Ecommerce web app for grocery shopping

Software Requirements Specification

Version 1.5

11/08/2020

Deepak Malempati

Omkar Abhiteja Badda

Gangadhar Yerramsetti

[Sushmita Rudra](https://nwmissouri.instructure.com/groups/25657/users/29420)

[Kamal Reddy Donthireddy](https://nwmissouri.instructure.com/groups/25657/users/29433)

Submitted in partial fulfilment

Of the requirements of

CSIS 44-691 Graduate Directed Project 1

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 06/09/2020 | 1.0 | Deepak Malempati | Added requirements R1 through R3 |
| 06/09/2020 | 1.1 | Gangadhar Yerramsetti | Added requirements R4 through R6 |
| 06/09/2020 | 1.2 | Kamal Reddy Donthireddy | Added requirements R7 through R9 |
| 06/09/2020 | 1.3 | Sushmita Rudra | Added requirements R10 through R12 |
| 06/09/2020 | 1.4 | Abhiteja Badda | Added requirements R13 through R15 |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

**Table of Contents Page Number**

1. Introduction
   1. Purpose
   2. Scope
   3. Definitions, Acronyms, and Abbreviations
   4. References
   5. Overview
2. General Description
   1. Product Perspective
   2. Product Functions
   3. User Characteristics
   4. General Constraints
   5. Assumptions and Dependencies
3. Specific Requirements
   1. External Interface Requirements
      1. User Interfaces
      2. Hardware Interfaces
      3. Software Interfaces
      4. Communications Interface
   2. Functional Requirements
   3. Use Cases
   4. Class/Objects
   5. Non-Functional Requirements

3.5.1. Performance

3.5.2. Reliability

* + 1. Availability
    2. Security
    3. Portability
  1. Inverse Requirements
  2. Design Constraints
  3. Logical Database Requirements
  4. Other Requirements
  5. Prototypes (for complete project)
  6. Use Case Diagrams

1. Design

4.1. ER diagram

4.2. GUI

1. Analysis Models

4.1. Data Flow Diagram

4.2. Sequence Diagram

6. Technical Manual

7. End user Manual

## CHAPTER 1

**INTRODUCTION**

## 1.1 Purpose

The purpose of this project is to develop an e-commerce web application to users for groceries shopping. Due to global pandemic, many customers do not prefer to buy groceries by visiting the store. This web application will act as a virtual store which enables the customers to buy groceries online. The users can choose to either pick up the order from the store or get it delivered to their homes.

## 1.2 Scope

To develop a web application that saves a lot of time of customers that they would otherwise spend on visiting the grocery store and roam from aisle to aisle in search of the products. Users can add products listed under categories to their cart and checkout by either opting for delivery or pick up from the store. A chat box to aid the users with any required information is handled by the owner.

## 1.3 Definitions, Acronyms, and Abbreviations

OSS - Online Shopping System (for Grocery shopping).

SRS - Software Requirement Specification.

GUI - Graphical User Interface.

Stack-holder - The person who will participate in system or site Ex: Customer, Admin, Visitors, etc.

CSS - Cascading Style Sheet.

HTML - Hypertext Markup Language.

TCP/IP – Transmission Control Protocol/Internet Protocol.

HTTPS – Secure Hypertext Transfer Protocol.

SSL – Secure Sockets Layer.

## 1.4 References

## 1.5 Overview

The main idea of the web application is to provide customers a chance to shop groceries virtually using internet form any place and permits customers to purchase the grocery items and snacks of their own from the store. The data or information related to the products are stored on database at the server side which is at store. This server help to process the entire action performed by the customer and the selected items are shipped to the address given by customers. The web application was planned into two modules initially is for the customers who wish to purchase the grocery items and snacks, second is for the admins who keeps up and refreshes the data relating to the products, those of the customers and the admin maintains the database. Moreover the products which are selected by customer are added to the shopping cart and once the order process is done, it will be added to the list of active orders in admin portal.

Also, we provided customers with live chat system which will be part of help & assistance page and customer can track their order which will be shown in history of orders in customer account.

The rest of the SRS is organized as follows:

* Chapter2 is the General description of the project.
* Chapter3 cites the specific requirements of the project.
* Chapter4 displays the Design of the project.
* Chapter5 presents Analysis Models of the project.

## CHAPTER 2

## GENERAL DESCRIPTION

## 2.1 Product Perspective

All the stakeholders are responsible for product success. All the requirements needs to be fulfilled to achieve the product success. The requirements define the product’s functional behavior. It is important to define product requirements to reach product’s perspective. The product be available to all users and use client server model.

## 2.2 Product Functions

The final delivery of project should cover all the requirements and work as expected. Each functionality should be secure and all the user data should be strongly secured. The product should function in such a way that any functionality should not lead to crash of webpage.

## 2.3 User Characteristics

Application has users of two types, Customer and Shop owner or admin. Both of them need to have an account to access our application. Customers can only access the products which are in display and information related only to their account, whereas admin can access information related to every customer like the products which they purchased. Customers have the privilege of resetting their passwords. Admin can view the orders placed and have a track on them. In the user profile activity, customer details will be present where they can verify their orders, look for previous orders, and even edit their personal information.

## 2.4 General Constraints

We are a team of 5 members and have divided the work equally among ourselves to build this application. Our main agenda is to develop the application and make it ready for use in six months. To achieve that we have set up several deadlines with the latest one being on June 20th (A day before our Mid-term presentation). As our team members are located at several different places we are constantly conducting our meetings over zoom to verify that we all are going on the right path.

## 2.5 Assumptions and Dependencies

The progress and success of the project depends on each team member actively responding to the requirements and changes during the development stage. Tasks are divided and assigned under the assumption that assigned member is capable of delivering the task on time, else any instance requiring additional support from the team members shall be brought into discussion during team connects.

The application is available only for web. Users can check their previous orders, and can order the same items. Users can save their preferred payment methods.

## CHAPTER 3

## SPECIFIC REQUIREMENTS

## 3.1 External Interface Requirements

**3.1.1 User Interfaces**

* Login and signup
* Homepage
* Products page
* Checkout page
* Profile page
* Order status page
* Live chat

**3.1.2 Hardware Interfaces**

* Desktop or a laptop with 8GB RAM and 100GB disk space

**3.1.3 Software Interfaces**

* Operating System: Windows (Vista/7 or above)
* Web Browser: IE 10 or above, Mozilla FF 31 and above or Google Chrome
* Database used: Google Firebase
* Drivers: Java Runtime Environment
* Integrated Development Environment: Eclipse J2EE or Apache Tomcat, Visual Studio, HTML, CSS and JavaScript.

**3.1.4 Communications interfaces**

* All the pages in the application communicate with each other by passing data.
* Users can connect with system using a browser and login to buy grocery.

## 3.2 Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Requirement name** | **Description** |
| R1 | Login and registration for customer | Every user must able to register and login using registration details. User must be able to reset the password |
| R2 | Homepage with list of categories | An attractive homepage that display list of categories available in store. And display items on demand/offers |
| R3 | User profile management | A simple user profile with his/her account details and able to update those anytime they wish to. Functionality to logout and view the items in cart. User should be able to view the previous orders as well. |
| R4 | Products display | All users must be able to view all the products in chosen category. Each product should show price and quantity |
| R5 | Cart management | User must be able to view all the items added to the card along with the quantities and price of individual item. User can able to update cart anytime |
| R6 | Payment | This page should allow the user to input billing address and the means to select payment type and proceed with payment. |
| R7 | Checkout | This page should present information of order and the order confirmation, delivery or pick up slot details. |
| R8 | Help & Assistance with email and textbox | Help and assistance where user can enter information and a ticket ID will be generated and assigned to user |
| R9 | live chat | This chat box provides guide for site, how to order and contact details of the owner with live chat. |
| R10 | Order Management | This page helps customer to track their order at any time to  avoid miss communication between customer and owner. |
| R11 | Search bar with product search feature | Every user must be able to search products available in store using a search bar. |
| R12 | Product filters and sort option | User must have option to filter products based on price , in stock items, quantity |
| R13 | Check order history | User should be able to view order history in his account. |

