One-Stop Shop: An E-Commerce Based Web Application

Submitted By
Md. Tanzilur Rahman Rakib
ID: 2012-2-60-017

Md. Atikur Rahman Talukdar
ID: 2012-2-60-018

Supervised By
Md. Shamsujjoha
Senior Lecturer
Dept. of CSE, EWU

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
EAST WEST UNIVERSITY

December 2016

Abstract

This project presents a full functional web application for a medium sized business enterprise. It will provide convenient support for the parties relating to the business *e.g.*, customers, staff, owners etc. This application will also help the business personnel manage and distribute the product effectively. In addition, related information of product, customer support, and online transaction features are available here. The key intent of this project is to maintain a professional standard to compete in the current market of online business.

Letter for Acceptance

This Project entitled "One-Stop Shop: An E-Commerce Based Web Application" submitted by Md. Tanzilur Rahman Rakib(ID:2012-2-60-017) and Md. Atikur Rahman Talukdar (ID: 2012-2-60-018), to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted by the department in partial fulfillment of requirements for the Award of the Degree of Bachelor of Science in Computer Science and Engineering on December, 2016.

Chairperson

____

Dr. Md. Mozammel Huq Azad Khan

Professor and Chairperson,

Department of Computer Science and Engineering,

East West University, Dhaka, Bangladesh

Supervisor

____

Md. Shamsujjoha

Senior Lecturer
Department of Computer Science and Engineering,

East West University, Dhaka, Bangladesh

Acknowledgements

First, we are thankful and expressing our gratefulness to Almighty who offers us divine blessings, patience, mental and psychical strength to complete this project. The progression of this project could not possibly be carried out without the help of several people who, directly or indirectly, are responsible for the completion of this work. We are deeply indebted to our project supervisor Mr. Md. Shamsujjoha. His scholarly guidance, especially for his tolerance with our persistent bothers and unfailing support. He gives us the freedom to pursue aspects of reversible fault tolerant computing which we found interesting and compelling. This helped our project to achieve its desired goals.

We wish to thank the great people of Department CSE at East West University. A special thank goes to all faculties for their well-disposed instructions and Encouragements.

Finally, we would like to thank our friends and family. Their continued tolerance with our moods and tendency to disappear for weeks at a time gave us a much needed break from the world computing.

Table of Contents

List of Figures

Chapter 1

Introduction

One-Stop Shop is an e-commerce based web application that makes the trade for buyers and sellers in few clicks. This application helps the business personnel (*e.g.*, administrators, managers) to easily manage users and products, apply or remove sales, keep track of orders etc. In addition, subscribers can search and buy products through secure transactions online and see their transactional histories later. The scenario seems pretty basic, and the idea is to keep it that way through backend complex fail-safe design.

1.1 Motivation

For this project we considered building a full functional web application because of two key factors:

i. In recent years, e-commerce has been dominating the world of trades and business. Due to the ease-of accessibility of the internet and microcomputer devices along with the simple manner of usability, it has been a sector where more and more people are merging every day. We wanted to get a little insight of the practical events and issues related to this sector.

ii. Our goal was to build a professional standard project, which would help us to get an idea about how things work in real world (i.e., software development firms). Working with Laravel has enabled us to become a little familiar to such kind of a workflow.

In short, mixing and exploring two of the largest sectors that not only keeps growing fast in different parts the globe but also shows a great deal of promise in Bangladesh's near future as well- was the actual motivation behind our project.

1.2 Key Features

- **E-mail verification** for user registration
  - **Advanced User Info** can be added / updated later

- **Proper user redirection** after different tasks

- **Fail safe** manner for user inputs
  - Front end warnings and restrictions
  - Strict **Server-side validation**

- **Free from data redundancy**
  - Thoroughly used Laravel's ORM (Object Relational Mapping)
  - Framework's **OOP** concept gives extra benefit in keeping the data safe and the security intact

- **Autocomplete suggestions** for search queries

- **Find products by Categories, Brands etc**.

- **Search** products conveniently
  - **Sort by attributes**

- Shows **Recently viewed products**

- **User Feedbacks** on products can be added and deleted by users (the same user)

- Administrators and Managers can **add Products**, **Categories, Brands**

- Add **Sale** on products
  - Easily add/remove product from sale
  - **Sale ends on schedule automatically** (Scheduled Task)

- **Product Checkout** is done safely and securely using stripe.
    - Account authentication is validated

- **Stock availability** is validated throughout multiple step of order
    - During Adding to Cart
    - During final checkout

- **Order status** can be monitored by managers/admins afterwards
    - Staff can change

- A **Printable Receipt** is given to the user after the checkout is completed
    - Includes **unique receipt number,** transaction date etc.

- Users' **Cart** info is saved in **Order** table inside database
    - Order includes payment_id, user_id etc.

- **User support**
    - **Contact** staff though messaging to get info
    - Create **tokens** for refund.
    - **Get help** from staff through messages

- **Related Products**
    - Suggest related products

1.3 Scope

Guests:
(Unsubscribed users)
- Search products
- View product details
- Add to cart (Cart access will result in a prompt to login/register)

Subscribers:
- Search products
- View product details
- Add reviews on products

- Add to cart
    - Increment, decrement products by 1
    - Discard a product from cart
- Delete cart
- Checkout
- Check previous transactions
- Change/Reset password
- Store/Change users' advanced info
    - View self-profile
- Contact staff

Managers:

- Create/update products
- Create/update categories, brands
- Order status checking
- Update product stock
- Create Sales (Sales are set to be deleted automatically at ending date)
    - Delete sales manually
- Assign Sales on products
- Give supports and solution to users

Administrator:

- All of the tasks that a manager can carry out
- Complete power over users (including managers)
    - Add/delete users
    - Change activation status
    - Change user roles etc.

1.4 Outline

The next chapter (Chap.2) discusses the requirement analysis of the project. Diagrams from different perspectives and software requirements are also shown in the chapter.

Chap.3 discusses the application's functionalities i.e. discussion of special and usability features in detail.

Chap. 4 discusses the implementation plan of the project, i.e. the approaches taken for completion and maintenance of individual and cooperative tasks.

Chap.5 shows the technical procedures for modules of the application. Some potential risk and problematic scenarios and their solutions, preventions and validation processes are discussed in the chapter.

Chap 6 discusses the view optimization in brief.

Chap. 7 finally discussed about Conclusions and Future work.

1.5 Summary

This chapter demonstrates motivations, features and scopes of this project. A brief elementary instructional text of remaining chapters of this project have also been described in outline.

Chapter 2

Requirement Analysis

Requirement Analysis, also termed as SRS (Software Requirements Specification) is the process of finding the requirements for developing a software, i.e., data-dependencies, software-dependencies, time-constraints, feasible features to be applied etc. Usually this includes interviewing personnel from client or buyer side, who provides with the primary expectations of what the application might or should do. As we do not have any clients, we brainstormed to find out some functional requirements of the web application.

2.1 Functional Requirements

- Users must be able to search for products
- Users must be able to view product details
- On registration, users should have to verify their e-mails
- Users must be able to add or delete products to cart
- Subscribers should be able to add feedback on products
- Subscribers should be able to view and edit their profile info
- Subscribers must be able to change/reset password
- Subscribers should be able to view transaction histories later
- Products should be able to be set on sale by office staff (i.e., managers, administrator)
- Managers must be able to add, update and delete products
- Managers must be able to add, update and delete brands, categories etc.

