# Acknowledgement:

# Abstract:

Table of Contents

[Acknowledgement: 1](#_Toc6014563)

[Abstract: 2](#_Toc6014564)

[Chapter 1: Introduction 4](#_Toc6014565)

[Background of the project 4](#_Toc6014566)

[Problem statement 4](#_Toc6014567)

[Aim 5](#_Toc6014568)

[Objectives 5](#_Toc6014569)

[Scope and limitation 6](#_Toc6014570)

[Scope 6](#_Toc6014571)

[Limitation 6](#_Toc6014572)

[Features of the project: 7](#_Toc6014573)

[Chapter 2: Analysis 8](#_Toc6014574)

[Analysis 8](#_Toc6014575)

[Requirement 8](#_Toc6014576)

[Functional requirements 9](#_Toc6014577)

[Non-functional requirements 11](#_Toc6014578)

[Prioritization 12](#_Toc6014579)

[NLA (Natural language analysis) 14](#_Toc6014580)

[Structural Diagram 15](#_Toc6014581)

[Initial class diagram 15](#_Toc6014582)

[Behavioral diagram 16](#_Toc6014583)

[Use-case Diagram 16](#_Toc6014584)

[System architecture 20](#_Toc6014585)

[Waterfall methodology 22](#_Toc6014586)

[Chapter 3: Design 24](#_Toc6014587)

[Chapter 4: Implementation 25](#_Toc6014588)

[Introduction 25](#_Toc6014589)

[Choice of language 25](#_Toc6014590)

[Development environment 25](#_Toc6014591)

[IDE 25](#_Toc6014592)

[Deployment strategy 26](#_Toc6014593)

[System migration 26](#_Toc6014594)

[Chapter 5: Testing 27](#_Toc6014595)

[Introduction 27](#_Toc6014596)

# Chapter 1: Introduction

Currently, the project I am working undergoes on L5DC’s module computing project.

The project is on supermarket management system which is the web-based project. It is easy to manage mini markets as the customers entering the details such as the item in the inventory is too less. Human resource deployment and paper works can even manage effectively to the mini-market but no for the case with big-markets. Supermarket is the place where customers come to purchase their daily using products and pay for that. Thus, there is need to calculate how many products are sold and need to generate the bill for their customers in a systematic way.

## Background of the project

The project that I am currently developing is mostly fascinated on automation for supermarket management system. In this system, system will allow the customer to add product in their own cart according to their needs and requirements. Order can be made from their home and can get product at their door. This will be web-based application so the main focus is in providing the user-friendly UI, which allow easy access to the customer in application. Customer can buy products from “**anywhere and anytime**”. By using this system will reduce paper based work, kept data secure, automation in bill generating and regular back-up could be done efficiently.

## Problem statement

The existing system facing the problems in shopping because it was paper based in all supermarkets. Though it is paper work and less manpower, the existing system was not very economical for the market. All kinds of information are stored such as relevant and irrelevant information in the same place. Which make the process very untidy and clumsy? The main problem is time consuming to both the clients and staff in the existing system. Even all the client does don’t get the equal attention in the mart. In case of big supermarkets, the system is some extend computerized but it is not fully automation to cover all the aspects. The data entry, storing and retrieval are very inefficient. There are chances of misplacement and irrelevant information entered. The system is still insecure and not flexible to adapt to user requirements. It was difficult to generate report and the system was not user-friendly. To fulfill their needs and requirements, business has to update as computerized system. To cover the aspects, the online sites are provided which the customer can buy product from their home and get the products at their door. The main aim is to maintain the relationship between the business process and customers and to make healthy relationship among them.

## Aim

* To manage inventories from the system.
* To generate bill for customers satisfaction.
* To make management system automated for both clients and staff.

## Objectives

* To always is the first choice of customers.
* To manage the supermarket activities.
* To maintain the stock details.
* To reduce paper work; so that users can spend more time on monitoring their requirements.
* To reduce time in calculation of sales activities.
* To store huge amount of data in the database this will reduce clumsiness.
* Making UI design user-friendly.

# Scope and limitation

## Scope

According to the given problem statements, the scope of the project are providing the main functionality that can give the customer satisfaction which can meets the specified requirement in the given problem statements. The functionality that includes is paper based work replaced by automated bill generating. Likewise, user-friendly UI design, storing record’s of product, buy products from anywhere at any time when user wants, generating bill according to the user order products including shipping charges as per rules and policy are the project scope.

## Limitation

The current project that will manage product, manage order and could generate bill. However, this project lack upon doing payment system and cart notification. The customer could only pay through hand-cash during delivery. They couldn’t use their any credit card. The customer couldn’t get the discount on any products. User couldn’t see their notification after their product is added in their cart until and unless they see their carts.

## Features of the project:

* + **Login by User.**
    - There is login feature which will allow user to access the system and can buy products.
  + **Add product category**
    - We can add different types of product category to our stock on the basis of customer requirements.
  + **Display product**
    - Admin can display all the products for the customers.
  + **Modify product category**
    - Update the product category in case wrong input is done.
  + **Delete product**
    - Product can be deleted if incase product is expired or something gone wrong.
  + **Navigation** 
    - Navigation allows admin and customers to navigate around the interface.
  + **Create and print bill**
    - Automatic bill is created once customer buys the product and check out from the supermarket. Created bill can be printed.
  + **Search engine**
    - It allows searchers to find a product, with search results featuring product information, Rate and description.
  + **Manage personal information**
    - User can update their personal information provided by system service.
    - Example: name, phone no, address, etc.

# Chapter 2: Analysis

## Analysis

Analysis is the process of breaking down the project for the detail inspection to understand our project in an effective way and to make a proper decision. It’s not a development methodology neither it’s writing a programming language but it supports them.

Analysis is the first phase where we gathered all the requirements which are also broken down into two categories:

* Functional requirements.
* Non-functional requirements.

Furthermore, MoSCoW prioritization and different diagrams are drawn such as initial diagram, ER-diagram, user-case etc.

## Requirement

Requirement is the process of gathering information before doing this, need to be clearly understood, well thought out and balanced of the project.

The requirement techniques are further divided into two categories:

* Functional requirements: refers to “what a system does”.
* Non-functional requirements: refers to “how a system does”.

### Functional requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function ID** | **Title** | **Description** | **Purpose** | **Dependency** | **Impact(MoSCoW)** |
| F1 | Registration | New user must fill up the form to register. | To login into the system. | F2 | Must have |
| F2 | Login | Only valid user can login into the system. | To access the system feature in a secure way. | F1 | Must have |
| F3 | Add new product | Information of product is added in the system. | To view product by admin and customer. | F2 | Must have |
| F4 | Search product | Users can search product to buy their product. | To provide information of product as searched. | F1,F2 | Should have |
| F5 | Edit product | Only admin can edit product if only they want to change. | To change product information | F2, F3, F4 | Should have |
| F6 | View product | Users can view product and their details. | To view more product details. | F2, F3, F4 | Should have |
| F7 | Delete product | Only admin can delete product if they want. | To delete product if it is not available. | F2, F3, F4, F5 | Should have |
| F8 | Order product | Customer will order their product as if they want | To order product by customers | F1, F2, F3, F4, F6 | Could have |
| F9 | View order | Admin can view customer order | To view ordered of product list. | F1,F2, F8 | Should have |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F10 | Add product to cart | Customers can add product to the cart if they want to buy | To add product in cart. | F1, F2, F8 | Should have |
| F11 | View cart | Customer can view items of cart. | To view product of cart. | F1, F2, F8, F10 | Should have |
| F12 | Remove product from cart | Customer can remove items if they want. | To remove product from cart only. | F1, F2, F10, F11 | Could have |
| F13 | View profile | User can view their profile after login succeeds. | To view detail information | F1, F2 | Should have |
| F14 | Edit profile | User can edit their information of profile | To edit profile information | F1, F2, F13 | Could have |

### Non-functional requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Non -function ID** | **Title** | **Description** | **Purpose** | **Dependency** | **Impact(MoSCoW)** |
| NF1 | User-friendly interface | UI design must be user-friendly for the user to access the system | To access system in an easy way. | F1,F2,F3,F4,F5,F6,F7,F8,F9.F10,F11,F12,F13,F14 | Should have |
| NF2 | Faster performance | System should have high performance whenever user access the system | To facilitate the task carry out by user. | F4,F6,F10,F11 | Should have |
| NF3 | Availability | System should be available for the user anytime and anywhere. | To make system available to user. | F1,F2,F3,F4,F5,F6,F7,F8,F9.F10,F11,F12,F13,F14 | Must have |
| NF4 | Maintainability | System should be maintained to run for a long period of time | To recover system at time | NF3 | Should have |
| NF5 | Serviceability | Service should be provided by the system | To make user more attractive | F8,F10 | Should have |
| NF6 | Security | User information must be stored securely from hacking during registration as well as login. | To maintain security | F2 | Must have |

## Prioritization

MoSCoW prioritization is another technique for helping to understand and manage priorities in gathered requirements.