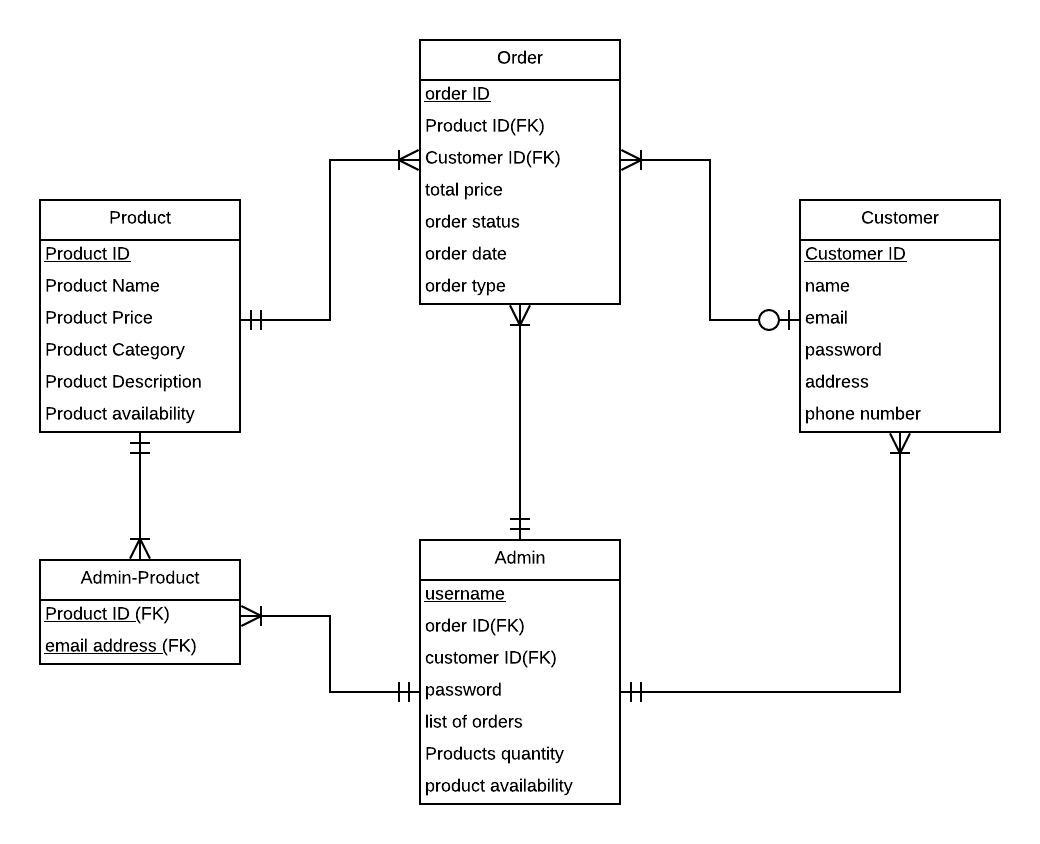
**Admin page requirements:**

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Requirement name** | **Description** |
| R14 | Login for shop owner/admin | Store owner/admin must be able to login to admin panel |
| R15 | Update in stock items/add new items into inventory by admin | Owner should have admin permissions and will be able to update the in stock items regularly and remove the item if it goes out of stock. |
| R16 | View active orders in admin account | Admin/owner should be able to view the list of new orders in his account and process them based on type of order. |
| R17 | Admin side customer support/chat functionality | Admin/owner will also have option to chat with customers if any complaints are raised through live chat option. |

## CHAPTER 4

## DESIGN

## 4.1 ER Diagram



We have a total of 5 entities namely Order, Product, Customer, Admin, and an associative entity named Admin Product. Customer can be considered as a user while Admin acts as the seller. The order table has a ternary relationship with the product, customer, and admin tables.

The product table has 6 attributes of which Product ID is the primary key and there are no other foreign keys in this entity. The customer table even has 6 attributes with customer ID being the primary key without any other foreign keys.

The order table has 8 attributes with order id being the primary key while product id, customer id, and admin user name being foreign keys from a product, customer, and admin entities respectively. Admin table has 7 attributes with User name being the primary key and customer ID, product ID being the foreign keys. Admin Product is an associative entity with product ID, User names as primary keys.