- An interactive communication system should be set between managers and subscribers for user support.
- Managers should be able to see and change orders' current statuses
- Admin must be able to carry out all the tasks a manager can do, additionally should have complete power over all the users.

2.2 Non-functional Requirements

- E-mail verification
- Server-side validation
- Proper routing
- Ease-of-use
- Stock validation
- Payment validation
- Redundancy free

2.3 Class Diagrams

2.3.1Application Class diagram

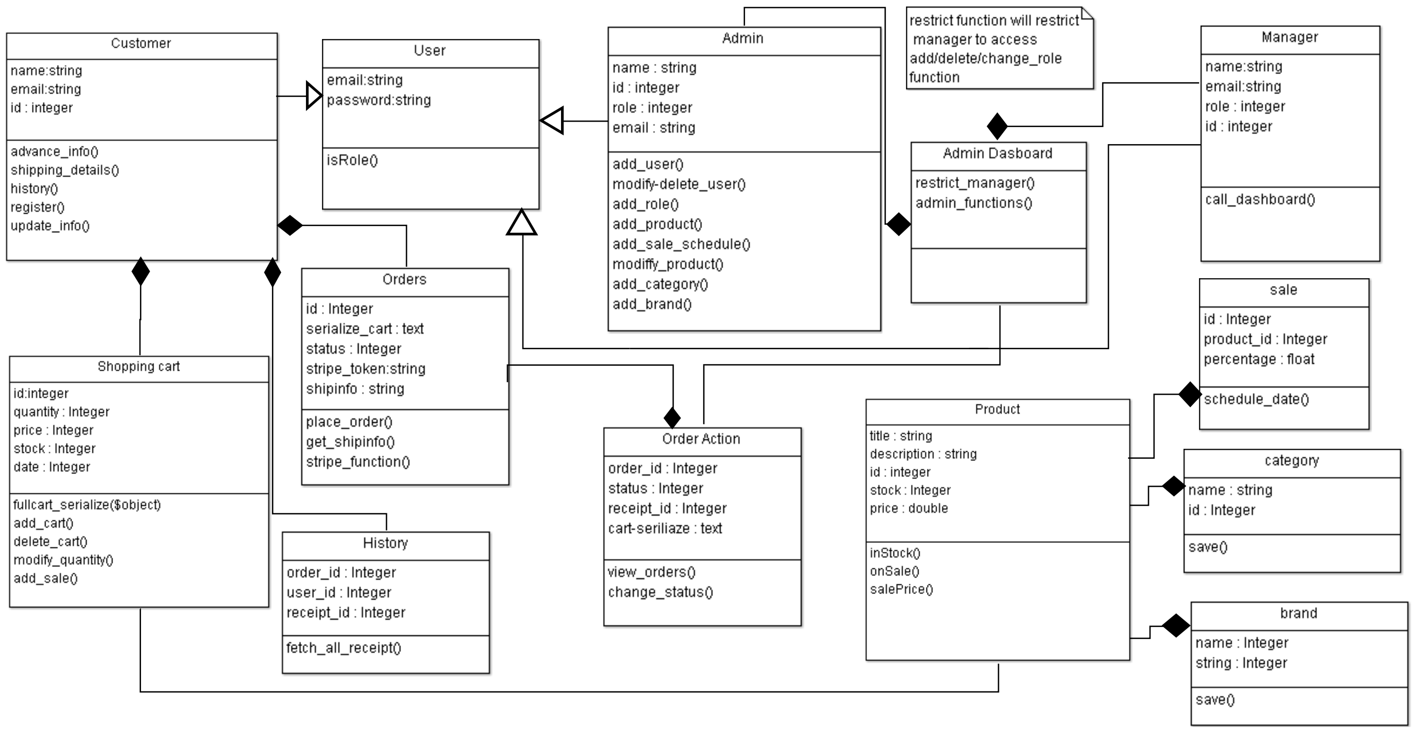


Fig 2.1: Class diagram of the application.

2.3.2 Class Diagram for the Admin

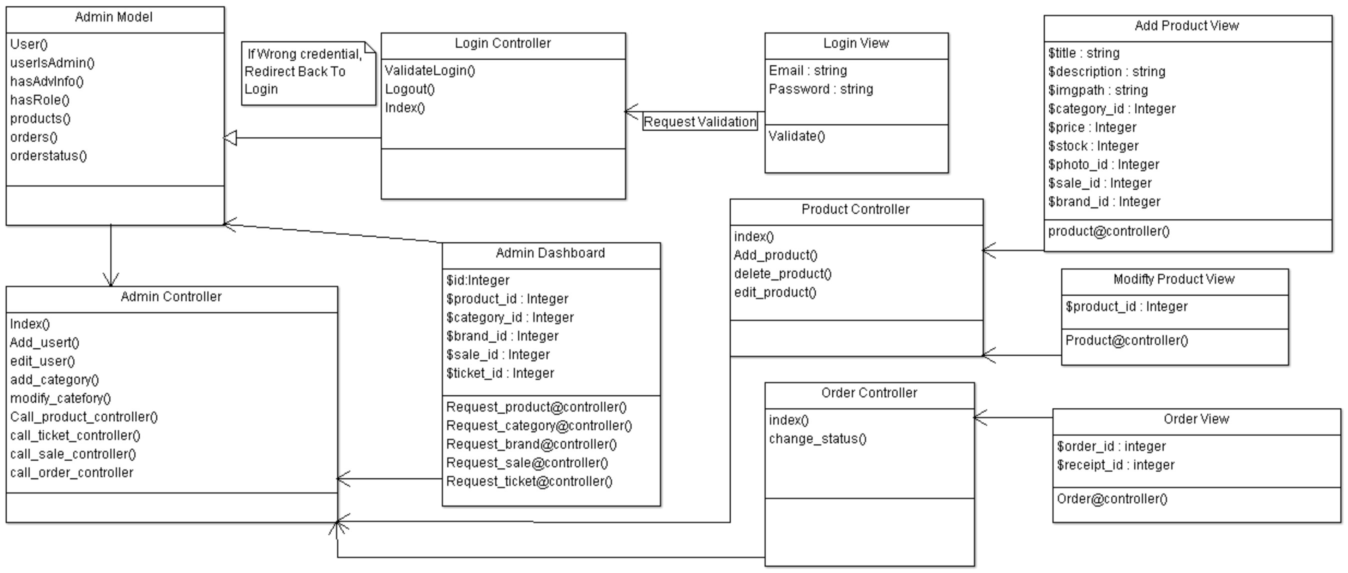


Fig. 2.2: Class diagram for the Admin.