* The letter stand for:
  1. **M**ust Have: The requirements are guaranteed to deliver on selective day.
  2. **S**hould Have: the requirement is important but if it is not then also it is viable.
  3. **C**ould Have: the requirement which has less priority.
  4. **W**on’t Have: the requirement will be delivering in future.

|  |  |  |
| --- | --- | --- |
| S.No | Requirements | Prioritization |
| 1 | Registration | Must have |
| 2 | Login | Must have |
| 3 | Add new Product | Must have |
| 4 | Search Product | Should have |
| 5 | View Product | Should have |
| 6 | Edit Product | Should have |
| 7 | Delete Product | Should have |
| 8 | Order Product | Could have |
| 9 | View Order | Could have |
| 10 | View User | Should have |
| 11 | Delete User | Should have |
| 12 | Send Notification | Should have |
| 13 | Add product to cart | Should have |
| 14 | View cart | Should have |
| 15 | Remove Product from Cart | Could have |
| 16 | View Profile | Should have |
| 17 | Edit Profile | Could have |
| 18 | User-friendly interface | Should have |
| 19 | Faster performance | Should have |
| 20 | Availability | Must have |
| 21 | Maintainability | Should have |
| 22 | serviceability | should have |
| 23 | security | Must have |

## NLA (Natural language analysis)

Abbreviation of NLA is Natural language analysis. It is the process of identification of nouns, verbs and adjectives on the basis of scenario. This NLA helps to proceeds filtration for nouns as candidate classes, verbs as potential methods and adjectives as attributes. The main objective of doing this NLA is to develop the business models and to make a class diagram which is a static model of the system. The given below are the classes and methods which are identified on the basis of scenario:

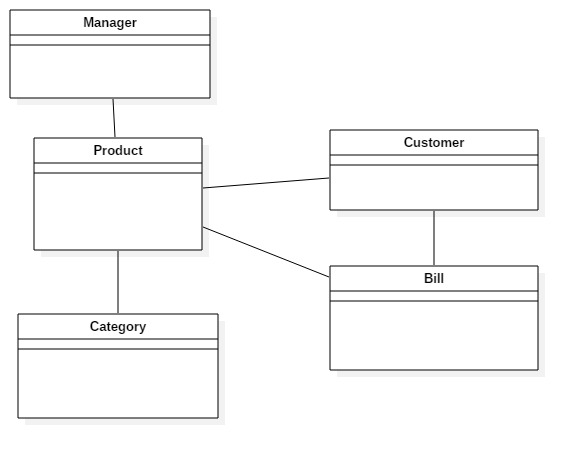
|  |  |
| --- | --- |
| **S.NO** | **Class** |
| 1. | Customer |
| 2. | Product |
| 3. | Product category |
| 4. | Manager |
| 5. | Bill |
| 6. | Order |
| 7. | Cart |

|  |  |
| --- | --- |
| **S.NO** | **Methods** |
| 1. | Add product category |
| 2. | Update product category |
| 3. | Delete product category |
| 4. | Add product |
| 5. | Update product |
| 6. | Delete product |
| 7. | Add to cart |
| 8. | Update cart |
| 9. | Confirm cart |
| 10. | Checkout |
| 11. | Generate bill |
| 12. | View order |
| 13. | View user |
| 14. | Edit profile |
| 15. | Registered user |

# Structural Diagram

## Initial class diagram

Initial class diagram comes under structural diagram which provides the static view of the system and defines the relationship between classes. Initial class diagram only shows the classes and relationship. But the final class diagram shows the attributes, methods, and detailed relationship among classes. It is also used in all forms of object-oriented programming. The given below initial class diagram was developed by using above NLA process.



**Justification**

The above diagram shows the simple relationship between classes such as product, customer, manager, category and bill. One customer will only have one bill, one bill contained multiple products. So, they are dependent to each other. Manager manages the product and products have their own category so they are dependent on category.

# Behavioral diagram

## Use-case Diagram

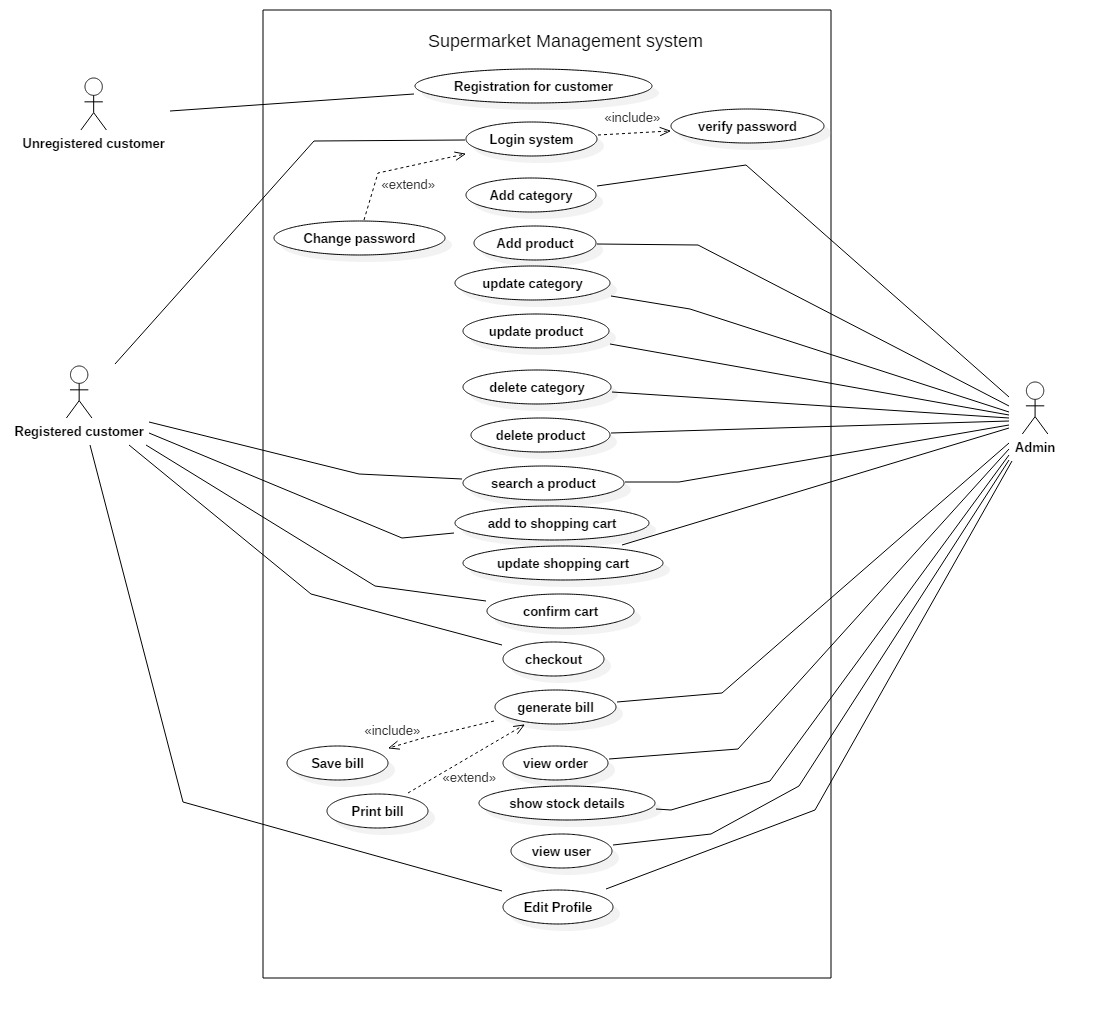
Use-case diagram comes under behavioral diagram. It is also the requirement gathering techniques by analyzing, clarifying and organizing the requirements. This diagram shows the interaction between actors and events and the system flows. To build one, we have use a set of specialized symbols and connectors.

**The symbols and notation that I have used in use-case diagram are given below:**

* **System:** It is shown in a rectangular box which holds use case.
* **Use case:** It is shown in oval shape which represents the system’s function.
* **Actors:** It is a system user.
* **Relationship:** It just shows the relationship between actor and use case by a simple line. Relationship is of two types i.e. **Extend and include.**

**In simple description word,**

1. **Extend:** Optional
2. **Include:** Mandatory



**Justification**

I have used use-case diagram to make an interaction that some system can perform in collaboration with one or more external users of the system. In the above diagram, admin and authenticate user privileges are shown. There are two types of privilege i.e. **high level and lower level privilege.**

In this system, the high level of privilege is given to the admin who can manage product, user and order. Another, the lower level of privilege is given to the authenticate user to access some features. All the above activities are performed after the login process:

* **Add product category:** Only admin can add the different types of product in their store.
* **Update product category:** Admin update the product category details if they want.
* **Manage orders:** Admin manage the customer orders and responded to them. After responding the order, it is deleted by admin.
* **Add product:** Only admin can manage the product by adding in their store.
* **Update product:** Admin can edit the product details as they want to update.
* **Delete product:** Product can be deleted if incase product is expired or something gone wrong.
* **Search product:** User surf their requirement product in search engine on the basis of product category. Also, product is displays in the system like beverage, personal make & care, vegetables & fruits etc.
* **Add to shopping cart:** User add their product as they want.
* **Update shopping cart:** If user wants to continue their shopping then shopping cart is updated.
* **Confirm cart:** After all the requirement product is added then the user click on confirm cart to order their product.
* **Generate bill:** Bill is generated automatically, after the order is confirmed.
* **Edit profile:** User can edit their personal information like their name, address, phone number, email, etc.

**Advantage**

* Functional requirement can be easily gathered.
* It helps in estimating, scheduling and validating effort.
* Easy to collect additional behavioral to improve system robustness.
* Easy to make understand between software developers and end user’s because there is no involvement of technicality in drawing.

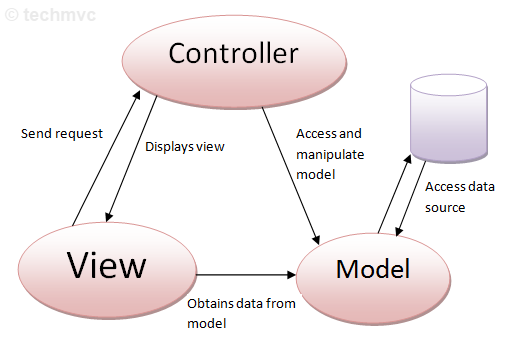
**Disadvantage**

* Difficult to capture non-functional requirements.
* Difficult to manage scenario.
* Use-case structure and flow identification is very poor.

# System architecture

I have used MVC architectural pattern which contains three folders i.e. model, view and controller that is used to separate application’s regard.

* **M** (Model) – It shows the functionality of the application.
* **V** (View) - Deals with the interface. User can view the interface of the system.
* **C** (Controller) – It handles the interaction between model and view.

****

**Justification**

Nowadays, doing web application mostly used MVC in everything. I find it hard to use this pattern. However, I generally understand the idea is to separate the front end and backend which represents the program. Mainly, the views always depend on the controller to some extent and ends up by depending on the model. It helps to improve modularity and facilitate the ease of TDD (test driven development) and testing. By using this MVC architecture, it helps to achieve decoupling in web application. By separating controllers, models and views. Decoupling means changing the code in one area of functionality without disturbing other functionality. (Quora, 2019).