**Relationships:**

Product: Order - 1:M

Customer: Order – 1:M

Admin: Order – 1: M

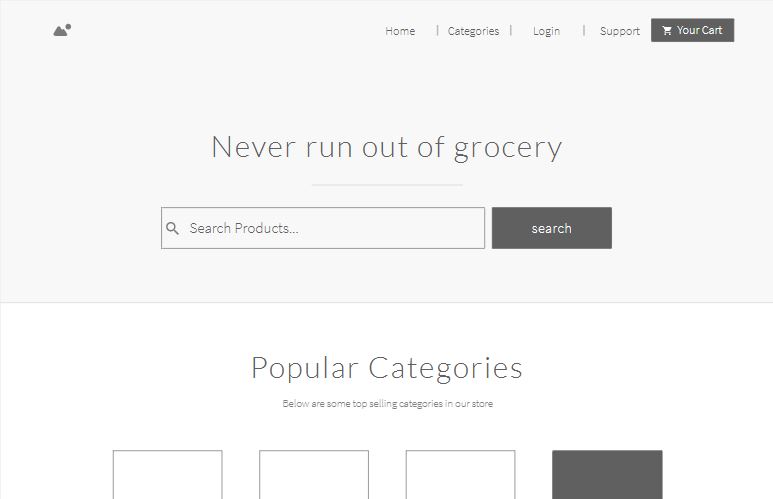
Admin: Customer – 1: M

Admin: Admin Product – 1: M

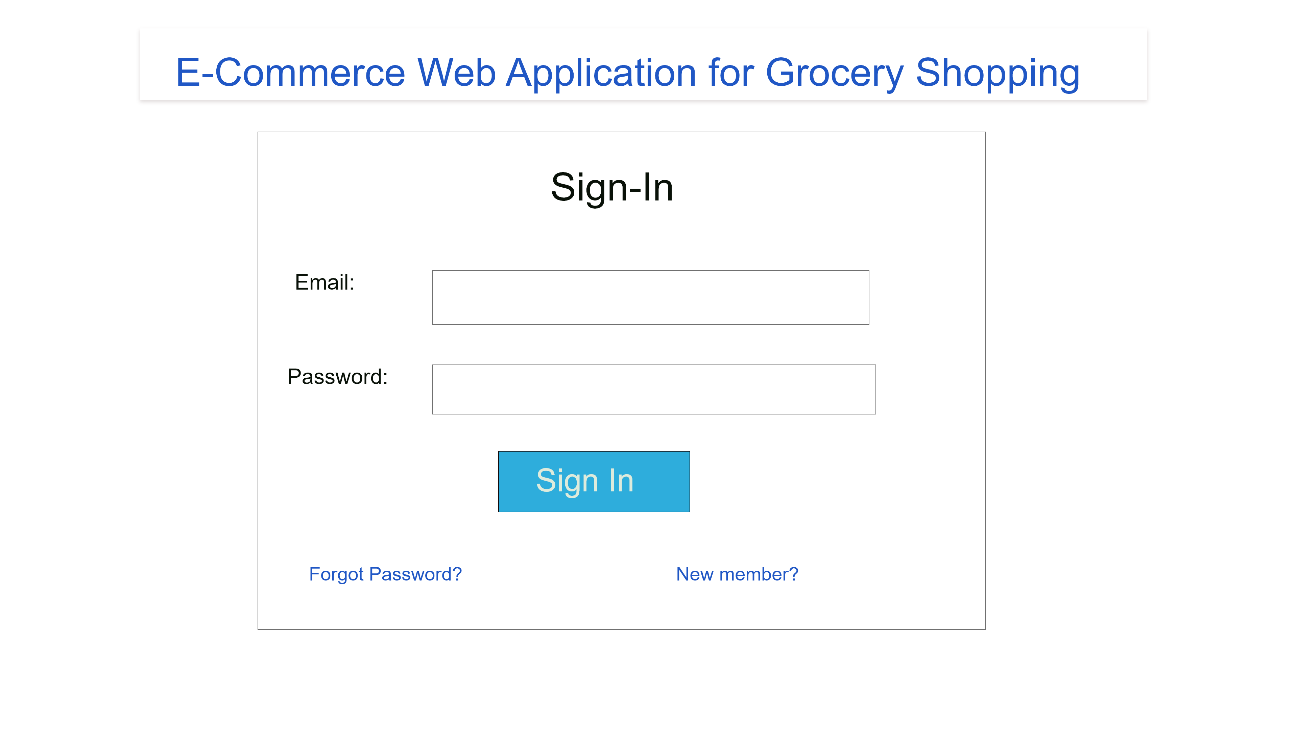
Product: Admin Product – 1: M

## 4.2 GUI

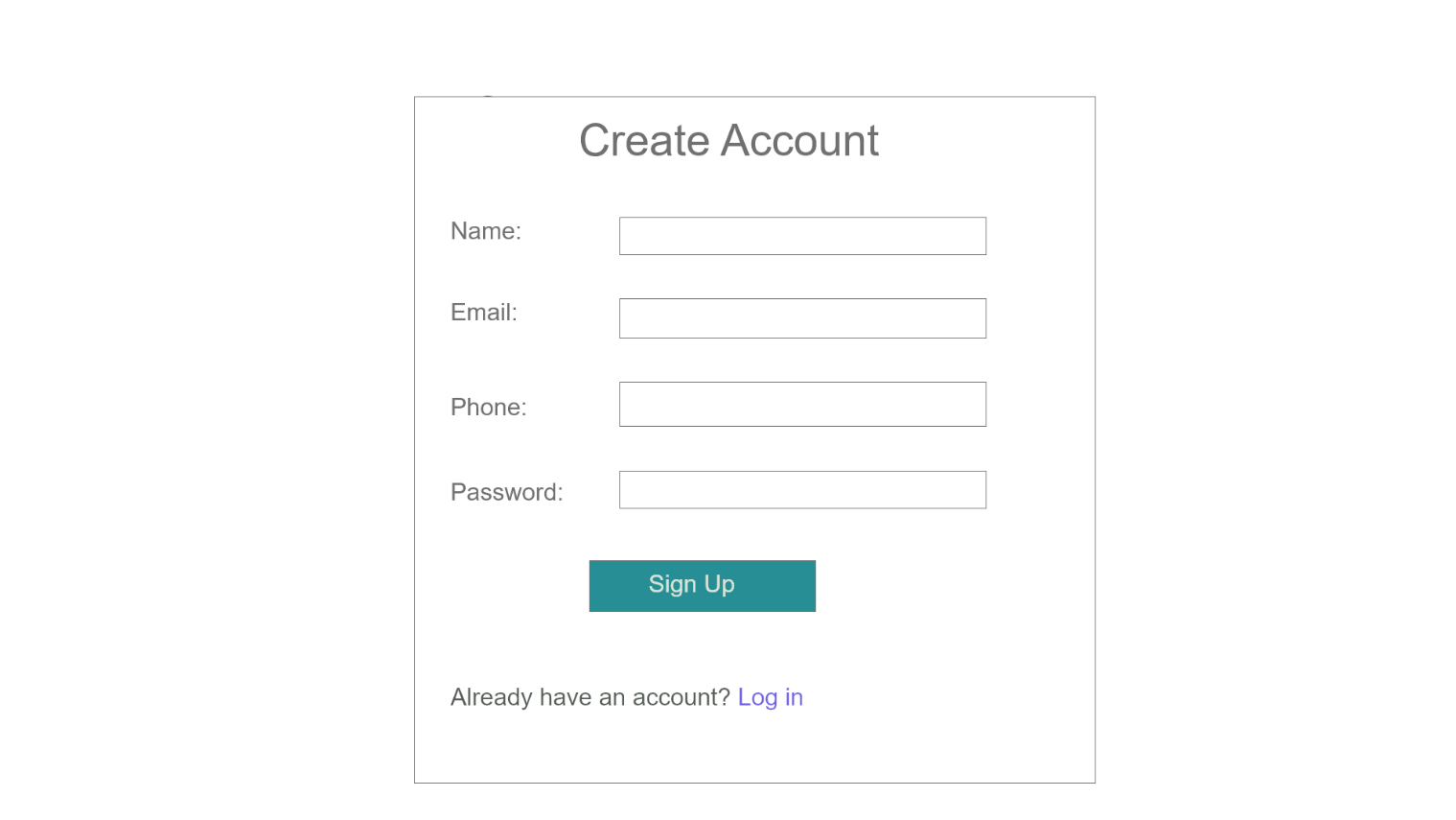
**Home Page:**



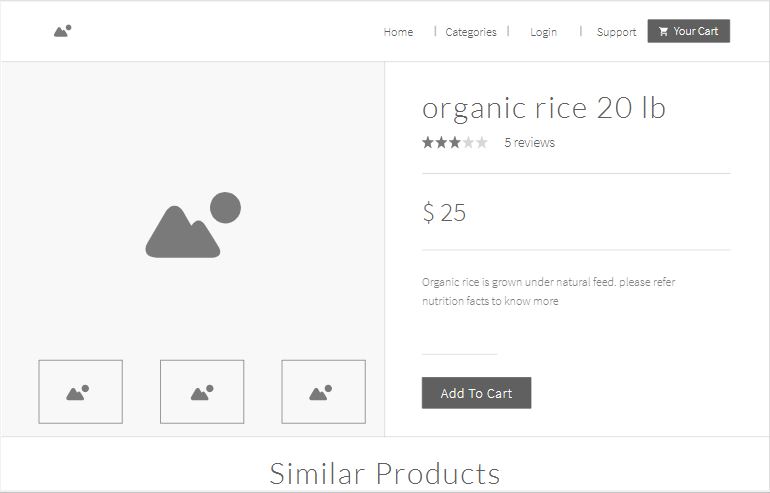
**Login Page:**

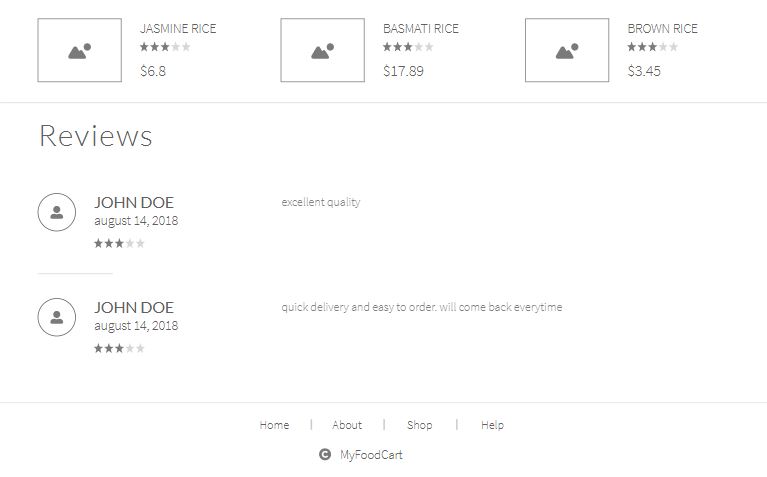


**Sign Up page:**

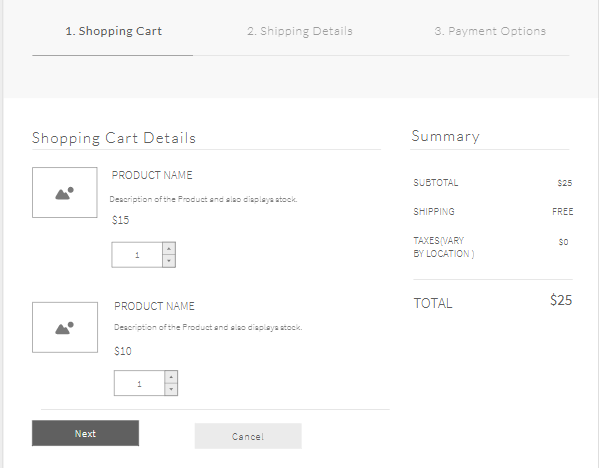


**Products display page:**

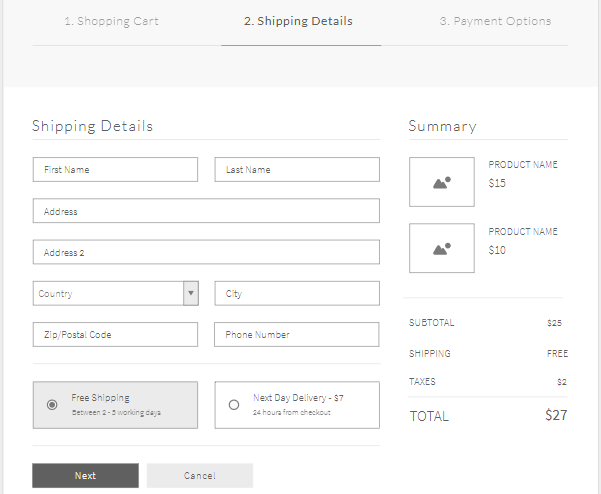




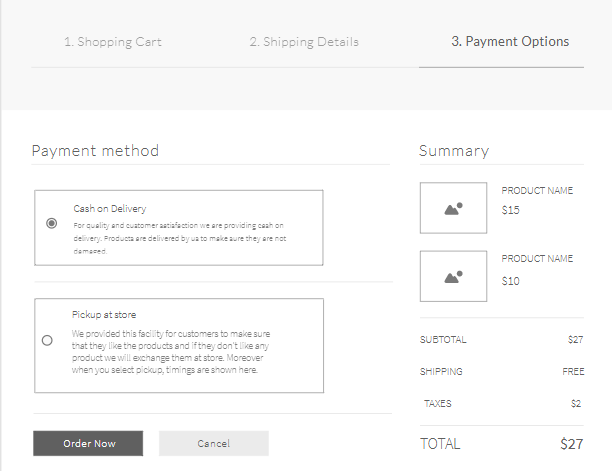
**Shopping Cart:**



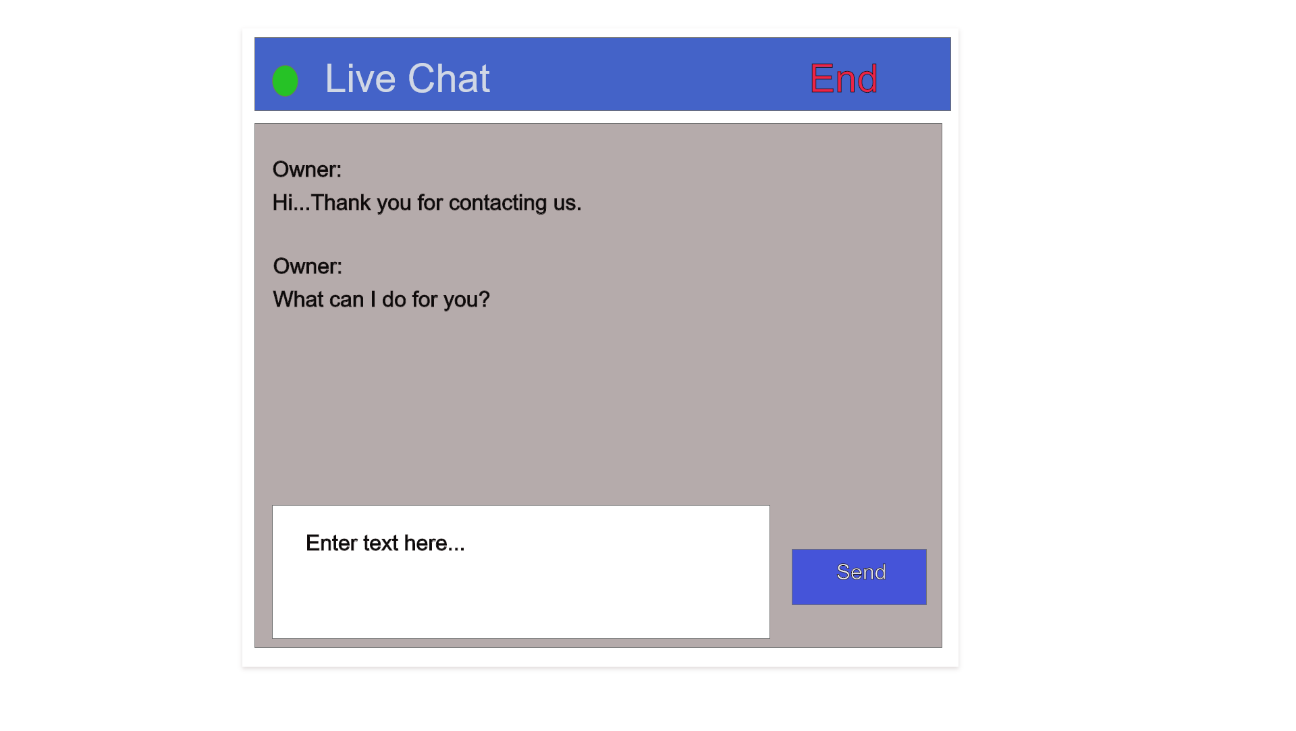
**Shipping details:**



**Payment:**

****

**Online Chat:**



## CHAPTER 6

## TECHNICAL MANUAL

(Team Member: Deepak Malempati)

1. **Admin Products Management:**

Admin products management is built on Angular and Type Script. Angular works as frontend framework and Typescript is used to write backend logic for storing a retrieving data. Admin products management consists of a form where admin can add new product into database. Firebase is used to store the list of products. At any time, admin is able to update the existing products in the database and delete the unwanted products from the store database. Firebase acts as storage service which stores data in JSON format. Firebase Storage API is integrated to the application.

**Sample code for Angular Form:**

|  |
| --- |
| <form #f="ngForm" (ngSubmit)="save(f.value)">  <div class="form-group">  <label for="title">Title</label>  <input #title="ngModel" [(ngModel)]="product.title" name="title" id="title" type="text" class="form-control" required>  <div class="alert alert-danger" \*ngIf="title.touched && title.invalid">  Title is required.  </div>  </div>  <div class="form-group">  <label for="title">Description</label>  <input #title="ngModel" [(ngModel)]="product.description" name="description” id="description" type="text" class="form-control" required>  <div class="alert alert-danger" \*ngIf="title.touched && title.invalid">  description is required.  </div>  <button [disabled]="!f.valid" class="btn btn-primary">Save</button>  <button \*ngIf="id" type="button" (click)="delete()" class="btn btn-danger">Delete</button>  </form> |

All the data entered in the form is stored and when admin clicks on update button, which call the update function to make the changes to existing products. Similarly, Delete function is used to delete the product from database. Router defines the path to navigate within the admin management of products. When admin delete a product, it is deleted in the database based on the id of the product