2.4 Use Case Diagram

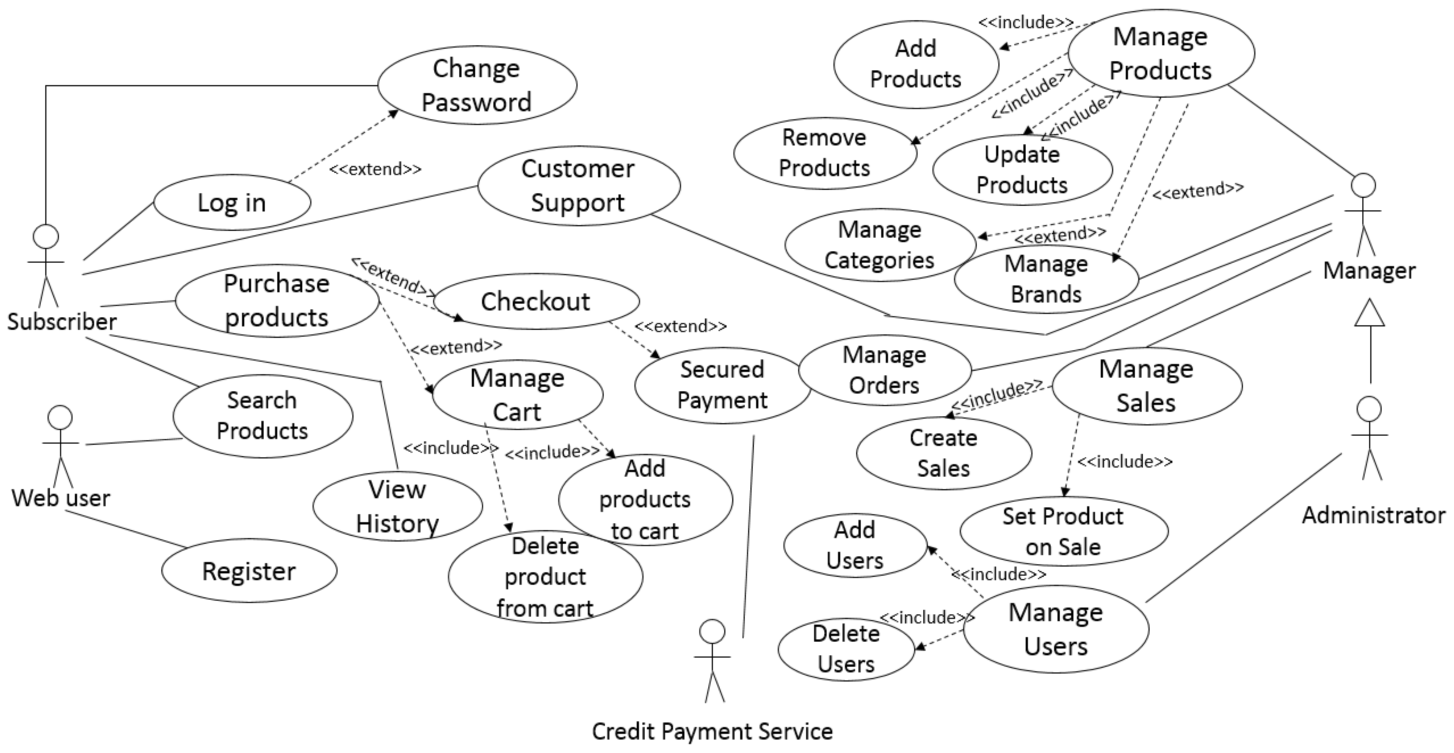


Fig. 2.3: Use Case Diagram

2.5 Activity Diagram

The activity diagram below shows the process of product purchasing using the application.

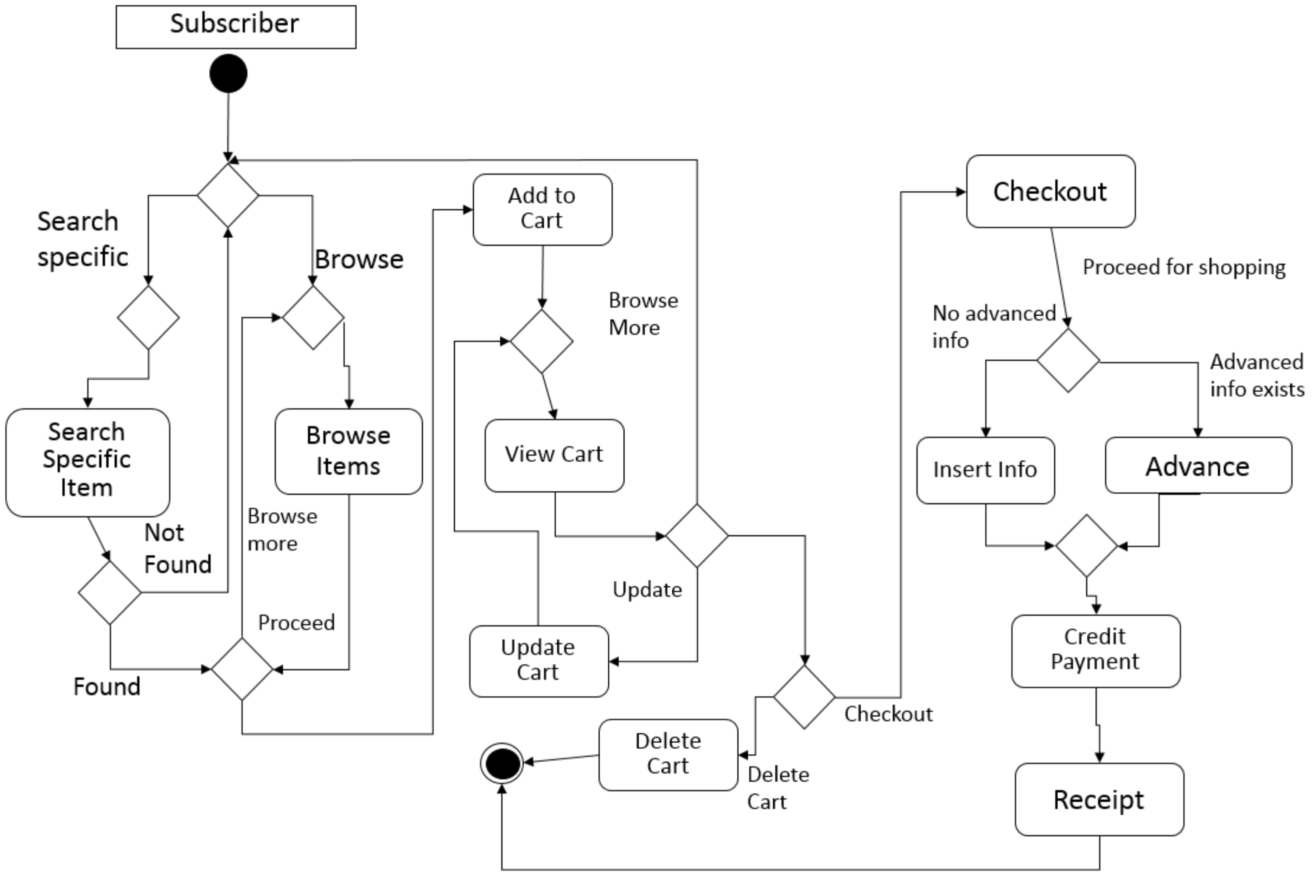


Fig. 2.4: Activity diagram for users purchasing products

2.6 Software requirements

For establishing the system we need these following software packages:

Web Server	**:**	**Apache (**Included in WAMP/XAMPP/LAMP)
DBMS	**:**	**MySQL (**Included in WAMP/XAMPP/LAMP)
Server-side Language	**:**	**PHP (**Included in WAMP/XAMPP/LAMP)
PHP Dependency Manager	**:**	**Composer** (https://www.getcomposer.org)
Version Control Software	**:**	**github**
IDE	**:**	Any text editor/IDE can be used as per user preference

2.7 Plugin Requirements

Stripe	**:**	API for payment (debit/credit card authenticity) validation.
Laravel Throttle	**:**	for limiting too many login attempts at once. (by Graham Campbell)
Ticketit	**:**	for user-staff communication (customer support).(by Ahmed M. Kordy)

2.8 Summary

This chapter discusses the basic structural and supplemental requirements for the application using software requirement analysis, sets the plot for a better understanding of the following chapters.