**Advantage of MVC pattern:**

* It helps faster development process because it supports rapid and parallel development.
* MVC has low coupling among models, views and controllers.
* Simultaneously, multiple developers can work on model, view and controller.
* Reduce code duplication because the data and business logic is isolated from the display.

**Disadvantage of MVC pattern:**

* It increase complexity and difficult to understand.
* Need manpower i.e. multiple programmers.
* Required the knowledgeable person on multiple technologies.
* In view, Inefficiency of data access is done.

# Waterfall methodology

In my project, I have used waterfall methodology in which the whole process of software development is divided into different phase which helps to manage my project on a sequential design process. It finishes on stage before another stage can begin. (tutor, 2019)

For the following reasons, I have used this methodology:

* Simple and easy to understand.
* It enables to find error early in the requirement analysis phase.
* Each phase has deliverable and review process which make easier to manage.
* Planning and designing will be straightforward because developers and customers agree on what will be delivered early in the development lifecycle.
* Task can be easily managed.
* In documentation, process and results are well documented.
* Milestone can be easily track.



**The stages of waterfall methodology are:**

* **Requirement gathering and analysis:** In this phase, all possible requirements are gathered to developed and documented.
* **System design:** After the requirement gathering, this phase helps to collect the requirement regarding hardware and system likewise, it helps to illustrate the overall system architecture.
* **Implementation:** System is developed by combining the different units. Each unit is developed and tested in its functionality, which is also known as unit testing.
* **Integration and testing:** After testing each of the unit, all the units are integrated in the implementation phase. If any faults and failures are found then the post integration of the entire system is done.
* **Deployment:** After passing all the functionality and non-functionality testing, the product is released into the market.
* **Maintenance:** if any issues arise in the client environment, patches are released to fix those errors or issues or better version is released to maintain the product quality.

# 

# Chapter 3: Design

This is the third phase of the project which is representing as system design. It is the procedure of designing the elements of a system like architecture, modules, interface, components and the data that is implemented in the system. The main aim of doing this is to provide detailed data and information about the system. (Odhiambo, 2019)

The main components of the system are:

* Architecture: it defines the structure, behavior and many more views of the system which is the conceptual model.
* Modules: system is made by combing the different modules.
* Components: it provides a specific or variety of related functions.
* Interfaces: it is the place where the system exchanges information.
* Data: it helps to manage the information and flow of data.

There are two types of UML diagrams i.e. Behavior and structure diagrams.

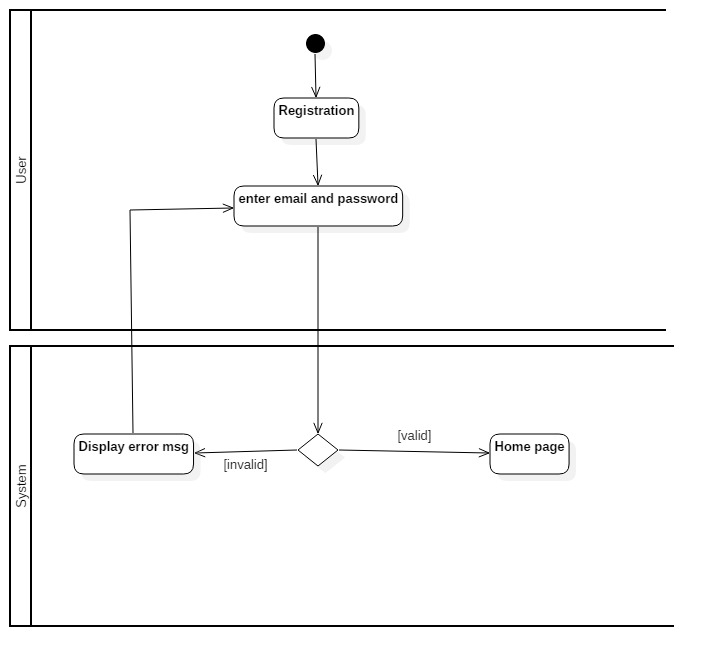
# Behavioral Diagram

It presents the dynamic behavior which is characterized as change to the system over time.

## Activity diagram

Another important behavioral diagram is activity diagram which describe the dynamic aspects of the system and how the activities are co-ordinate. The notations that I have used are given below:

* Activity: Represents a set of actions.
* Action: Represents a task that has to be performed.
* Control flow: It shows the execution sequence.
* Object flow: It shows the flow form one activity to another.
* Initial node: Represents a beginning of a set of actions.
* Activity final node: Represents an end set of actions.
* Decision node: Represents a condition and object flow only one path.
* Merge node: Combing the entire different decision path that was created in decision node.
* Fork node: Splitting the behavior into a set of parallel.
* Swimlane: A group of activities performed by same actor in one activity diagram.



# Chapter 4: Implementation

## Introduction

Implementation is the process of moving an idea from design to real work environment which is done by using within a various programming language. This phase is carried out when design model meet’s the customer assumption.

## Choice of language

For the developed website, I have used PHP language which is an open source scripting language mainly used for developing website which is immerse into HTML. This language was developed for interaction of interface and database by combining with various programs such as common gate interface. I have used PHP because it is flexible for database connectivity, less expensive software and fast load time because it runs in their own memory space.

## Development environment

Windows was used for developed and designed website.

IDE

IDE acronym” **integrated development environment**”**.**

Sublime is a best IDE for developing website. This IDE also supports the framework i.e. laravel framework. The sublime common features are supporting different language, code quality analysis, debugging, cross-platform, instant project switch, etc.

## Deployment strategy

The process of delivering the actual project from development phase to client workstation is deployment strategy. It has different steps to apply this deployment strategy that is given below:

* Release: When the development process finishes, system is ready to release in real environment.
* Activation and installation: executing software components is activation method and using of system is installation.
* Deactivate: just opposite to activation process.
* Adapt: Modifying the system which has been installed.
* Update: Replacing the older version by newer one is update.

## System migration

System migration is a process of transferring the computing assets when older hardware doesn’t provide the level of requirement. Both logical and physical dependencies must be checked during migration and must fulfill at initial stage. List of physical and logical dependencies are:

* Server: It is a device which controls a large user at a time because of high configuration.
* Apache: This serve runs in PHP scripts.
* SQL server management studio: database server
* Composer: Dependency manager of PHP.

# Chapter 5: Testing

## Introduction

Testing is an action to analyze the functionality of a software application in order to find out the specified requirements that the developed software has met or not and in order to produce the quality software, weakness is identify to assure that the product is defect free.

There are different types of testing but I have used unit testing and black box testing which have their own features and goals for testing.

Test case: It is a document of test data, precondition, expected results, actual results and outcome as expected.

**Advantage of test case:**

* Time-saving.
* Provide good quality software.
* Reduce weakness.

**Unit testing**

Unit testing is software testing by breaking the program into units and subjecting each unit to a series of test. Usually tests are run as separate operation but the testing method is different based upon the language and the software.

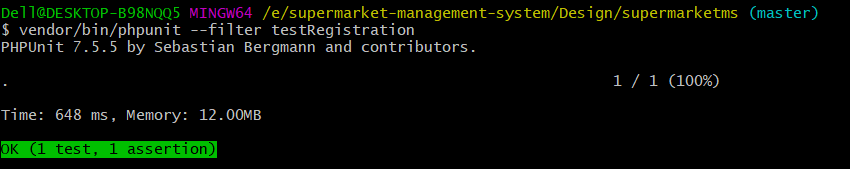
White box is known as code-based testing in which the internal structure of the several programming and coding is tested. This testing is also known as unit testing. The white box testing for my supermarket is given below:

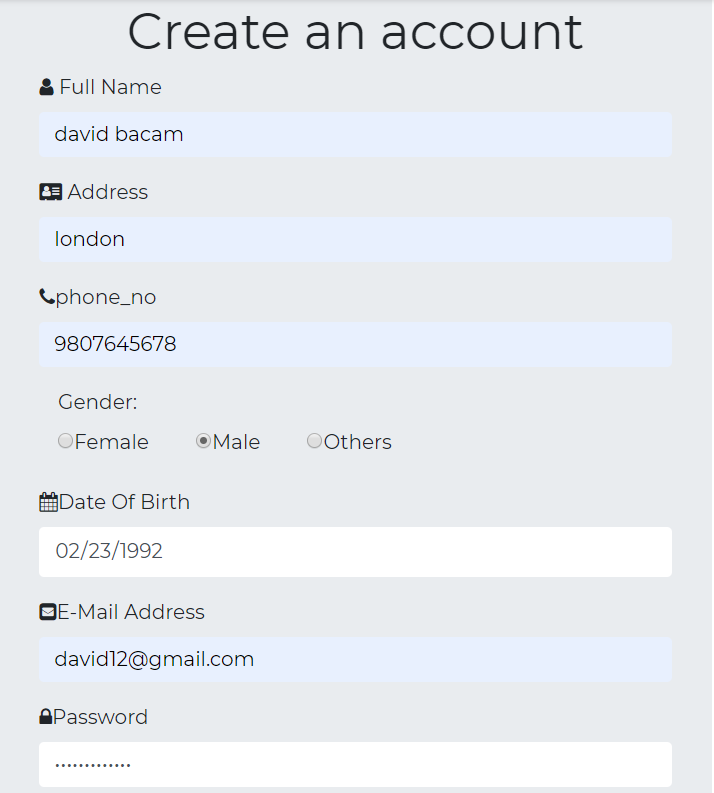
|  |  |
| --- | --- |
| **Test case** | **T1** |
| **Purpose of test case** | User can successfully register or not. |
| **Test data** | Full\_name= david bacam  Address= London  Phone\_no= 9807645678  Date\_of\_birth= 1992-23-02  Email= [david12@gmail.com](mailto:david12@gmail.com)  Password= david12345@#$  Confirm\_password= david12345@#$ |
| **Class name** | supermarketTest |
| **Function name** | testRegistration |
| **Expected result** | To successfully register user. |
| **Actual result** | User registered successfully. |
| **Conclusion** | Yes, the expected result matched actual result. |