**Sample Backend code for admin product management:**

|  |
| --- |
| @Component({  selector: 'app-product-form',  templateUrl: './product-form.component.html',  styleUrls: ['./product-form.component.css']  })  save(product) {  if (this.id) this.productService.update(this.id, product);  else this.productService.create(product);    this.router.navigate(['/admin/products']);  }  delete() {  if (!confirm('Are you sure you want to delete this product?')) return;    this.productService.delete(this.id);  this.router.navigate(['/admin/products']);  } |

1. **Search Functionality:**

Product search is one of the core functionality of the application. Any user can search the products based on title or description. Searching of products in homepage works based on the products stored in the database by admin. All the products are taken in to the array for searching. Each time user enter a letter or string, search happens within the array of products and matching results are displayed on the home screen. Search functionality works efficiently when description of product is written accurately.

**Sample code for Product search:**

|  |
| --- |
| export class ProductsComponent implements OnInit {  products: Product[] = [];  filteredProducts: Product[] = [];  search(query: string){  console.log(query);  this.filteredProducts = (query) ?  this.products.filter(p => p.description.toLowerCase().includes(query.toLowerCase())) :  this.products;    } |

(Team Member: Gangadhar Yerramsetti)

**III. User Profile:**

The user profile section is build as a component using angular for front end and TypeScript as back end. The html code of the user profile is rendered into the main section of the application and is aligned with the nav-bar.

**User profile html content:**

|  |
| --- |
| <h2>My Profile</h2>  <div class="col-4 mx-auto">  <div class="card" style="width: 18rem">  <img  class="card-img-top"  src="../../../../assets/img/userProfilePicture.png"  alt="Logo"  />  <div class="card-body" \*ngIf="this.authService.user$ | async as user">  <h5 class="card-title">{{ user.displayName }}</h5>  <p class="card-text">  {{ user.email }}  </p>  </div>  </div>  <div class="mx-auto mt-3">  <a  class="btn btn-outline-danger my-2 my-sm-0 mr-5"  [routerLink]="['/my/orders']"  >View your orders</a  >  </div>  </div> |

The data here in this page is extracted from the firebase database. The code to extract data from the database is written in the “myprofile.component.ts” TypeScript file.

**User Profile Component Section:**

|  |
| --- |
| import { Component, OnInit } from "@angular/core";  import { AppUser } from "../../../shared/models/app-user";  import { AuthService } from "../../../shared/services/auth.service";  @Component({  selector: "app-myprofile",  templateUrl: "./myprofile.component.html",  styleUrls: ["./myprofile.component.css"],  })  export class MyprofileComponent implements OnInit {  // appUser: AppUser;  constructor(private authService: AuthService) {}  async ngOnInit() {  // this.auth.appUser$.subscribe(appUser => this.appUser = appUser);  }  } |

**IV: Shipping page and Checkout:**

This page pulls information from the shopping cart and displays brief details of the products in the cart. Two forms are used to here in this page. All the input fields in this page are mandatory the inputs are validated using angular built-in functions. Successful orders are registered into the database. These details of the orders can be viewed in the order history of the user in the user profile section.

The TypeScript file of the component consists of the routing descriptions and methods to identify the user and the information from the shopping cart. The information of the order if successful is routed to the database from this component section.

**Shipping-form.Component.ts**

|  |
| --- |
| import { ShoppingCart } from '../../../shared/models/shopping-cart';  import { OrderService } from '../../../shared/services/order.service';  import { AuthService } from '../../../shared/services/auth.service';  import { Router } from '@angular/router';  import { Subscription } from 'rxjs/Subscription';  import { Component, OnInit, OnDestroy, Input } from '@angular/core';  import { Order } from "../../../shared/models/order";  @Component({  selector: 'shipping-form',  templateUrl: './shipping-form.component.html',  styleUrls: ['./shipping-form.component.css']  })  export class ShippingFormComponent implements OnInit, OnDestroy {  @Input('cart') cart: ShoppingCart;  shipping = {};  userSubscription: Subscription;  userId: string;    constructor(  private router: Router,  private authService: AuthService,  private orderService: OrderService) {  }  ngOnInit() {  this.userSubscription = this.authService.user$.subscribe(user => this.userId = user.uid);  }  ngOnDestroy() {  this.userSubscription.unsubscribe();  }  async placeOrder() {  let order = new Order(this.userId, this.shipping, this.cart);  let result = await this.orderService.placeOrder(order);  this.router.navigate(['/order-success', result.key]);  }  } |

**(Team Member: Sushmita Rudra)**

**Filters and categories:**

All the products added by admin are categorized and stored in firebase. The user has the ability to view the products based on the category selected. Depending upon the category selection, the products which are under that category are retrieved from firebase and displayed in the UI. A category service is used to integrate the database and typescript methods/observables.

**Sample code for UI template(Product-filter.html template):**

<div class="sticky-top">

  <div class="list-group">

      <a

        class="list-group-item list-group-item-action"

        [class.active]="!category"

        routerLink="/"

        >All Categories</a>

      <a

        \*ngFor="let c of categories$ | async"

        routerLink="/"

        [queryParams]="{ category: c.$key }"

        class="list-group-item list-group-item-action"

        [class.active]="category === c.$key"

        >

        {{ c.name }}

      </a>

    </div>

</div>

**Sample code of backend (product-filter.ts):**

|  |
| --- |
| @Component({    selector: 'product-filter',    templateUrl: './product-filter.component.html',    styleUrls: ['./product-filter.component.css']  })  export class ProductFilterComponent implements OnInit {    categories$;    @Input('category') category;    constructor(categoryService: CategoryService) {      this.categories$ = categoryService.getAll();    }    ngOnInit() {    }  } |

**Sample code of category service:** The category service is used to filter the products and retrieve them from firebase based on the name of the category.

@Injectable()

export class CategoryService {

  constructor(private db: AngularFireDatabase) { }

  getAll() {

    return this.db.list('/categories', {

      query: {

        orderByChild: 'name'

      }

    });

  }

}

**User order history:** Order history is an important functionality of the application. All the orders placed by the user are displayed in the UI. Each user has a unique ‘uid’, so the orders belonging to the logged in user are retrieved from the firebase by filtering out orders based on the ‘uid’.

**Sample code for User order history template:**

|  |
| --- |
| <h1>Order History</h1>  <table class="table">    <thead>      <tr>        <th>Date</th>        <th style="padding-left: 50px">Items ordered</th>      </tr>    </thead>    <tbody>      <tr \*ngFor="let order of orders$ | async">        <td>{{ order.datePlaced | date }}</td>        <td>          <ul class="list-group" \*ngFor="let item of order.items">            <li class="list-group-item" style="border: none">              {{ item.product.title }} x {{ item.quantity }} =              {{ item.totalPrice | currency: "USD":true }}            </li>          </ul>        </td>      </tr>    </tbody>  </table> |

(Team Member: Omkar Abhiteja Badda)

1. **Product display component:**

Products display component is developed using Angular which works as frontend framework and Type Script is used to write the backend logic for storing and retrieving data. Product display component is used to display products present in the store and users can login and add or remove their desired products. Products data is stored Firebase which consists of a list of products which stores data in JSON format. Firebase storage API is integrated to the application.

**Sample code for Angular Form:**

|  |
| --- |
| <div \*ngIf="product.title" class="card" style="width: 250px">  <img  \*ngIf="product.imageUrl"  class="card-img-top"  [src]="product.imageUrl"  alt="{{ product.title }}"  />  <div class="card-body">  <h4 class="card-title">{{ product.title }}</h4>  <p class="card-text">{{ product.description }}</p>  <p class="card-text">{{ product.price | currency: "USD":true }}</p>  </div>  <div \*ngIf="showActions && shoppingCart" class="card-footer"> |

User can add multiple products and if in case does not require any product he can remove one or all at a time by clicking on the “Clear Shopping Cart” field present on the top right corner of the page. After selecting all the required products, user can click on the Checkout button present at the bottom by which he will get redirected to the checkout page where he can review his products and pay the price.

**Sample Backend code for product display component:**

|  |
| --- |
| <div class="row no-gutters">  <div class="col-2">  <button  (click)="removeFromCart()"  class="btn btn-secondary btn-block">-</button>  </div>  <div class="col text-center quantity">  {{ shoppingCart.getQuantity(product) }} in cart  </div>  <div class="col-2">  <button  (click)="addToCart()"  class="btn btn-secondary btn-block">+</button>  </div>  </div> |

1. **Live Chat and Support Page:**

One of the most useful functionality for our customers is Live Chat where users can directly contact the admin regarding their queries. Every single page in the application has this functionality present at the bottom right corner with an icon in green and white color. The chat is entirely private and is only visible to user and admin. The best feature is admin is able to chat with multiple users at the same time. A different channel will get created to each user when they start the chat. Users can even upload files and react with emoticons which are pre-build in the chat functionality.

**Sample code for Live Chat:**

|  |
| --- |
| <app-root></app-root>  <script type="text/javascript">  var Tawk\_API = Tawk\_API || {},  Tawk\_LoadStart = new Date();  (function () {  var s1 = document.createElement("script"),  s0 = document.getElementsByTagName("script")[0];  s1.async = true;  s1.src = "https://embed.tawk.to/5f839039f0e7167d00181868/default";  s1.charset = "UTF-8";  s1.setAttribute("crossorigin", "\*");  s0.parentNode.insertBefore(s1, s0);  })();  </script>  <script src="https://checkout.stripe.com/checkout.js"></script> |

Contact support page is a normal static page which mainly displays information about what BigKart is and some of the frequently asked questions and their answers. Users can find this page in the navigation bar present at the top of the page. A field names “Support” is present right beside the cart icon.

**Sample code for Support page:**

|  |
| --- |
| @Component({  selector: 'app-support',  templateUrl: './support.component.html',  styleUrls: ['./support.component.css']  })  export class SupportComponent implements OnInit {  constructor() { }  ngOnInit() {  } |

**(Team Member: Kamal Dontireddy)**

1. **Admin Order Management:**

Admin order management is built on Angular and Type Script. Angular works as frontend framework and Typescript is used to write backend logic for storing a retrieving data. Admin order management consists of a form where admin can view the orders placed by the customer using the database. Firebase is used to store the list of orders placed by users. At any time, admin is able to see the orders placed by the users in the database and delete the unwanted orders from the store database. Firebase acts as storage service which stores data in JSON format. Firebase Storage API is integrated to the application.

**Sample code for Angular Form:**

|  |
| --- |
| import { ShoppingCart } from './shopping-cart';  export class Order {  datePlaced: number;  items: any[];  constructor(public userId: string, public shipping: any, shoppingCart: ShoppingCart) {  this.datePlaced = new Date().getTime();  this.items = shoppingCart.items.map(i => {  return {  product: {  title: i.title,  imageUrl: i.imageUrl,  price: i.price  },  quantity: i.quantity,  totalPrice: i.totalPrice  }  })  }  } |

All the order data entered in the form is stored and when admin clicks on view order button, which call the view function to view the existing orders placed by the users. Similarly, Delete function is used to delete the product from database. Router defines the path to navigate within the admin management of orders. When admin delete a order, it is deleted in the database based on the id of the order.

**Sample Backend code for admin order management:**

|  |
| --- |
| import { ShoppingCartService } from './shopping-cart.service';  import { AngularFireDatabase } from 'angularfire2/database';  import { Injectable } from '@angular/core';  @Injectable()  export class OrderService {  constructor(private db: AngularFireDatabase, private shoppingCartService: ShoppingCartService) { }  async placeOrder(order) {  let result = await this.db.list('/orders').push(order);  this.shoppingCartService.clearCart();  return result;  }  getOrders() {  return this.db.list('/orders');  }  getOrdersByUser(userId: string) {  return this.db.list('/orders', {  query: {  orderByChild: 'userId',  equalTo: userId  }  });  }  get(orderId){  return this.db.object('/orders/' + orderId);  }  } |

1. **Adding products to shopping cart Functionality:**

When user adds products to the cart, this functionality which I developed calculates the total cost of the order and helps the user to add or delete the items in the shopping cart. This is also updated in the navigation bar cart with number of items in shopping cart.

**Sample code for shopping cart:**

|  |
| --- |
| import { Product } from './product';  import { ShoppingCartItem } from './shopping-cart-item';  export class ShoppingCart {  items: ShoppingCartItem[] = [];  constructor(private itemsMap: { [productId: string]: ShoppingCartItem }) {  this.itemsMap = itemsMap || {};    for (let productId in itemsMap) {  let item = itemsMap[productId];  this.items.push(new ShoppingCartItem({ ...item, $key: productId }));  }  }  getQuantity(product: Product) {  let item = this.itemsMap[product.$key];  return item ? item.quantity : 0;  }    get totalPrice() {  let sum = 0;  for (let productId in this.items)  sum += this.items[productId].totalPrice;  return sum;  }    get totalItemsCount() {  let count = 0;  for (let productId in this.itemsMap)  count += this.itemsMap[productId].quantity;  return count;  }  } |

## CHAPTER 7

## END USER MANUAL

**(Team member: Deepak Malempati)**

1. **Admin Product Management:**
2. Login to the application with admin privileges and navigate to manage products from navigation bar

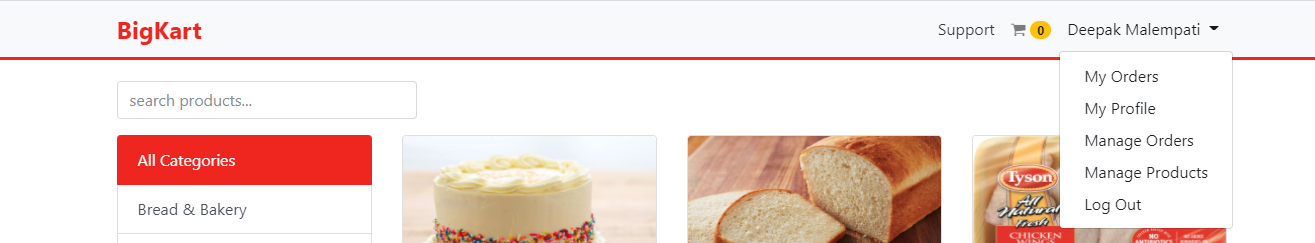


Fig 7.1: Admin Product Management

1. Select add new product and enter product details

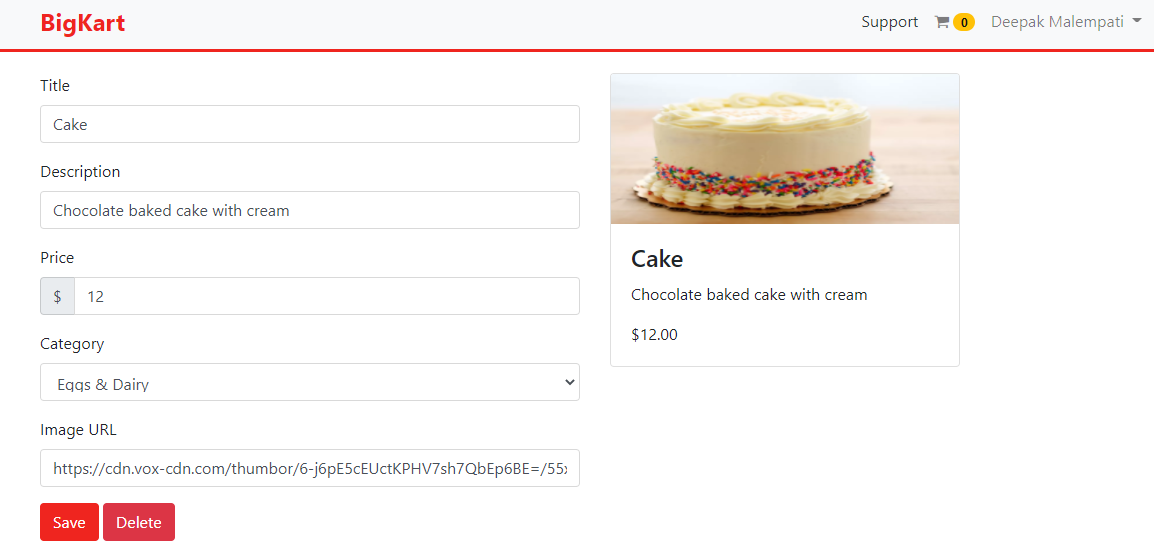


Fig 7.2: Products addition

1. Save the product and view in the list which is added to the database.

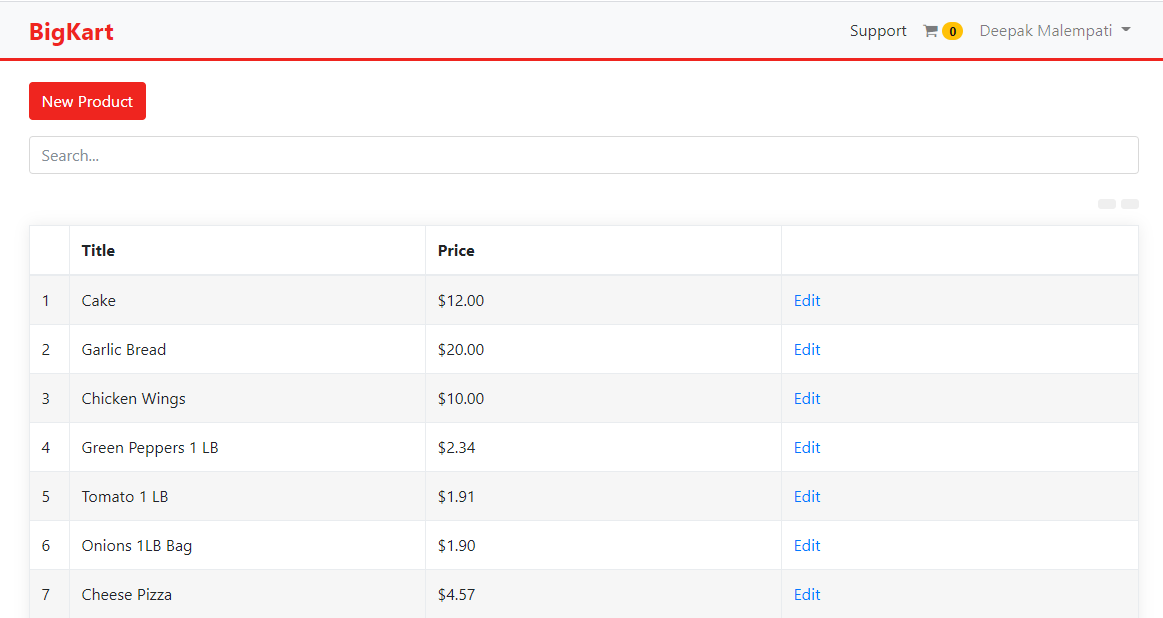


Fig 7.3: Product List in Database

1. **Product Search Functionality:**

User can search the products in homepage by entering description or title of the product in search bar.