Chapter 3

Application Functionalities

The application offers diversified tasks for the users, different user groups carry out different tasks. From searching to purchasing products, from adding sales to removing products- a moderately wide range of tasks are performed by subscribers and office staff. In this section we try to mention the primary functionalities of the web application.

3.1 User Registration and Verification

The web application "One-Stop Shop" will let "guests" browse and search through the product details and add the products they want to purchase to the cart, but to access the cart and advance through the process, they have to login, if not- register on the website by verifying their e-mails.

The registration page will take users' First Name, Last Name, a unique E-Mail id, and confirmed password initially.

A confirmation e-mail is sent to the user-given e-mail address with a **randomly generated 40 character unique confirmation code.** The uniqueness is maintained by encrypting the current UNIX time using **SHA1** hash function.

User is kept inactive (is_active = 0), hence unable to carry out subscriber tasks until the e-mail verification is confirmed.

After the registration is completed, the user can access the cart, edit their profiles, contact the customer support and advance through the purchase options.

3.2 Throttled Login

Limiting too many login attempts at once and hence negating the chance of getting hacked by any kind of brute-force attacks is a primary security feature an application should have. To provide this security feature we used **Laravel Throttle** plugin to make sure the application can record the false-requesting IP and block it from making farther requests for a given amount of time to prevent potential attacks.

3.3 Product Searching

Users can search for products in a more convenient way through features like-

- ### Auto-Suggestion

Our application's search bar provide the users with auto-suggestions by matching the keywords to any product related info and shows it below the search bar. This piece of AJAX functionality is managed completely with plain and Javascript code. No jQuery or any other library was used for this part of the functionality to maintain simplicity.

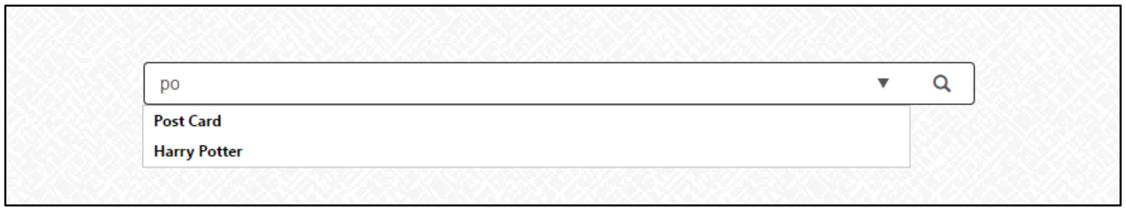


Fig. 3.1: Autocomplete suggestion

- **Sort Search Result**

Users can sort the search result to their likings, sorting can be done considering products'-

- ▪ Name (Ascending/Descending)
- ▪ Price (High to Low/ Low to High)
- ▪ Categories
- ▪ Brand Names

3.4 Cart Functionalities

One-Stop Shop's cart system provides convenient features for users to help product purchasing more interactive.

- **Add to Cart**

  Users add their to-be purchased products to the cart, either from the partial views (e.g., related products, recently viewed) or from the product details view. They can select their desired quantity to be added to the cart but not more than the stock permits. This stock limit validation is done through multiple stages, which is discussed later in the "Application Implementation" part of this paper.

- **Addition, Deduction& Removal On-The-Go**

  After a user adds a certain number of products to a cart, he/she might come to a decision to add another piece of the product to the cart, or deduct the product by one, or the choice might even be to remove the product from the cart. This fine-tuning feature is implemented on the cart to increase customer satisfaction.

- **Product Count and Total Price**

Throughout the whole browsing session of a user the total number of products added to the cart and accumulated price is shown beside the cart's button. This feature can be useful for users to get a better estimation of their purchase status.

- **Delete Cart**

  Cart deletion can be performed to discard a cart full of products and continue with a new one.

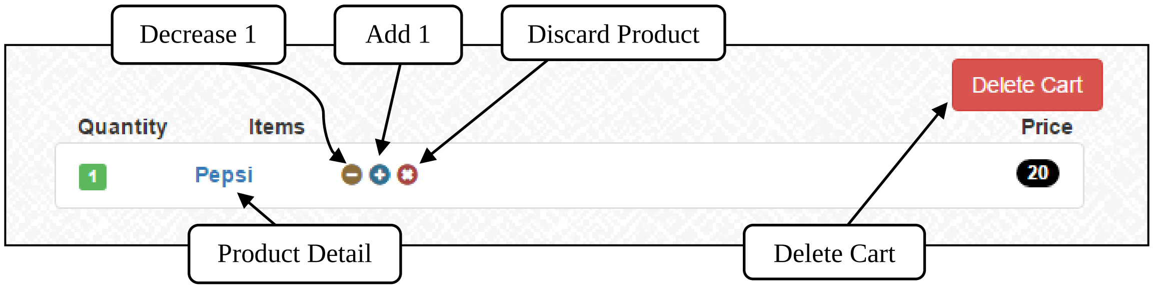
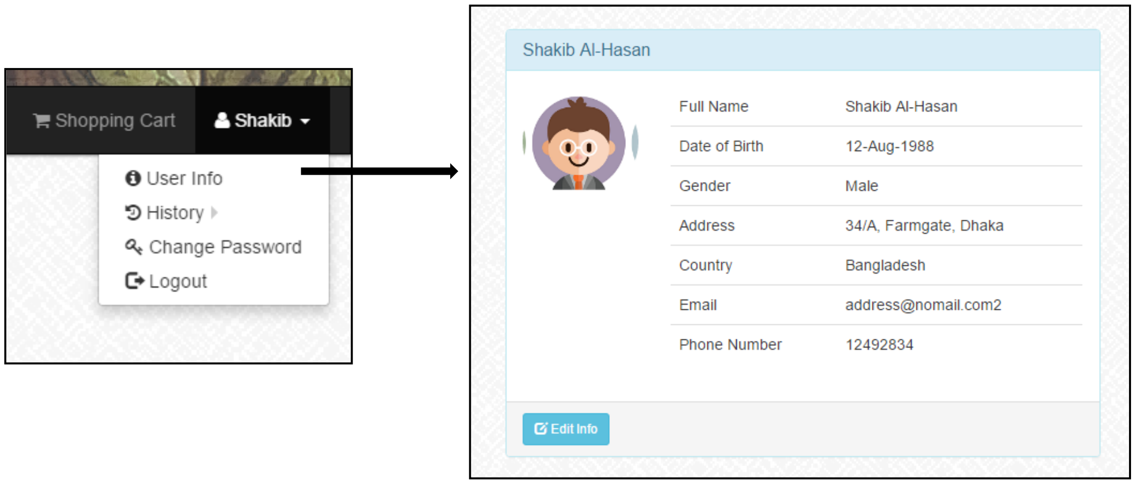


Fig. 3.2: Cart Functionalities

3.5 User Profile

Before a user can advance through to the checkout stage, the user's profile containing related advanced info(e.g., address, phone number, date of birth, etc.) must be submitted. The idea is to make sure of the availability of shipping address, phone number and such user contact-info beforehand, so the purchased products does not go to a wrong address. The profile info can later be viewed and edited from going to the "User Info" tab(/usersinfo url) on the dropdown menu.