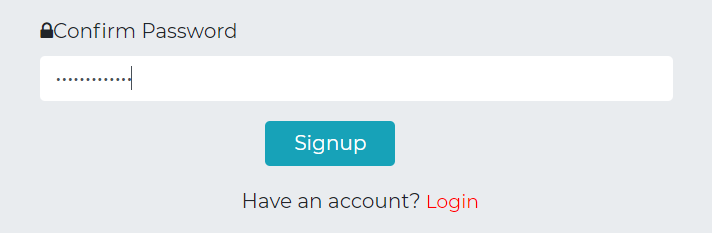
1. Registration

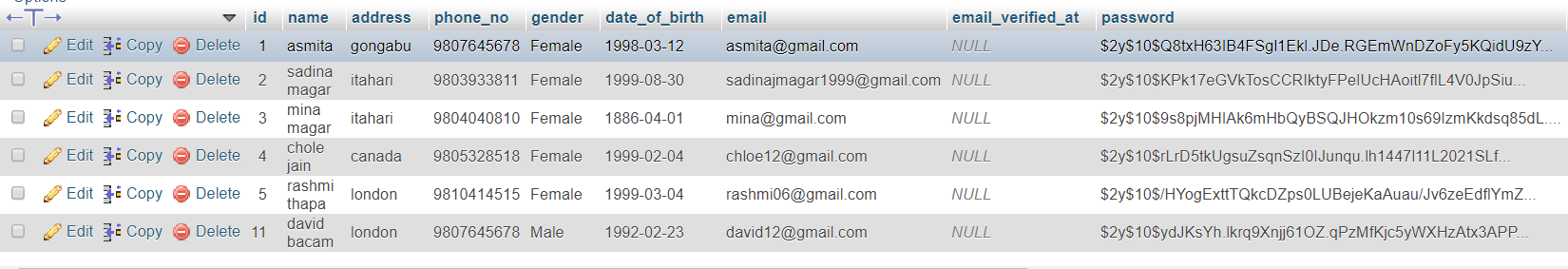
**Unit testing:**



****

**Blackbox testing:**



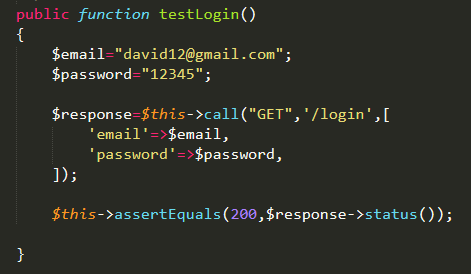


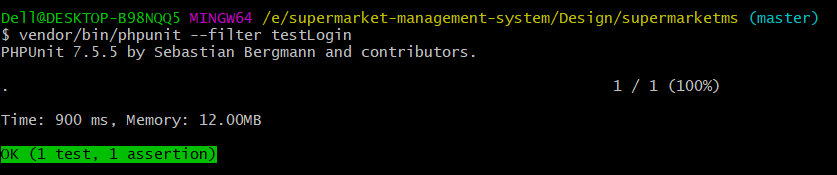
Output:

|  |  |
| --- | --- |
| **Test case** | **T2** |
| **Purpose of test case** | Email and password can be retrieved or not to login the system. |
| **Test data** | Email= david12@gmail.com  Password= 12345 |
| **Class name** | supermarketTest |
| **Function name** | testLogin |
| **Expected result** | To login into the system. |
| **Actual result** | Successfully login. |
| **Conclusion** | Yes, the expected result matched the actual result. |

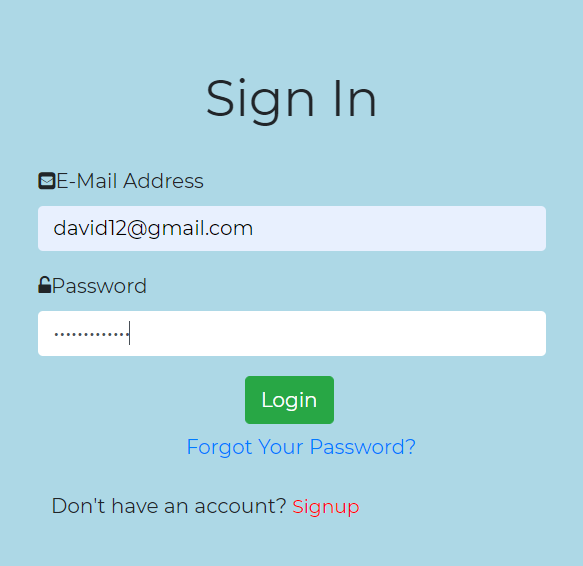
**2. Login**

**Unit testing:**

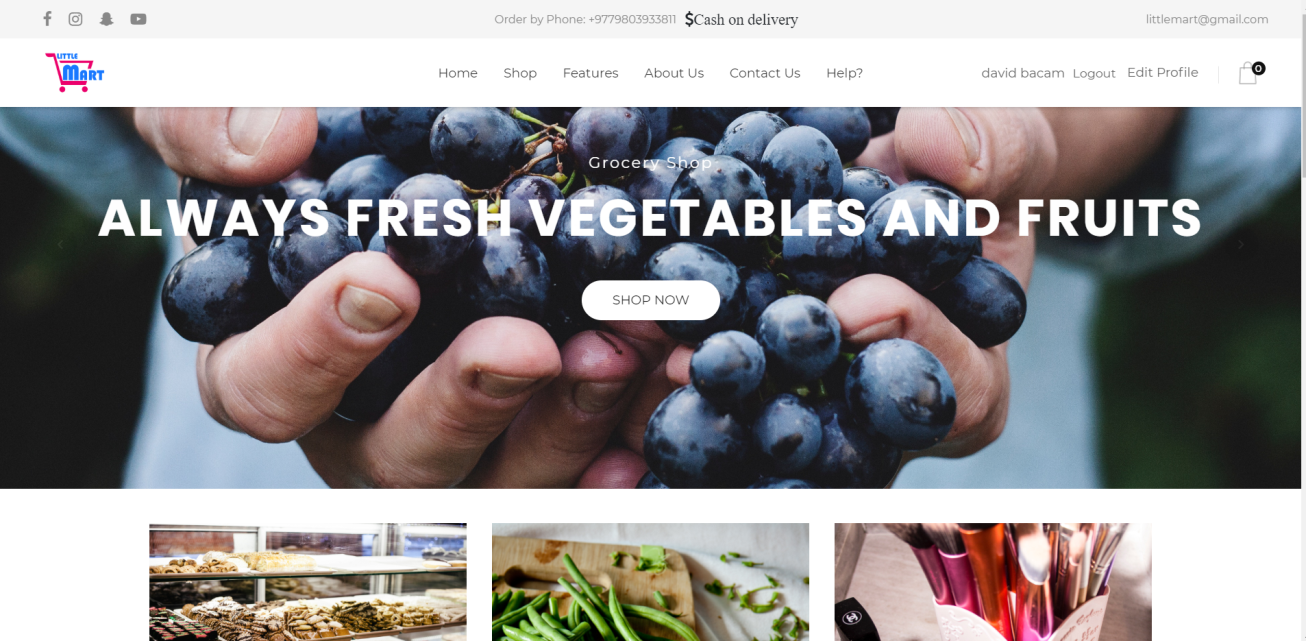
****

****

**Blackbox testing:**



Output:

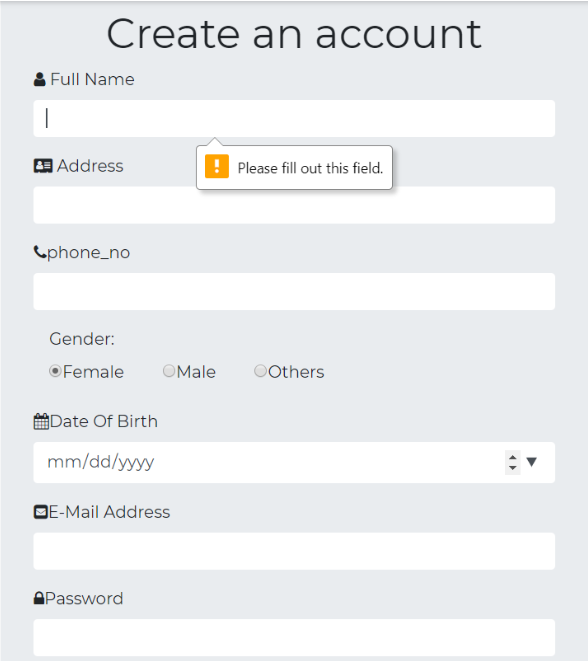


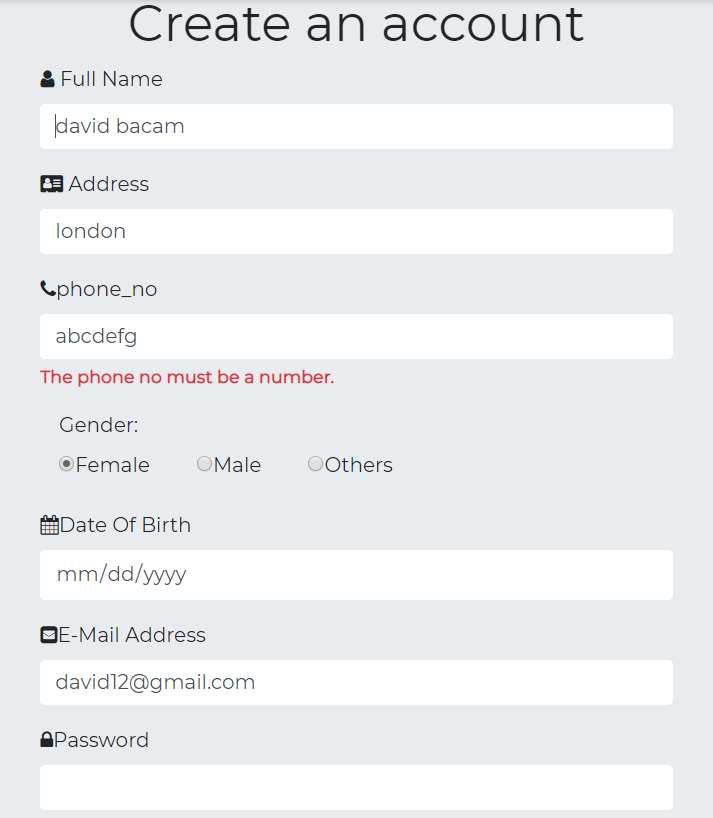
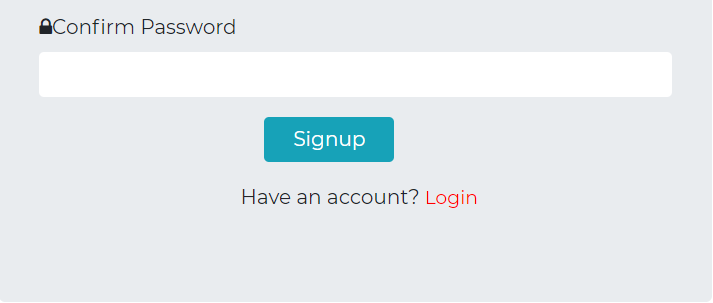
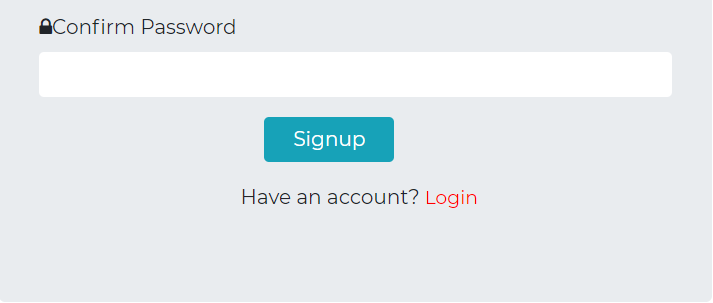
3. Registration validation form

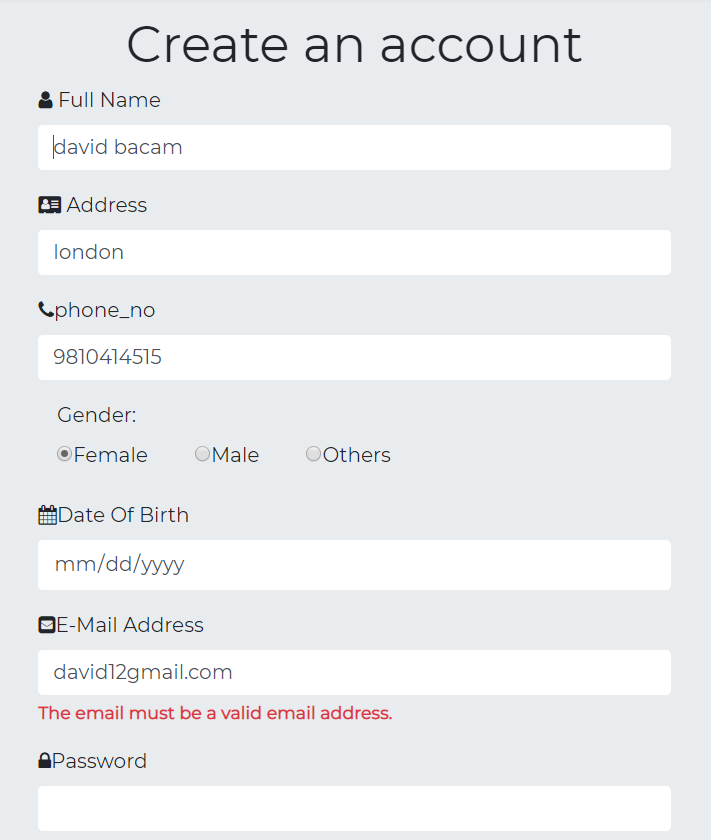
|  |  |
| --- | --- |
| **Test case** | **T3** |
| **Purpose of test case** | To check the validation level of registration page. |
| **Test task** | 1. User doesn’t input the value for the registration  2. User input the alphabetic letter in a number input.  3. User doesn’t provide email in an email format.  4. User input only 5 characters. |
| **Expected result** | 1. Throw error message with “Please fill out this field”.  2. Throw error message with “the phone no must be a number”.  3. Throw error message with “The email must be a valid email address”.  4. Throw error message with “The password must be at least 6 characters”. |
| **Actual result** | Successfully thrown error message. |
| **Conclusion** | Yes, the expected result matched the actual result. |

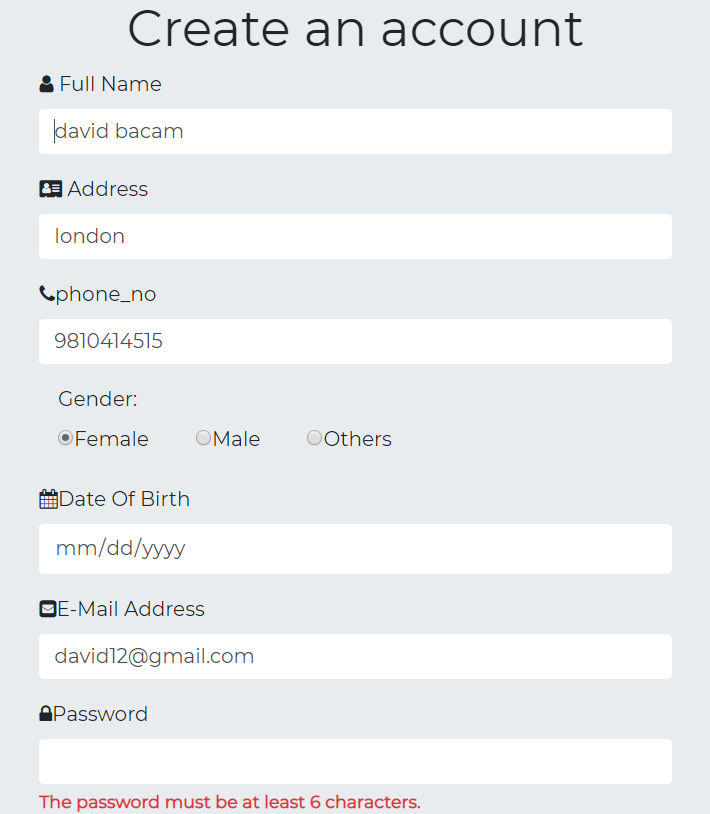
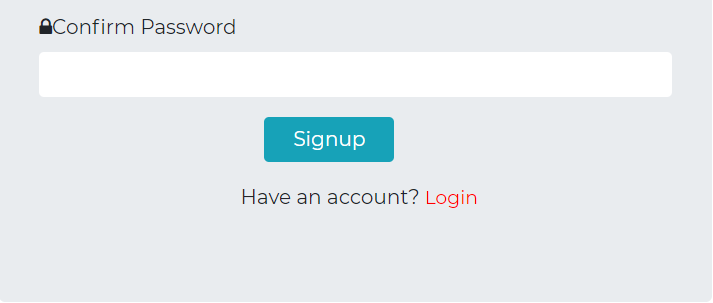
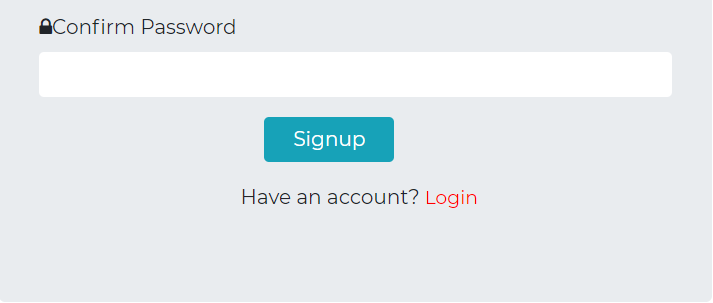
Blackbox testing:

Output:







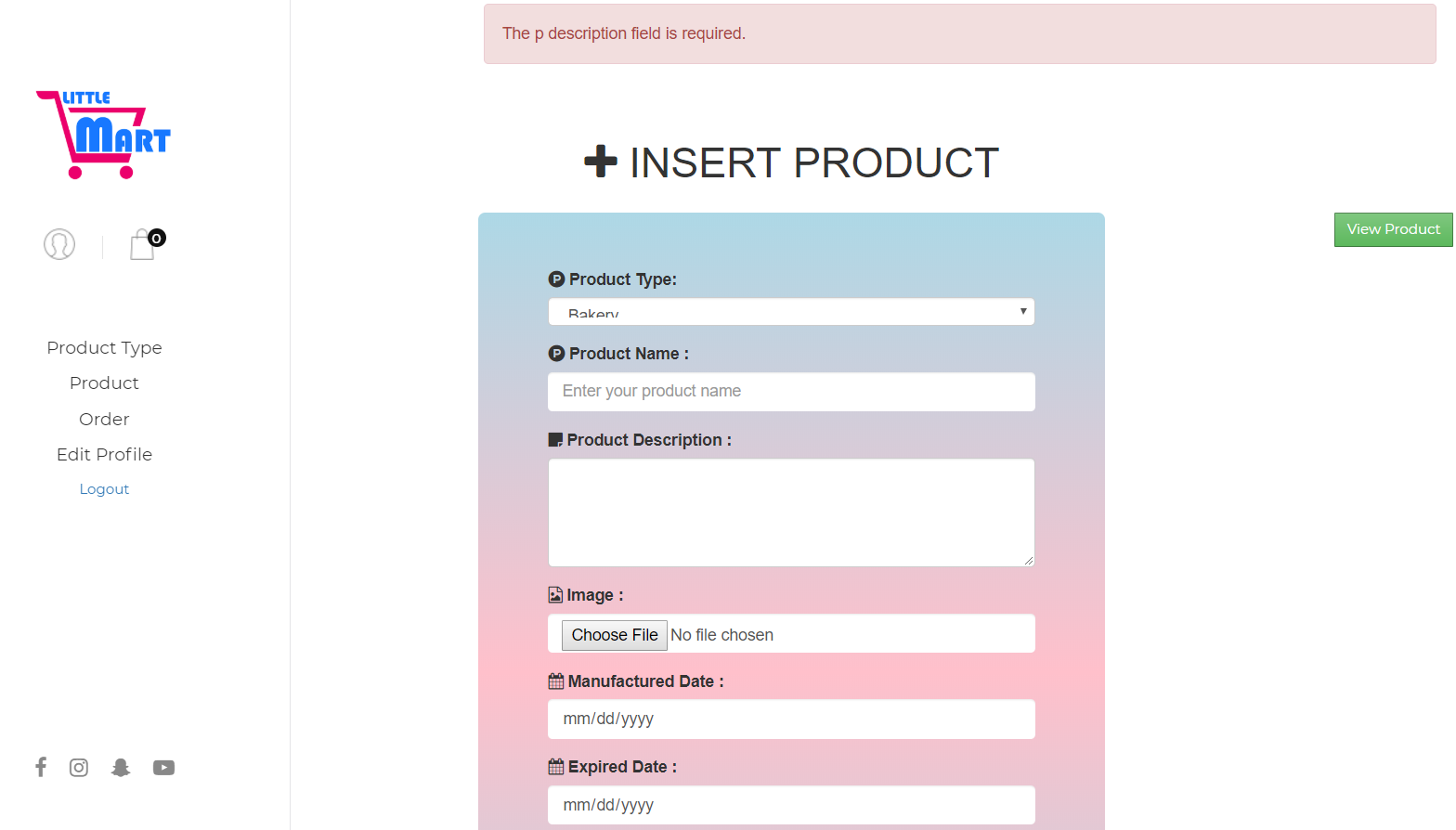


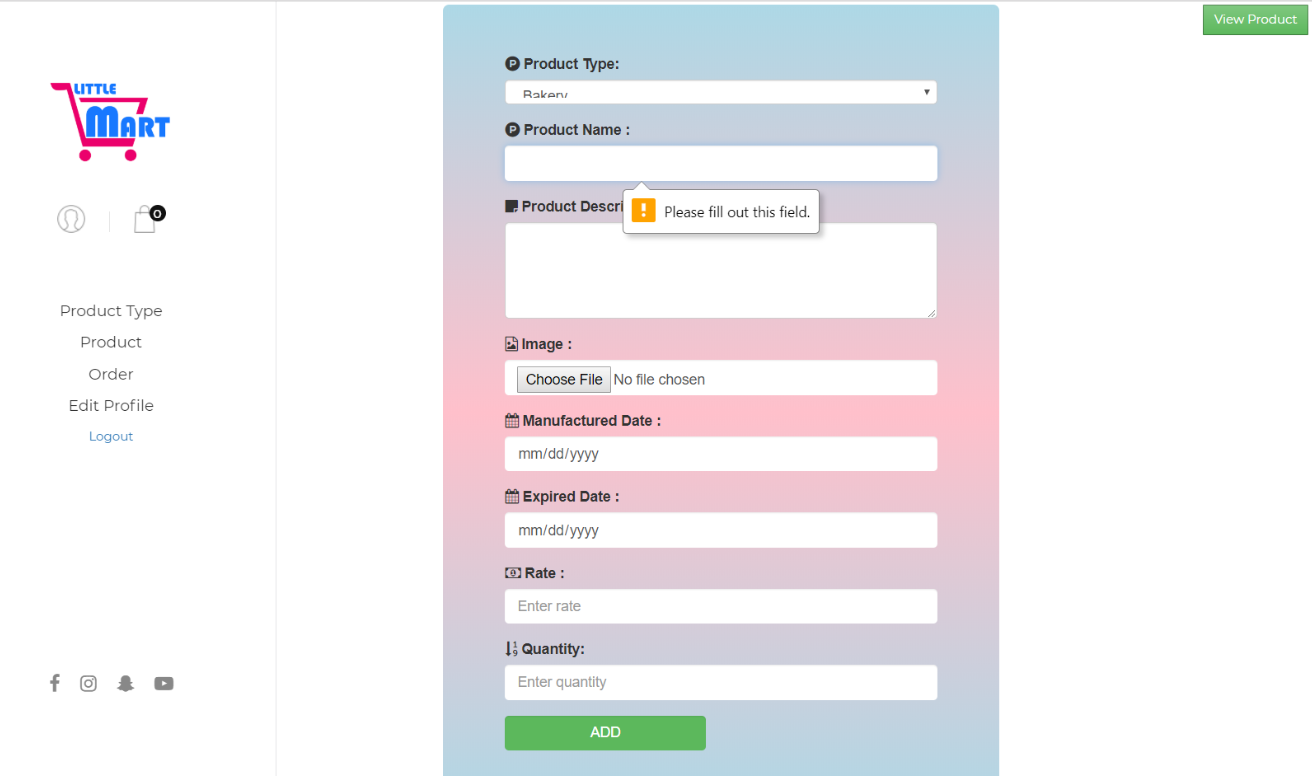
|  |  |
| --- | --- |
| **Test case** | **T4** |
| **Purpose of test case** | To check the validation level of product form. |
| **Test task** | 1. Admin doesn’t provide description to the product  2. Admin doesn’t input any details. |
| **Expected result** | 1. Throw error message with “the P description field is required”.  2. Throw error message with “please fill out this field”. |
| **Actual result** | Successfully thrown error message. |
| **Conclusion** | Yes, the expected result matched the actual result. |

4. Validation of product form

Blackbox testing:

Output:

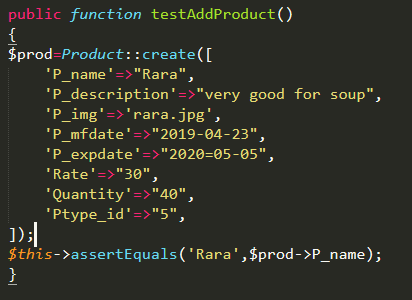




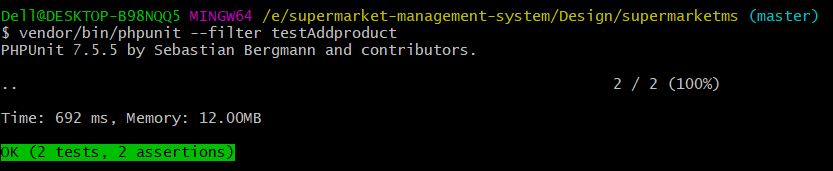
5. Add product

|  |  |
| --- | --- |
| **Test case** | **T5** |
| **Purpose of test case** | To add product by admin. |
| **Test data** | Ptype\_name= Packaged food  P\_name= Rara  P\_Description= very good for soup  P\_img=rara.jpg  P\_mfdate=2019-04-23  P\_expdate=2020-05-05  Rate=30  Quantity=40  Ptype\_id=5 |
| **Class name** | supermarketTest |
| **Function name** | testAddproduct |
| **Expected result** | To successfully add product. |
| **Actual result** | Product added successfully. |
| **Conclusion** | Yes, the expected result matched with actual result. |

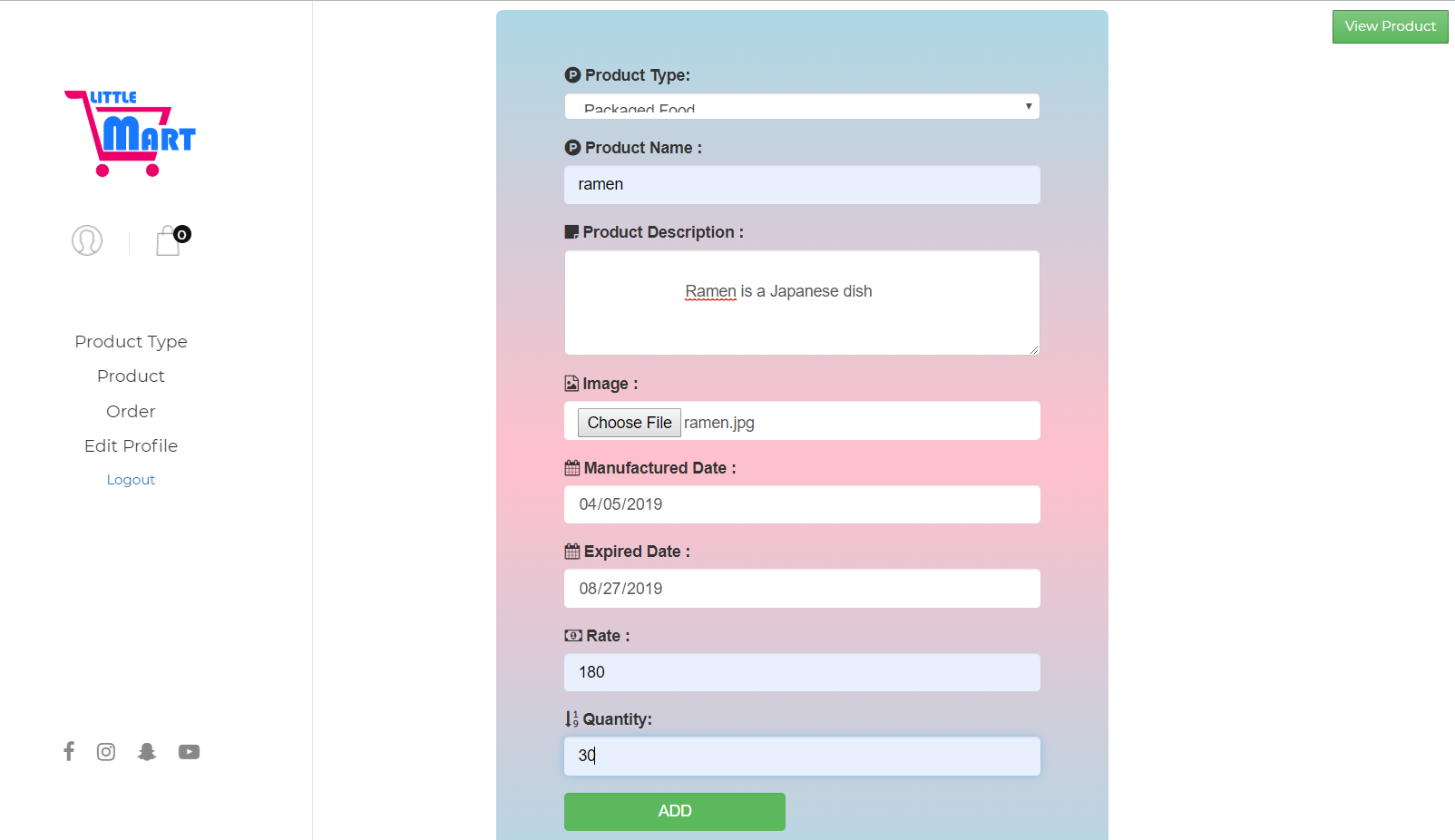
Unit testing:



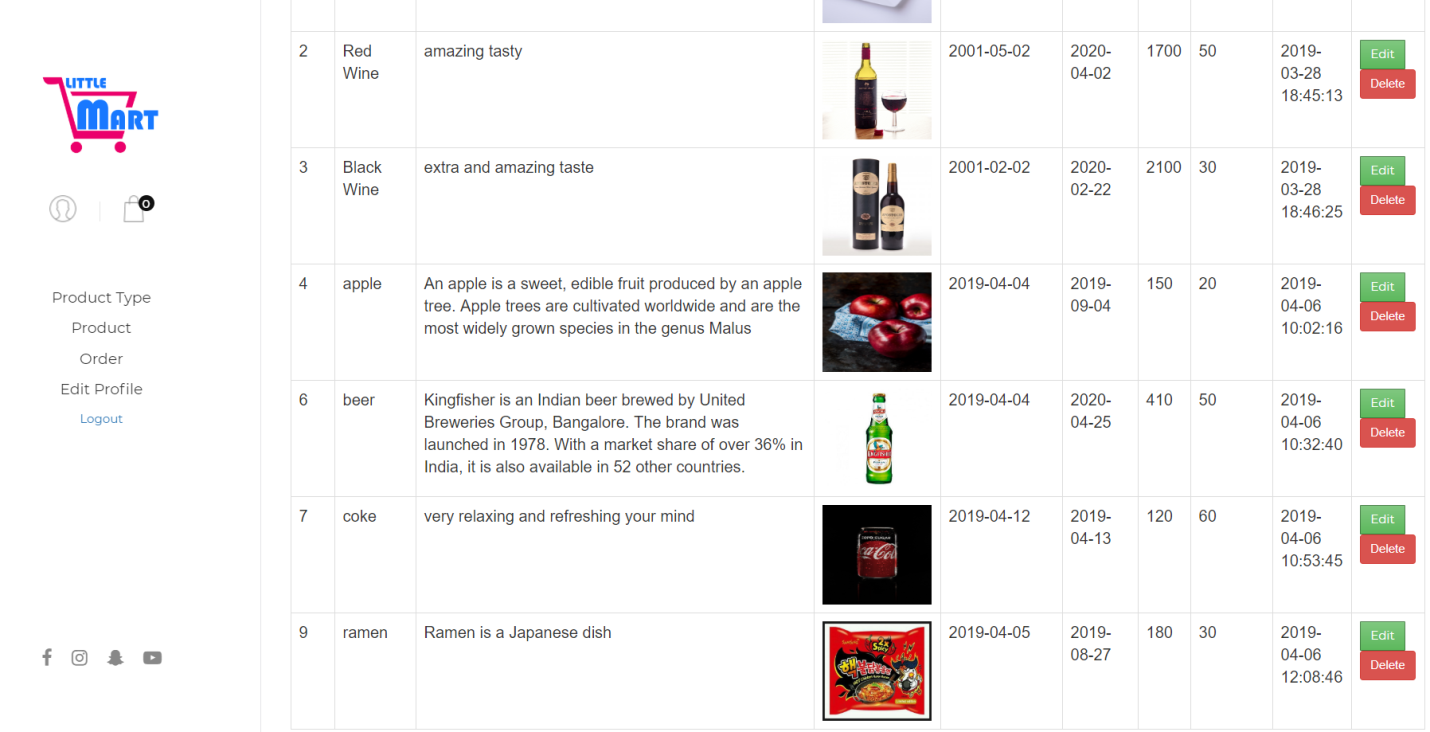
Output:



Blackbox testing:



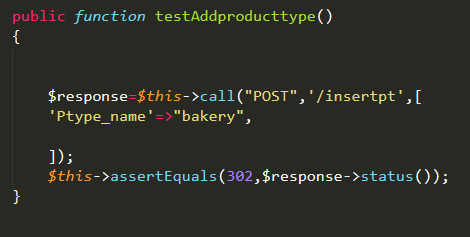
Output:



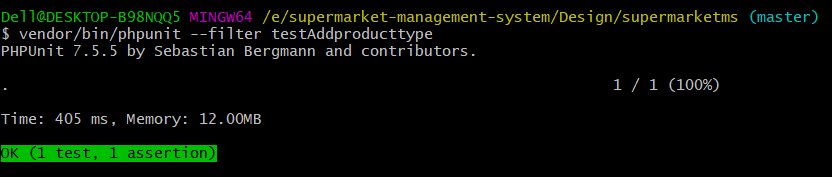
6. Add product type

|  |  |
| --- | --- |
| **Test case** | **T6** |
| **Purpose of test case** | To check product category can add or not. |
| **Test data** | Ptype\_name= bakery |
| **Class name** | supermarketTest |
| **Function name** | testAddproducttype |
| **Expected result** | To add product successfully. |
| **Actual result** | Successfully added. |
| **Conclusion** | Yes, the expected result matched the actual result. |

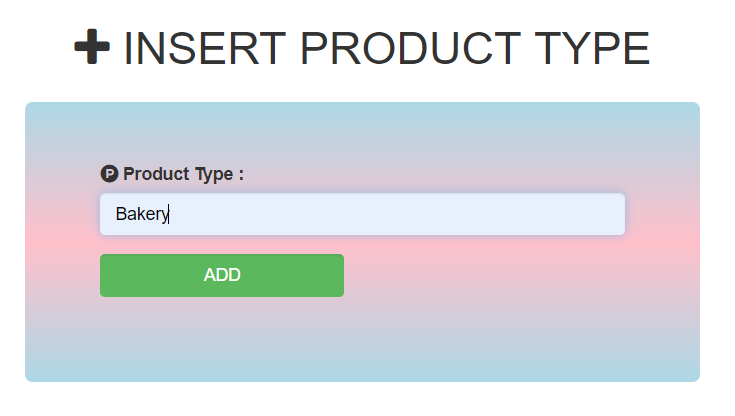
Unit testing:



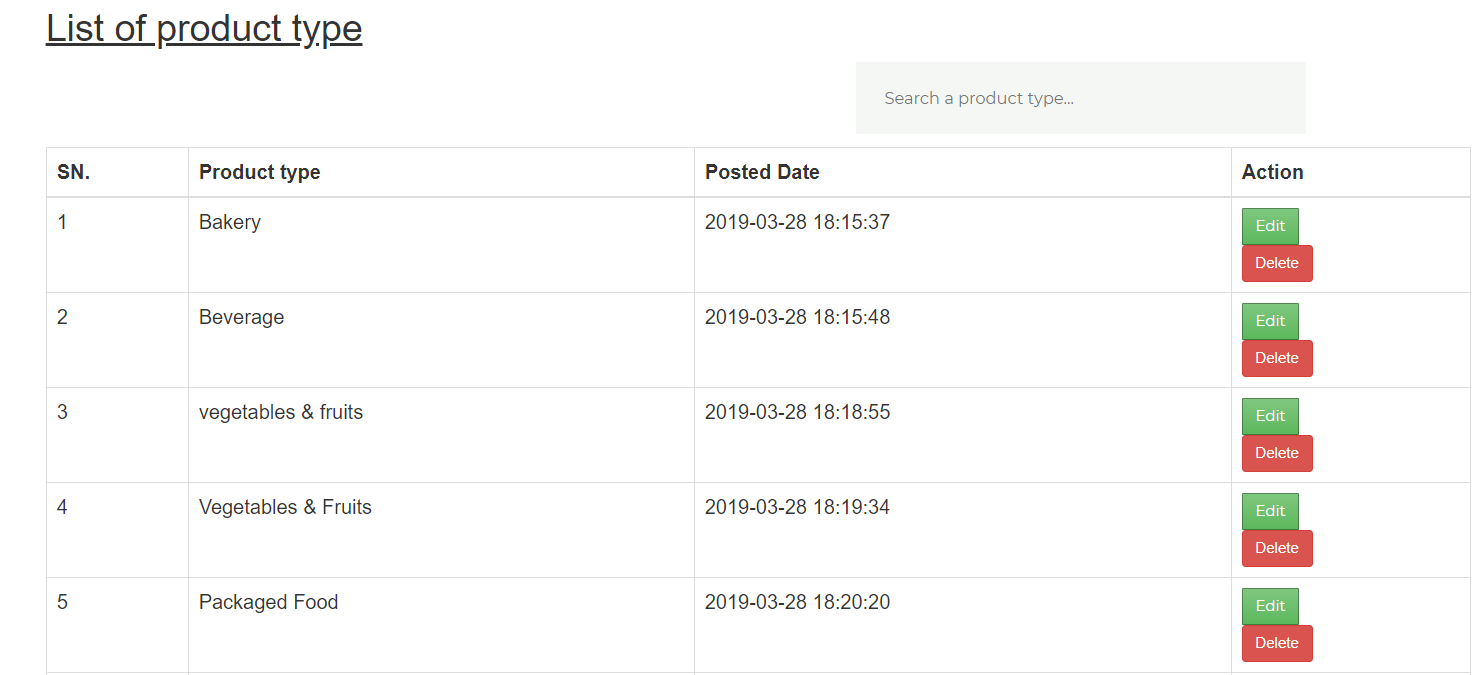
Output:



Blackbox testing:



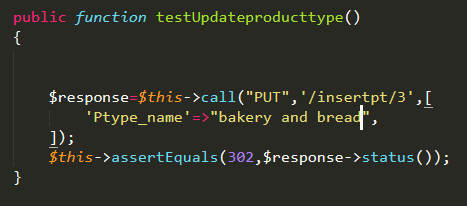
Output:



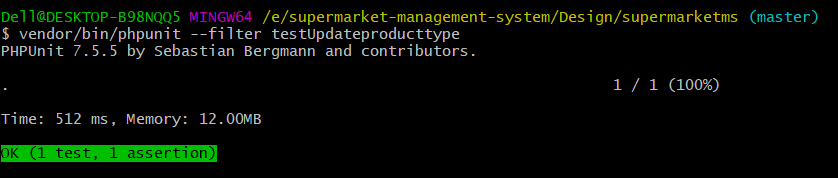
7. Update product type

|  |  |
| --- | --- |
| **Test case** | **T7** |
| **Purpose of test case** | To check product category can update or not. |
| **Test data** | Ptype\_name= bakery and bread |
| **Class name** | supermarketTest |
| **Function name** | testUpdateproducttype |
| **Expected result** | To update product successfully. |
| **Actual result** | Successfully updated |
| **Conclusion** | Yes, the expected result matched the actual result. |

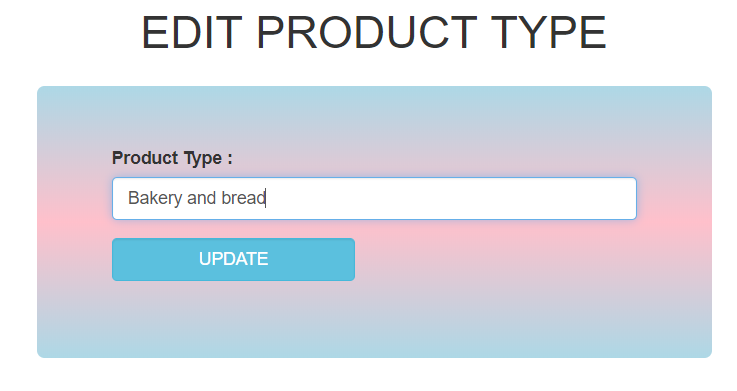
Unit testing:



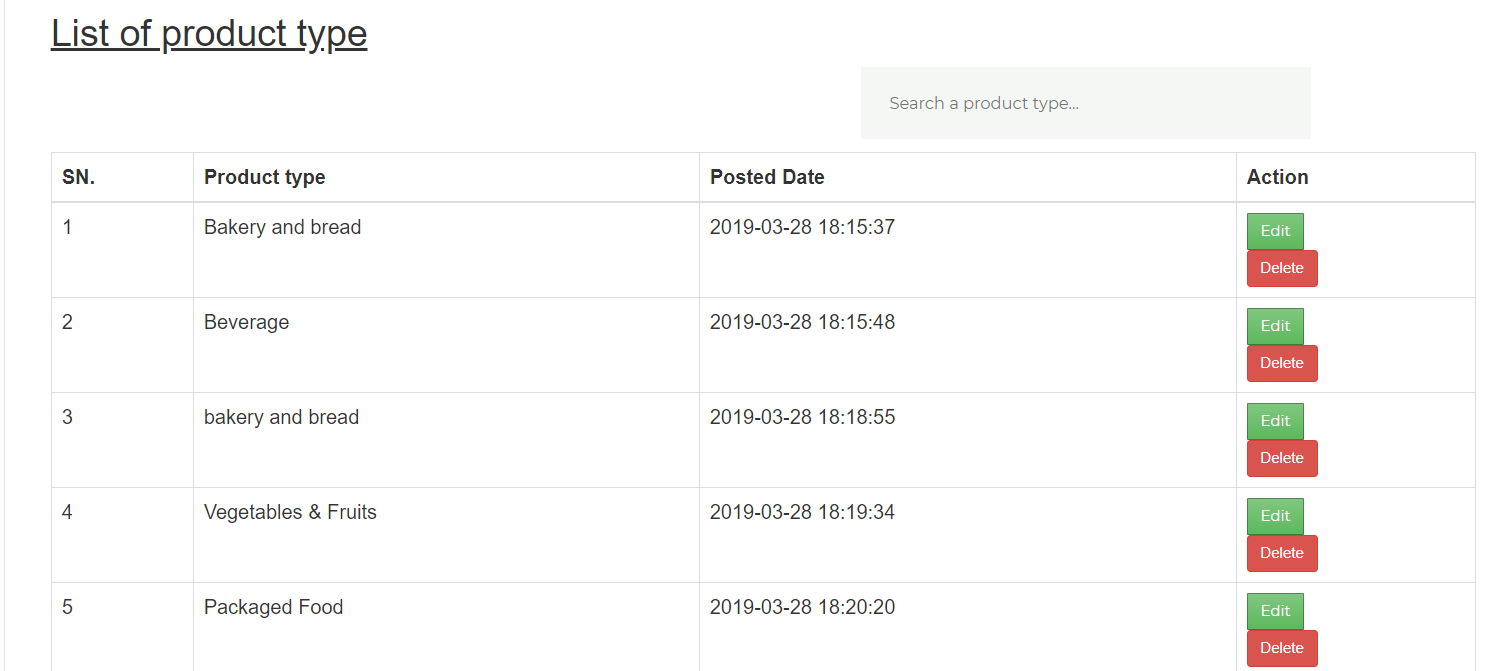
Output:



Blackbox testing:



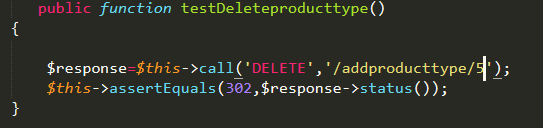
Output:



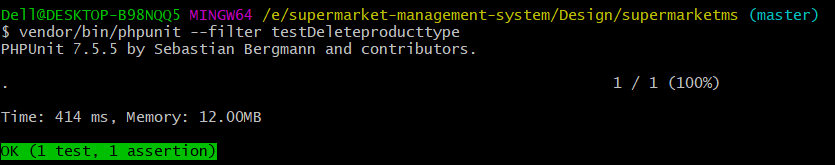
8. Delete product type

|  |  |
| --- | --- |
| **Test case** | **T8** |
| **Purpose of test case** | To check product category can delete or not. |
| **Test data** | Ptype\_name= bakery and bread |
| **Class name** | supermarketTest |
| **Function name** | testDeleteproducttype |
| **Expected result** | To delete product successfully. |
| **Actual result** | Successfully deleted |
| **Conclusion** | Yes, the expected result matched the actual result. |

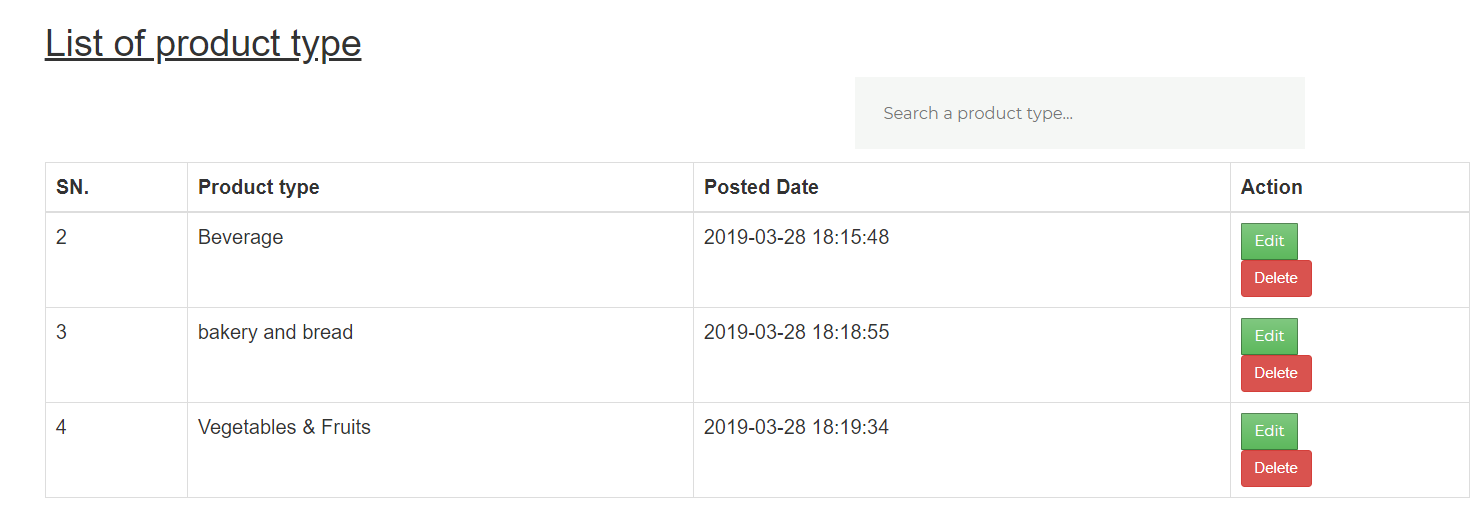
Unit testing:



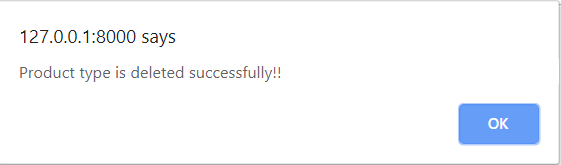
Output:



Blackbox testing:



|  |  |
| --- | --- |
| **Test case** | **T9** |
| **Purpose of test case** | To add product in cart by users. |
| **Test task** | Addition of product in cart by user. |
| **Expected result** | Throw successful message ”your product is added in your cart”. |
| **Actual result** | Successfully product is added in cart. |
| **Conclusion** | Yes, the expected result matched the actual result. |



9. Add to cart

Blackbox testing:

