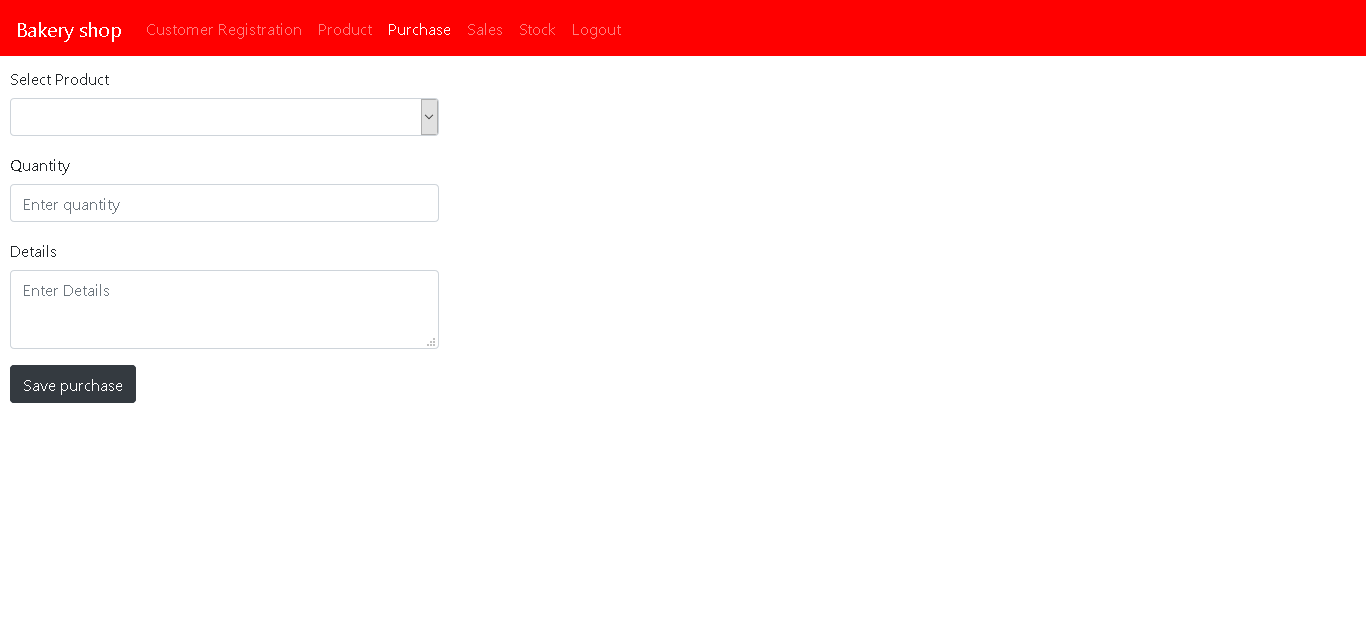
## APPENDIX II

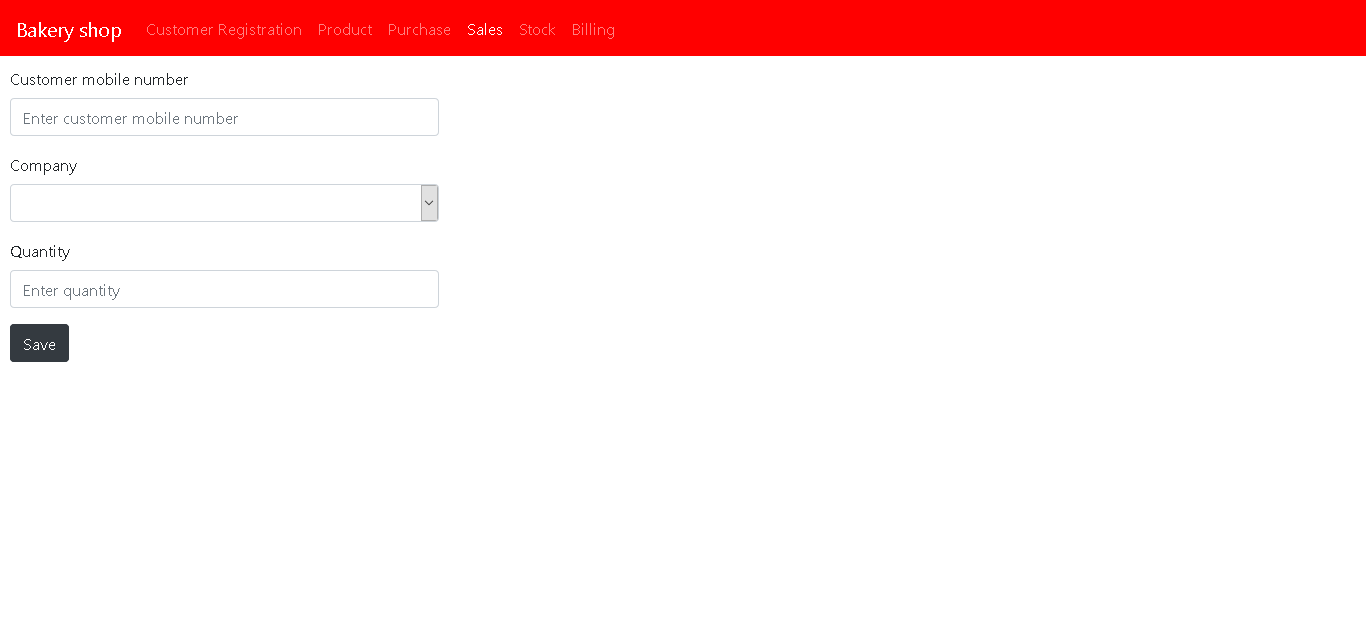
**SAMPLE SCREENSHOTS**

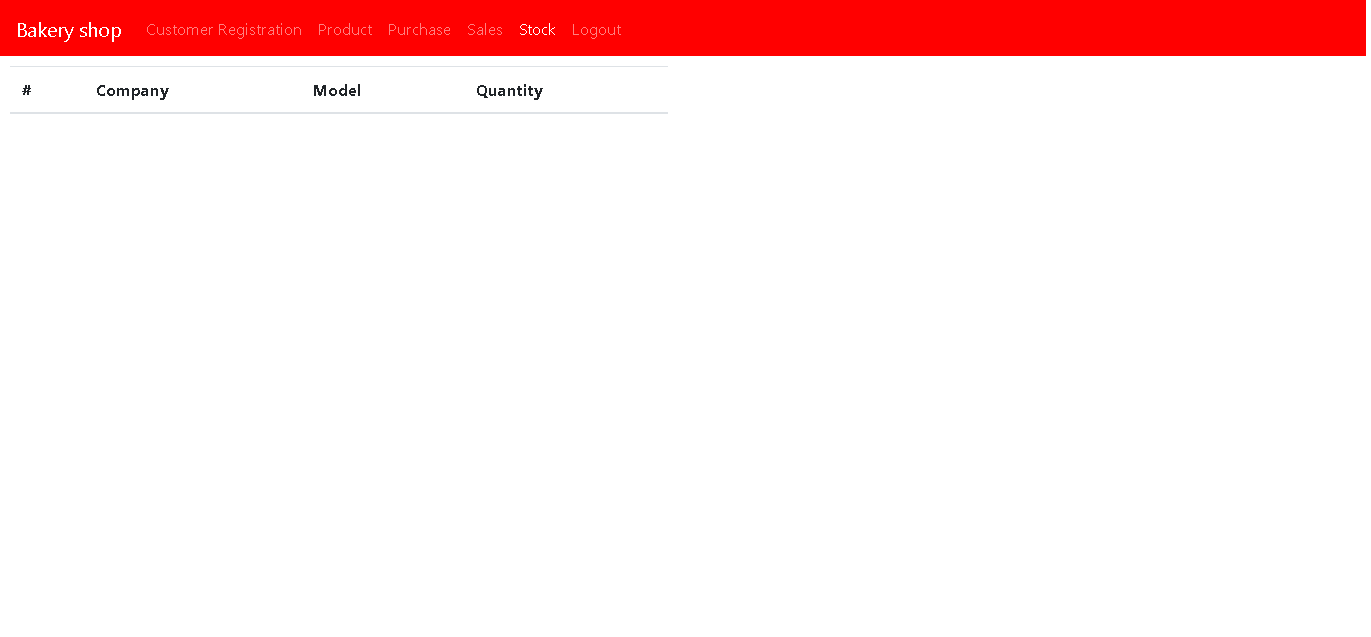












**REFERENCES**

**BOOK REFERENCES**

1. Jeff Forcier, Paul Bissex, Wesley J. Chun, (2018) “Python Web development with Django (Developer’s Library)”, WesleyProfessional.
2. Roger S. Pressman, (2017) “Software Engineering”, Tata McGraw-Hill –Europe.

## WEB REFERENCES

1. https://[www.javaprogramming.net](http://www.javaprogramming.net)
2. https://[www.w3schools.com](http://www.w3schools.com/)
3. https://[www.tutorialspoint.com](http://www.tutorialspoint.com/)