Shakib Al-Hasan

Full Name	Shakib Al-Hasan
Date of Birth	12-Aug-1988
Gender	Male
Address	34/A, Farmgate, Dhaka
Country	Bangladesh
Email	address@nomail.com2
Phone Number	12492834

Fig. 3.3: View of a User's Profile

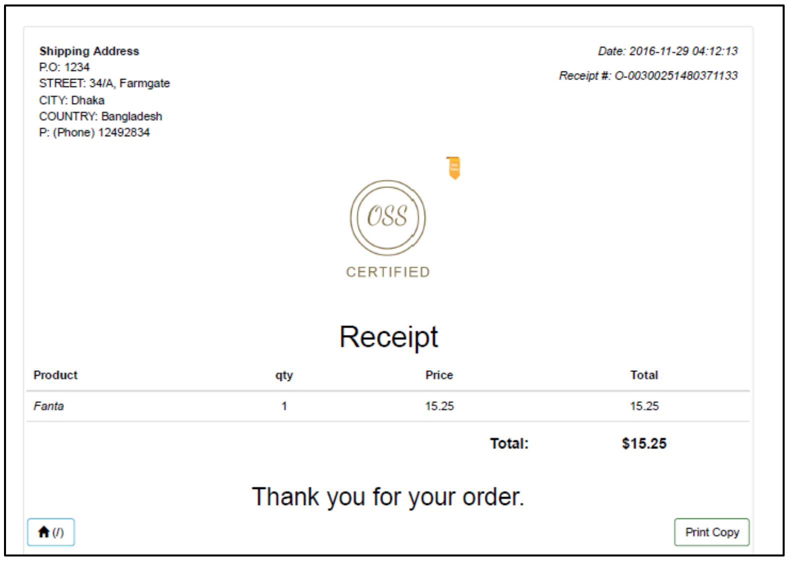
3.6 Checkout

Users can checkout only using valid payment cards. The validity and authentication is managed by **Stripe-** a well renowned, accredited API for handling payment procedures globally, and supporting user with by providing convenience.

3.7 Receipt

A printable receipt is given to the user after a purchase is successfully completed. These receipts can the viewed later by the user. The receipt id is kept unique by combining three different parts,

- First three digits are from user's id
- Next four digits come from the order's unique id
- The last ten digits are the current Unix time



Shipping Address
P.O: 1234
STREET: 34/A, Farmgate
CITY: Dhaka
COUNTRY: Bangladesh
P: (Phone) 12492834

Date: 2016-11-29 04:12:13
Receipt #: O-00300251480371133

Receipt

Product	qty	Price	Total
Fanta	1	15.25	15.25
		Total:	**$15.25**

Thank you for your order.

Fig. 3.4: A Purchase Receipt

Maintaining this format, an ‘O-’ prefix is set to identify this as a One-Stop Shop receipt id (e.g., O-00300251480371133).

3.8 User Feedback

Users can give their feedback on products, which can be removed later by the users at will.

3.9 Product Details

Product detail page shows a products detailed information about stock, price, brand, category etc. Stock interactively reduces/increases on the screen as the users add products to the cart or deletes them from the same. Product details page accommodates a number of features in a packed space.

- **Recently viewed products**

  This portion makes good use of cookies to keep and show the recently viewed products by a user.

3.10 Related Products

This part shows the related products to the current one in a sense that the users could potentially be interested to purchase them. (i.e., same category or frequently bought together (10)) These products suggestions are derived and fetched from accessing the cart data and finding the products those were sold together more frequently than the other products.

3.10.1 Algorithm

```
Foreach cart in carts
    If current_product_id exists in cart store the other ids in an array
End foreach
Foreach cart in carts
    If current_product_id exists in cart
        Foreach related_id
            If related_id exist in cart
                Store related_ids in array
            End if
        End foreach
    End if
End foreach
Count number of related_id occurrences
Sort from high to low
```

3.11 Customer Support

User support is managed by using an effective plug-in named “ticketIt”. This communicative feature lets customers enquire or complain about products, discuss refund issues or give feedbacks about the service. Pictures can be added by the users for better explanation of their problems.

- Users create ticket about their enquiry
- Staff respond to the open tickets and marks as completed

3.12 Sales Management

Sales can be set on products by One-Stop Shop staff (admin, managers). Products are added in simple click & pick manner negating any kind of difficulty. Product removal from sales is also managed in the same way. The most useful features about sales is that **sales are ended** (i.e., deleted) at the exact ending date **automatically**. We shall discuss about this functionality in the later part of our paper.

3.13 Product Management

- Products can be added, deleted and edited by admins and managers
- Categories and Brands are to be created by the staff
- An useful feature is **addition of new brands on-the-go** (non-redundant approach)

3.14 User Management

Admins can add, edit and activate users manually, and assign or change roles of existing users. User data manipulation is not accessible to the managers.

3.15 Summary

An overview of mentionable key functionalities of the application has been shown in the chapter. For better understanding, figures and algorithms have been shown in the chapter also. Following chapters discuss the implementation and technical procedures of these functionalities.

Chapter 4

Implementation Plan

Before we move on to the next section where the technical implementations of our project is discussed, we would like to discuss some of the mandatory to know key-terms first.

Both Laravel, and PHP are products of open-source projects. Hundreds of skillful contributors are constantly working on both of them to make them more competent, more secure. In that sense, these two make the perfect, powerful duo to be used to make powerful, robust web applications.

We used this open-source duo to build our project along with HTML, CSS & Javascript. We used **bootstrap 3,** a web-styling framework for front-end designing.

Apache serves as the local-host, **MySQL** is the **DB Management System. PHP** is the server-side language. A package manager **Node.js** is used to manage gulp elixir.

4.1 A Quick Glance at Laravel Framework

Laravel is an open-source new generation web framework that uses a vast number of advanced features to make web application development secure, and time-saving for developers. Implementation of **MVC** pattern gives the developers a level ground to start their app development in a guided sense.

4.1.1 Key Terms:

- **MVC:** MVC is abbreviation for **Model-View-Controller** pattern in software development. Laravel framework uses this highly efficient pattern in the process of web development.

- **ORM:** ORM is short form of **Object Relational Mapping**. ORM Helps make an entity with unsupported data-type into an "Object", which can be later manipulated using **OOP** (Object Oriented Programming) concept.

- **Eloquent:** Laravel's eloquent simplifies the task of information retrieval by using "Models" to access the respective data relations. Hence simple functions can be implemented on the

- **Migrations:** Migrations are used for creating, deleting, and modifying databases and relations. The key convenience of migrations is that, the databases can be rolled back and forth of its state (e.g., remove a column that has just been inserted, rename a column back to its previous name) during development process at ease. Large chunks of data can be "seeded" into the database with significantly less effort.

- **Middleware:** Middlewares work as passages through which view requests go through to the controller to be manipulated. In a sense a "Middleware" controls the view-controller communication.

- **Service Provider:** Service providers are the start-up entities those register event listeners, middleware, routes and other variable at boot level.

- **.blade:** Naming convention of Laravel views follows adding a ".blade.php" extension (e.g., index.blade.php) at the end of the view filename.

- **Route:** All the existing routes in the web application must be declared in the "web.php" file inside the "Route" folder.

4.2 Project Coordination

We managed the coordination of the tasks for our project by constantly keeping each-other updated about the current events. We kept logs about the to-do tasks regularly and separated the task of implementing different modules between us. The coordination was mostly done using online chat sessions and/or screen-sharing applications.

4.2.1Version controlling

We used github for our application's version controlling. Whenever a permanent change was done the files were pushed to the main respiratory of github where all our files were kept for better coordination.

4.2.2 Merging

Minor mergers were done by sharing particular files or through updating github respiratory. Major mergers were done however face-to-face under careful watch of our team.

4.3 Summary

Planning the implementation procedures and maintaining cooperation is a key task to be sorted out before going into project development. Proper cooperativeness leads to smooth mergers and easier issue-solving for a project.

Chapter 5

Technical Procedures

For the technical part of this project the issues we considered with utmost priority were-

i. Ease of use
ii. Secured server-side validation
iii. Redundancy free database

5.1 Ease of Use

Maintaining usability of an application may seem negligible if not much thought is given into the matter, but if we try to look at it carefully, it becomes pretty clear that, this is one of the, if not, the only significant quality an application should have. It is because of the fact that, applications are built for the users particularly. If they do not feel comfortable using the application, then the sole purpose of building one fails. We took careful notes, and tried to build the application as much easy to use as we could within our extent.

Few features those were included to support our goal-

- Auto-suggestion in search
- Search result sorting
- Consistent fail-safe pop-ups
- Effective redirection to proper routes

- User-staff communication (Customer Support)
- Automatic Sale Deletion (Scheduled Task)

Fail-safe manner of the interactions makes sure that the users navigate through their browsing sessions inherently.

5.1.1 Auto-suggestions

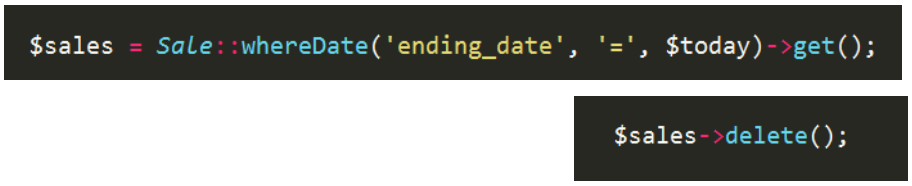
Suggestions appear below the search bar as users start typing to search for a product. AJAX functionality manages to send the request to the controller instantly as the letters are being typed, the controller searches for products with the matching keyword and returns the data to the view to be received by the AJAX code. This listening process activates as soon as a user selects the search bar.

5.1.2 Search Result Sorting

Search results are primarily shown to the user in an alphabetical order. But those can be sorted later according to the users' needs. Sorting by Price both high-to-low and low-to-high are two useful options. Sorting by ascending/descending alphabetical order and category, brand names are other options those are included for convenience.

5.1.3 Automatic Sale Deletion

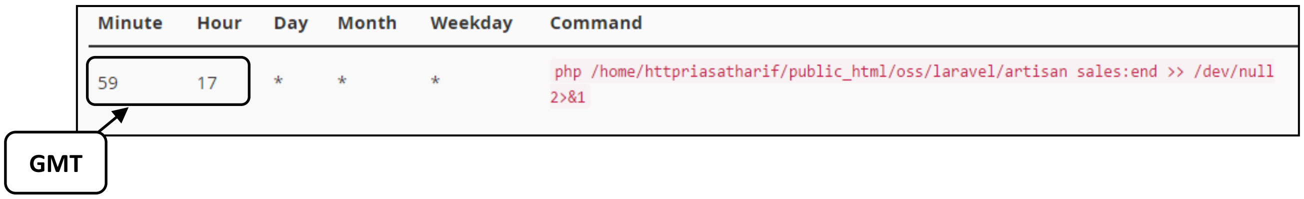
This feature is very useful for the office staff and certainly saves their time from doing another relatively less-important task. With the passing of the ending date-time of a sale, the sale is deleted automatically and the products are restored back to their original price. If there are multiple sales that are to be ended on exactly the same date-time, all of them would perfectly get deleted in time. Deleting sales manually is also an option that has been provided for the staff.



```
$sales = Sale::whereDate('ending_date', '=', $today)->get();
$sales->delete();
```

Fig. 5.1: Deleting sales on ending date

Server side cron-jobs are used for running the custom command on a scheduled time. Usually "Greenwich Mean Time" is used on servers for date-time of a scheduled task.



Minute	Hour	Day	Month	Weekday	Command
59	17	*	*	*	php /home/httpriasatharif/public_html/oss/laravel/artisan sales:end >> /dev/null 2>&1

Fig. 5.2: Task scheduling for custom command

5.2 Server-side Validations

Proper server side validations are performed before fetching any data from the database or storing into it. The significance of server side validation is that even if someone manages to inject HTML codes through to the server, the server-side validation will check if the request is a legal/ non-violating one before the request actually reaches the database. This way any kind of harm to the database can be prevented effectively.

5.2.1 During Registration

- During the registration process users are prompted to use unique e-mail addresses. The process of checking the uniqueness of the user given e-mail is performed through server-side validation. Inside the file "RegisterController.php"- the controller that manages the user registration process, checks the validity of the uniqueness of the given e-mail.(3) To make the request posting safe, a CSRF token is sent with the form request to the server.

- Password format and the correctness of the password in checked and confirmed in the controller.(3)

- Registering users are instantly assigned the "Subscriber" role to their account. (4)

- The confirmation code is generated by encrypting the current UNIX time and re-shuffling it before sending to the user's e-mail address. (4)

5.2.2In Cart Functionality

Users can conveniently add products to the cart and deduct or remove them at will. This process of addition and subtraction may seem very simple but in reality, even a tiny little loophole can break the database during this process.

5.2.2(a) Problematic Scenarios

If proper server-side validation were non-existent in a project, it does not take a genius to tell that the system would break any moment after it launches, even if it manages not to fail to launch in the first place. But for the sake of example, let us assume that, the server-side validation were not thought very thoroughly and were weak in implementation in the project. In that scenario, any tech-savvy user having intermediate knowledge about web can inject unreal and/or breakpoint values to manipulate the database and hence harming the transactions. I.e.,

- Create multiple cart images and send the requests altogether.
- Inject 0 to multiply with the total price and checking out with a huge amount of products

- Insert a garbage value to the cart quantity so that, on checkout it breaks the product stock in database

5.2.2(b) Prevention

To prevent these kind of failing scenarios firstly, we save our cart in the appointed session only. Users cannot use any other session to function as a cart. That is because the PHP sessions are cross-matched with a given id from the server, stored in the cookies, which is a hashed string and too hard to break; even it were to be broken, the session will fail to cross-match with the server and become invalid instantly. This process prevents the first problem from arising ever.

To prevent users from adding any invalid number of product that can dismantle the database, primarily the maximum number of products is set in the view file according to the product's available stock. The minimum number is set to 1 if the product is in stock. Both are set to 0 if the product in out of stock.

As we discussed before only this setup alone is not sufficient for preventing the damage from happening. So the server-side controller takes out the cart data and checks if any invalid amount of quantity has been requested. Attempts to insert 0 as product quantity is checked and restricted, attempts to add negative values are also discarded.(5)

One particular feature to be pointed here is that, though we are letting users add, remove products to/from the cart and also restricting them from inserting invalid values, the actual database is nearly untouched. That is, the actual stock of products does not get affected until the user checks-out the cart.

5.2.3 In Checkout

As the user proceeds forward to checkout, the controllers will look for users advanced information. If the info are not available, the user cannot go forward, the user must fill out the information form to continue the purchase session. If advance info pre-existed, the user can choose to either proceed instantly or edit the appropriate info to continue. This process is also validated server-side. After the order is placed, the current stock is checked accessing the database before forwarding, if the product has been reduced meanwhile, the user is redirected back or else the checkout process forwards and updates the database accordingly.(6)

There are quiet a few more places here and there inside the controllers to ensure the data validation, such as the ‘FeedbackController.php’ checks if the feedbacks shown on a product are of the current user, if they are not, they cannot be deleted by the current user. Another mentionable one is inside the ‘AddProductsController.php’, where the staff can choose to add a new brand for the product on the go and it checks server-side if the requested brand exists or not (discussed later in ‘Redundancy Free Database’).Concisely, we considered any visible/potential loopholes as threats to the system and put efforts within our extent to remove them.

5.3 Redundancy Free Database

Throughout the whole project we carefully tried to maintain the non-redundant approach for our database system. From storing users’ advanced info to not updating the product stock until the checkout is completed, less-to-no redundancy was one of our top priorities.

5.3.1 In Product Management

During addition of new products to the database (by office staff) there might come a scenario where a product of the same name and same brand exists but the staff failed to notice that beforehand. In such case, adding a product having same name results in data redundancy.

To prevent this kind of potential record duplication, the application checks if a product having the same name and the same brand pre-exists in the database. If a matching product is pre-existent, the stock number for the product is increased adding new stock quantity to the previous one and price for the product keeps the lower amount as the updated price.

Another quite similar a scenario arises during addition of a new product, when a new brand name for the product has to be entered but the database does not contain the brand name. In such case the staff can simply insert the name of the brand instead of an existing one. Now, the controller receives the request, checks for a match if the brand pre-exists. If it does, the **existing brand_id** is assigned to the product. If the brand is a new one, the controllers inserts it into the database giving it a new id and saves it after assigning it to the product.(7) This way any potential redundancy arising from duplicate data existence is avoided.

Note that, product's 'Categories' and 'Brands' are kept in different tables for the convenience of managing them in advanced fashion (e.g., use in navigation bars, sub-categorize), or manipulating later in time by the staff. This keeps the option of adding extra attributes to categories/brands open for future use.

5.3.2In Advanced Info for Users

For the advanced user info, we reduced redundancy by keeping the table corresponding to advanced user info ('advanceds') separated from the main users table. This way the users do not necessarily have to insert their advanced info until they try to purchase something; so the database and the table is more likely kept relatively smaller. This approach also helps to minimize the database access cost when a user performs regular tasks like adding feedbacks, contacting customer support, watching histories etc., where the only entity the user needs is the ID to get access to those task initially. Had the other data been in the same table, more cost would have caused due to fetching bigger data chunks. On deletion of any user the advanced info also gets deleted.

5.3.3In Receipts and Orders

This is another section where we maintained less redundancy according to the scenario of the two entities. An 'Order' will keep the ordering user's id (user_id), the items bought (cart), the payment confirmation id (payment_id) and the status of the order (i.e., pending, delivered, returned) to keep track of the actual order.
The 'Receipt' will keep the user's id (user_id), the order's id (order_id), shipping address and phone number of the user. This way the products related to the receipt will be fetched from the order to show on view. One important point to notice here is the inclusion of shipping address and phone number in the 'receipts' table. Primarily it may seem that

these two columns are used here redundantly, when it can simply fetch data from the user’s advanced info relation. But if a user moves from the place he/she ordered from to another place and updates the current address in the ‘advanced’ table, the actual address where the shipping was made gets lost. Same happens for the phone number too. That is the reason why these two columns were kept with receipts intentionally. Any other visible redundancy were avoided.

Redundancy free database is a highly desirable characteristic for an application. We tried to keep our database as less redundant as possible for a better system.

5.4 Summary

Key background features and their implementation procedures have been discussed in the chapter. Ease-of-use, proper validation and the process of non-redundancy go hand in hand in an interactive project. Implementing them and making sure they all work together is a delicate task.

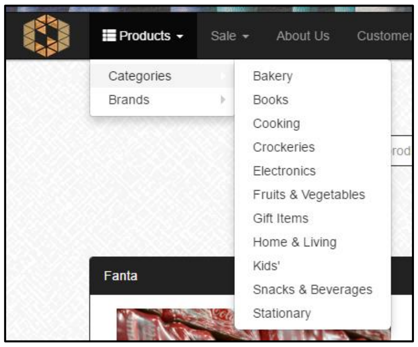
Chapter 6

View Optimization

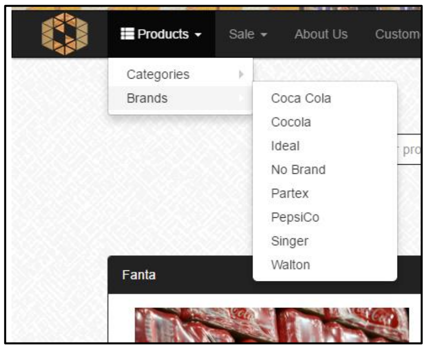
The UI (user interfaces), or ‘View’ as termed in MVC pattern is a significantly vital part of an application. That is because the users would likely have no interest in knowing what kind of features are used in the application, how fail-safe the application is or how less redundant the application’s database is. If the users do not like the view or if they feel uncomfortable using the app they would most likely lose their interest on the application. In this section we briefly mention a few of the procedures we undertook to optimize the application’s views.

6.1 Using Service Providers

Service providers helps load variables, routes, event listeners and such entities during the start-up, before any views or controllers are loaded. We passed our frequently used models such as categories, brands to our main layout view using the ‘AppServiceProvide.php’ at the application’s booting time. The categories and brands are fetched from the database and sent to the main layout (app.blade.php) page during program initiation. This way, we maintain non-redundancy in our views too.



(a)



(b)

Fig. 6.1: Categories (a) and Brands (b) on navigation bar fetched from database.

6.2 View Stock Availability

We managed to show the users the stock availability in a virtual manner. Initially the stock is shown to the users according to the actual product stock that is derived from the database. If the product has stock, a green colored 'In Stock' text is shown(a) beside the product's available quantity; if not, a red texted 'No Stock' is shown (b) in the same place.

(a)

(b)

(c)

Fig. 6.2: Stock availability indicator

Now, if a user starts adding the product to the cart, the product's available quantity keeps reducing and shown accordingly to the user, and as long as the available quantity does not reach 0 the 'In Stock' text is kept shown. If the user adds all the available stock to the cart the text then shows 'Limit Reached' in red colored text(c). All of these mentioned functionalities are carried out without affecting the database right away(9)- as we discussed in the earlier section, which adds a sense of virtuality to the view manipulation.

6.3 Search Bar

The search bar is built as a partial layout with its full functionality for the web-app. This enables the search-bar to be extended anywhere in any page as an external module.
To add the search bar to view using a single line is (e.g., @extends(‘layouts.search’)) enough. This way, we avoided using the search bar on places where it is not necessary (e.g., register, profile, receipt, checkout etc. views).

6.4 Summary

Laravel provides developers with a subtle view management system to avoid any unnecessary work procedure. Idea of extending a main layout, and/or partial layouts in views is an effective feature for reducing overhead; hence increasing the throughput quality.

Chapter 7

Conclusion

7.1 Learnings

There are a few learnings worth mentioning those we learned working on the project. These realizations would surely help if kept in mind for future projects.

- Building a web application is a continuous process
- Need for new modules will always persist
- Bug-free modules are products of countless re-testing
- Careful time-management is very crucial
- Regular team meeting during development is a must
- Maintaining good coding practices is necessary for bigger projects

7.2 Conclusion

In the project we explored and demonstrated a few advanced, user-friendly features to carry out the regular tasks. Working on this project, we came to a better understanding of how e-commerce and engineering techniques can work hand in hand to bring more advancement in the sector. We hope people interested in this sector would get a clear view on the workflow of these kind of businesses and take that extra step to explore the sector for the purpose of improvement.

7.3 Future Work

- A better looking UI
- Product rating feature
- Sub-categorize products
- Home delivery & tracking system
- Monolog
- SMS verification system (in progress)

Appendix

Code Snippets:

1.

```
function show(str) {
    var dataList = document.getElementById('json-datalist');
    var input = document.getElementById('tbx');
    if (str.length == 0) {
        input.placeholder="Search for products..";
        return;
        } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 &&
            this.status == 200) {
            //remove datalist element
            if (dataList.hasChildNodes()) {
            // It has at least one
                dataList.parentNode.removeChild(option);
                // window.alert('del');
            }
                var found = this.responseText;
                //JSON.parse
                var parsed = JSON.parse(found);
            var cnt = dataList.childElementCount;

            parsed.forEach(function(item) {
            // Create a new <option> element.
            var option = document.createElement('option');
            // Set the value using the item in the JSON array.
            // window.alert('ok');
            option.value = item;
            // Add the <option> element to the <datalist>.
            if (cnt<=6){
            dataList.appendChild(option);
            }
            });
            }
        }
        xmlhttp.open("GET", "/getq?q=" + str, true);
        xmlhttp.send();
    }
}
```

2.

```
public function handle()
            {
       $today= Carbon::today()->toDateString();
       $sales = Sale::whereDate('ending_date', '=', $today)->get();
       // $sales = Sale::where('id', '=', 1)->get();
       foreach ($sales as $sale) {

           $products = $sale->products;

               foreach ($products as $onsale) {
                       $onsale['sale_id'] = NULL ;
           $onsale->save();}
           $sale->delete(); }
```

3.

```
protected function validator(array $data)
    {
        return Validator::make($data, [
            'first_name' => 'required|max:255',
            'last_name' => 'required|max:255',
            'email' => 'required|email|max:255|unique:users',
            'password' => 'required|min:6|confirmed',
        ]);
    }
```

4.

```
protected function create(array $data)
    {
        return User::create([
            'role_id' => '3',
            'first_name' => $data['first_name'],
            'last_name' => $data['last_name'],
            'email' => $data['email'],
            'conf_code' => str_shuffle(sha1(time())),
            'password' => bcrypt($data['password']),

        ]);
    }
```

5.

```
// Cart Input validation
        //no old cart
        if($request['qty'] && !$request['qty']<=0){
                if($product->stock - $request['qty']<0){
                            return redirect()->back();
                }
        }
//old cart exists
        if($oldcart){
        foreach($oldcart['items'] as $old){
           if($old['item'] ['id']==$id){

            if($request['qty']  && !$request['qty']<=0){
                if($product->stock - $old['qty'] - $request['qty']<0){
               return redirect()->back();
                    }
                }
                 else if($product->stock - $old['qty']<=0){
                    return redirect()->back();
                    }
                }
            }
        }
        //Input Fail-Safe
```

6.

```
if($product->stock - $bought['qty'] >=0 ){
            $product->stock -= $bought['qty'];
            $product->save();
            // dd($product);
            }else{
return
redirect()->route('product.detail',$product->id);
            }
```

7.

```
$exist = Brand::where('brand',$brand)->first();
            if($exist){
                $input['brand_id'] = $exist->id;
            }else{
                $brand = Brand::create(['brand'=>$brand]);
                $input['brand_id'] = $brand->id;
}
…………//more codes//………
  Product::create($input);
```

8.

```
public function boot()
   {
       view()->composer('layouts.app',function($view){
                $view->with('cat',Category::all()->sortBy('name'))->with('bnd',Brand::all()-
>sortBy('brand'));
       });
       view()->composer('layouts.authapp',function($view){
          $view->with('cat',Category::all()->sortBy('name'))->with('bnd',Brand::all()-
>sortBy('brand'));
       });
   }
```

9.

```
if(Session::has('cart')){
          $cart = Session::get('cart');
      foreach ($cart['items'] as $item) {
             if($item['item']->id == $id){
                $exists['max'] = $product->stock - $item['qty'];
                $exists['qty'] = $item['qty'];
return view('products.detail',compact(['product', 'exists']));
             }
          }
       }
```

10.

```
$count = array_count_values($timesTogether);
arsort($count,SORT_NUMERIC);
```

Short Forms& Acronyms

MVC	**:**	Model-View-Controller

ORM	**:**	Object Relational Mapping
API	**:**	Application Programming Interface
OOP	**:**	Object Oriented Programming
SRS	**:**	System Requirements Specification
UI	**:**	User Interface
DBMS	**:**	Database Management Systems
IDE	**:**	Integrated Development Environment
SHA1	**:**	Secure Hash Algorithm 1

References:

[01] https://www.w3schools.com
[02] https://en.wikipedia.org/wiki/Laravel/
[03]https://laracasts.com/
[04] http://stackoverflow.com/
[05] http://www.laravel.io/
[06] http://www.tutorialspoint.com/laravel/
[07] http://www.youtube.com/
[08] http://bootsnipp.com/
[09] https://en.wikipedia.org/wiki/Design_specification/
[10] http://www.getbootstrap.com/
[11] https://github.com/thekordy/ticketit/
[12] https://github.com/GrahamCampbell/Laravel-Throttle
[13] https://stripe.com/