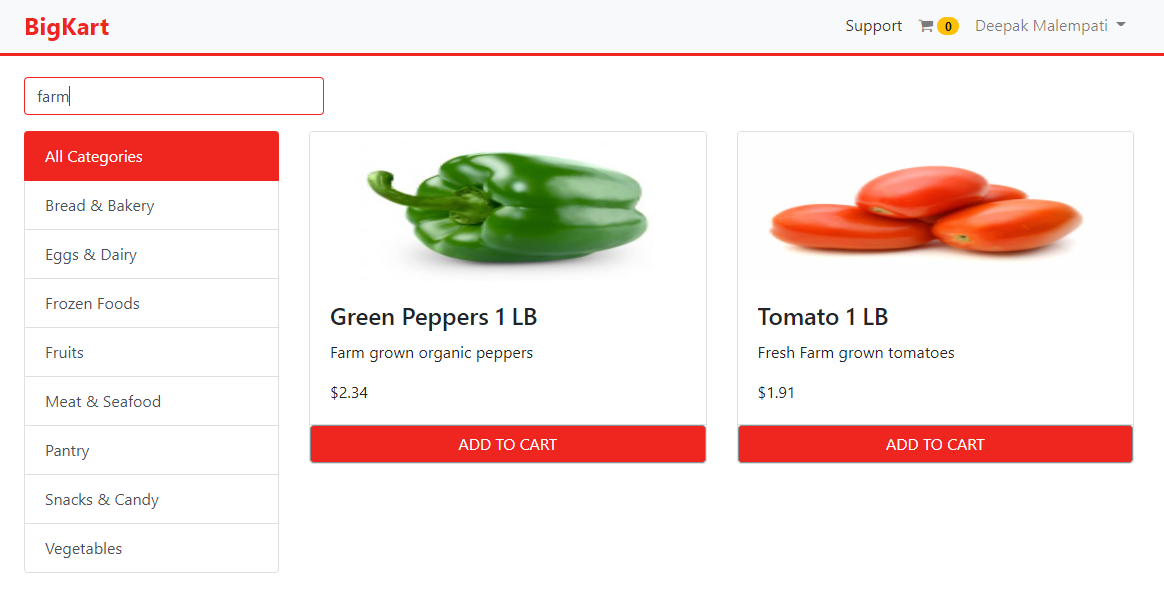


Fig 7.4: Search Functionality

**(Team Member: Kamal DontiReddy)**

**Admin Order Management:**

1. **Login to the application with admin privileges and navigate to manage orders from navigation bar**

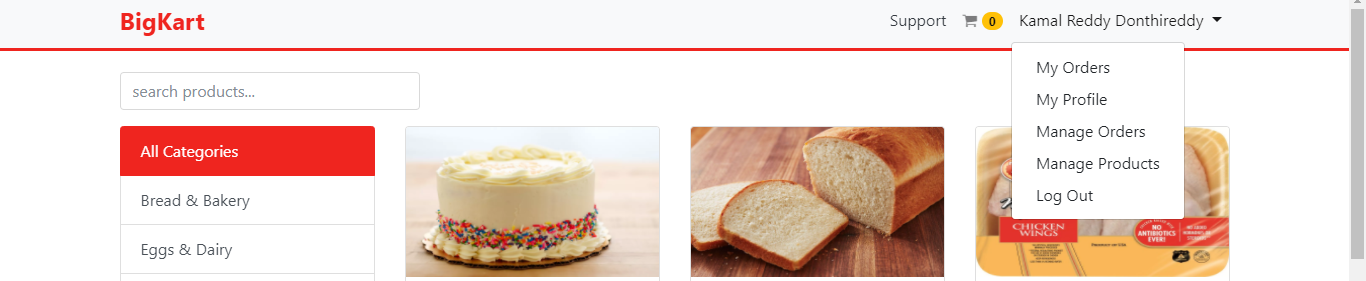


Fig 7.5: Admin Login

1. **When you select manage orders you can view different orders**

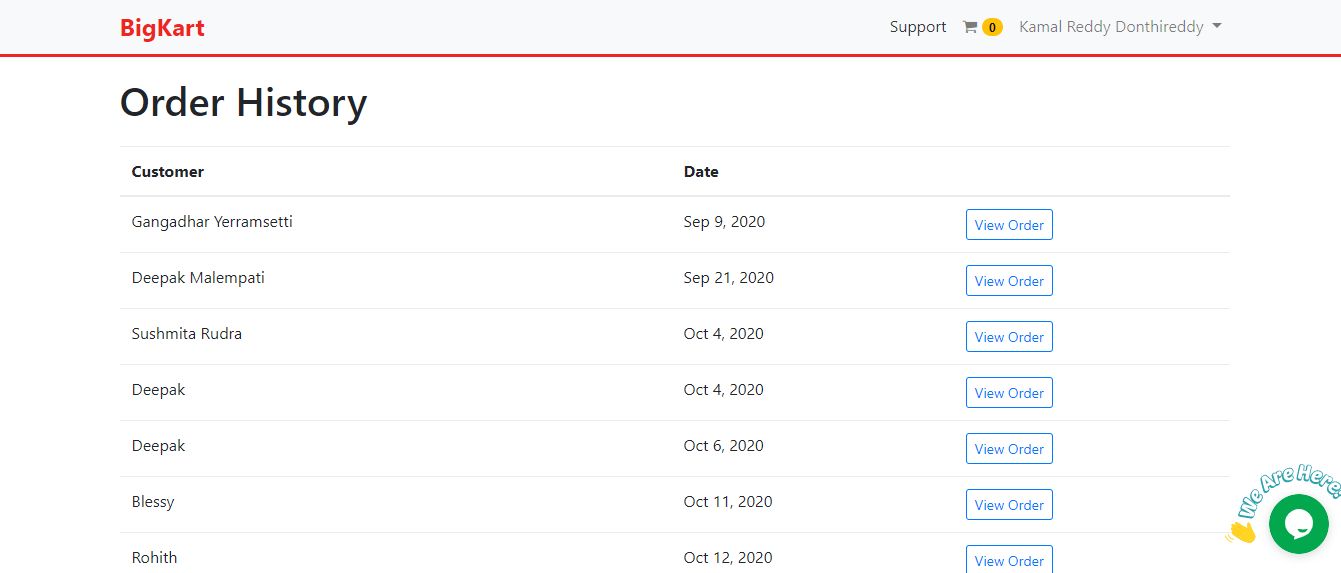


Fig 7.6 Admin side Order history

1. **Click on view order to see the order details.**

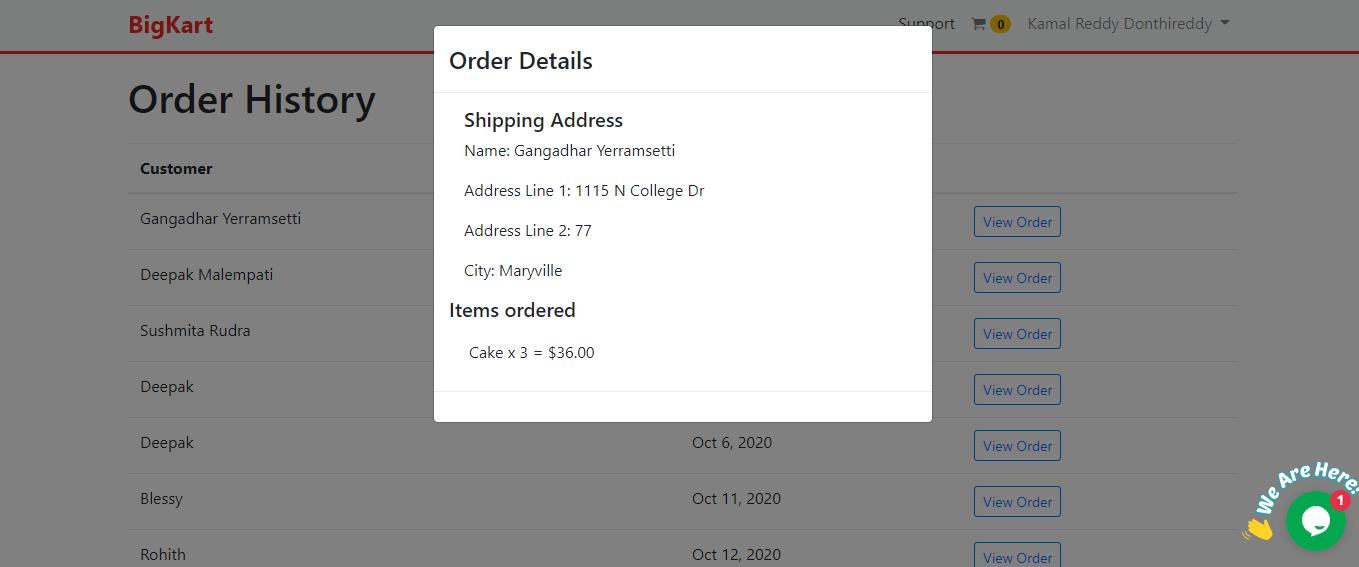


Fig 7.7 Admin side Order history details

1. **Adding products to shopping cart Functionality:**

When user adds products to the cart he or she can view the total cost of the order and from this he or she can add or delete products.

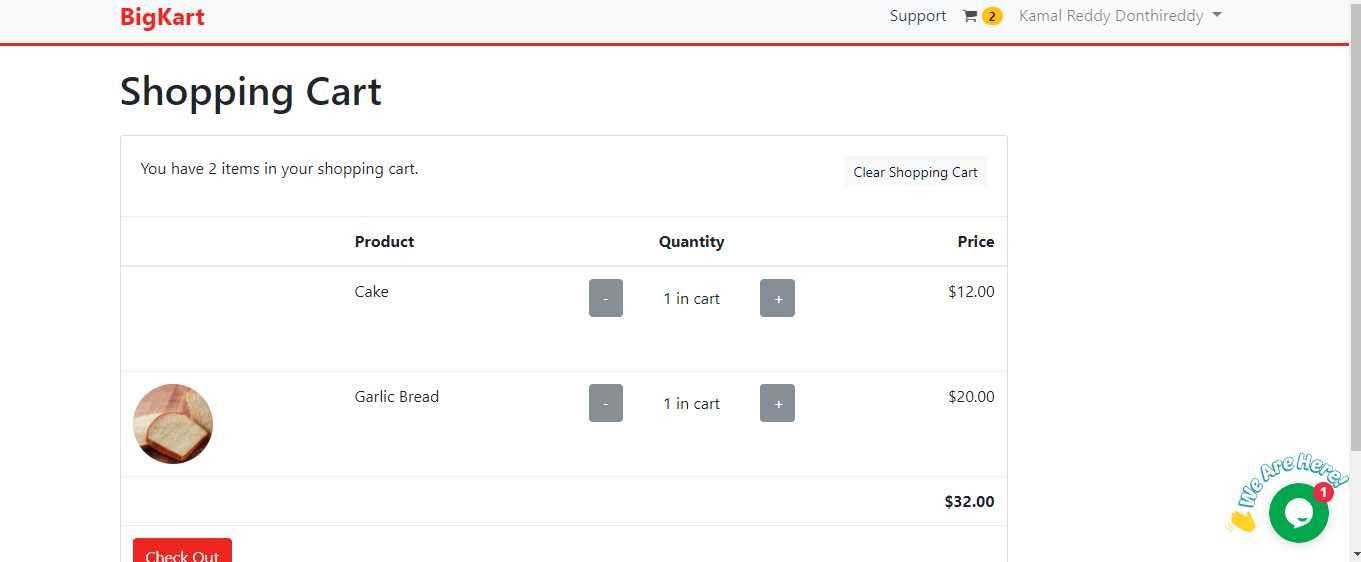


Fig 7.8 Shopping Cart

**(Team member: Gangadhar Yerramsetti)**

**III. User Profile:**

1. **User Login:** The user needs to sign in using the credentials that he has already registered with. If the user wishes to sign up he can use the “sign up” button at the button of the page. If the user has forgot his password he can use the “forgot password” button at the bottom of the page.

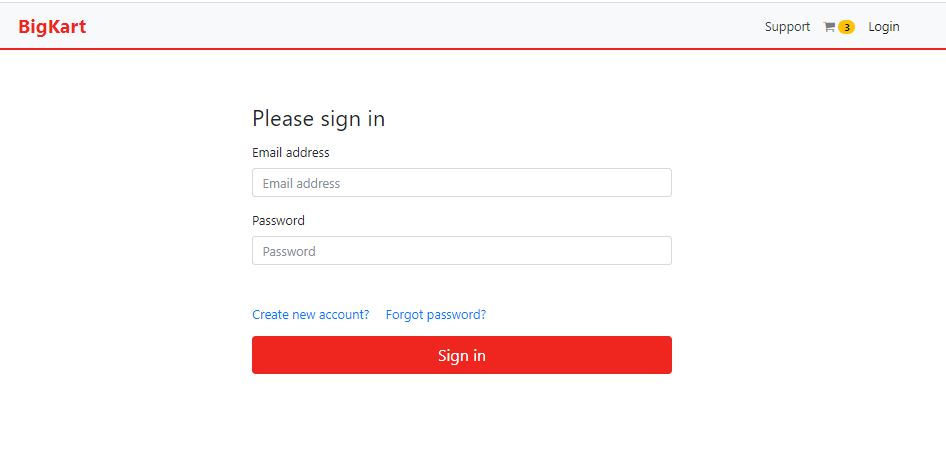


Fig 7.9: User Login

**2. User Sign Up:** The user needs to provide his “First Name”, “Last Name”, “Email address” and “Password”. The user can further use his registered email address and password to login into the application. Upon successful registration the credentials get registered in the firebase database.

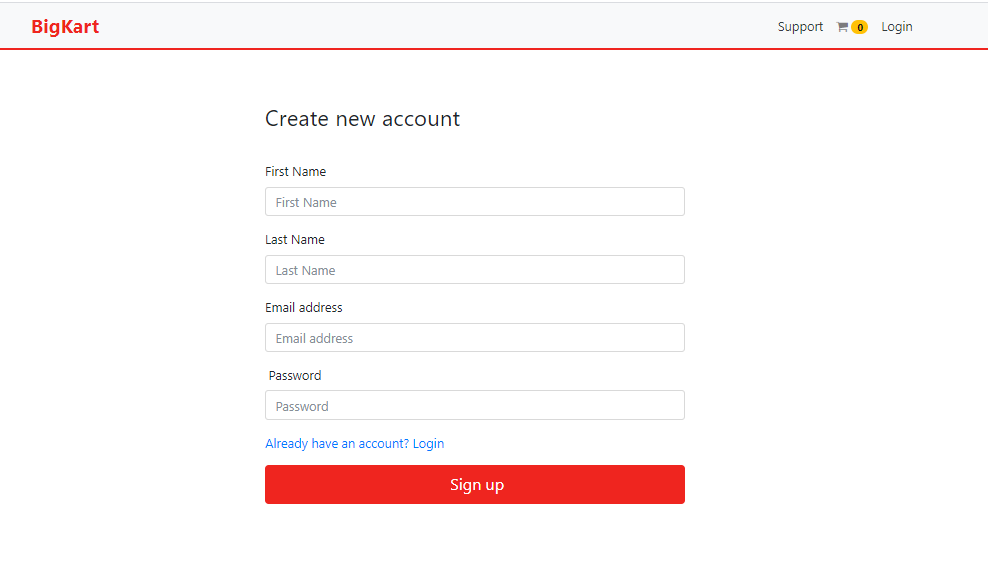


Fig 7.10: User Sign Up

**4. Forgot Password:**

The user can use the forgot password page to receive a password reset email in their personal email inbox.

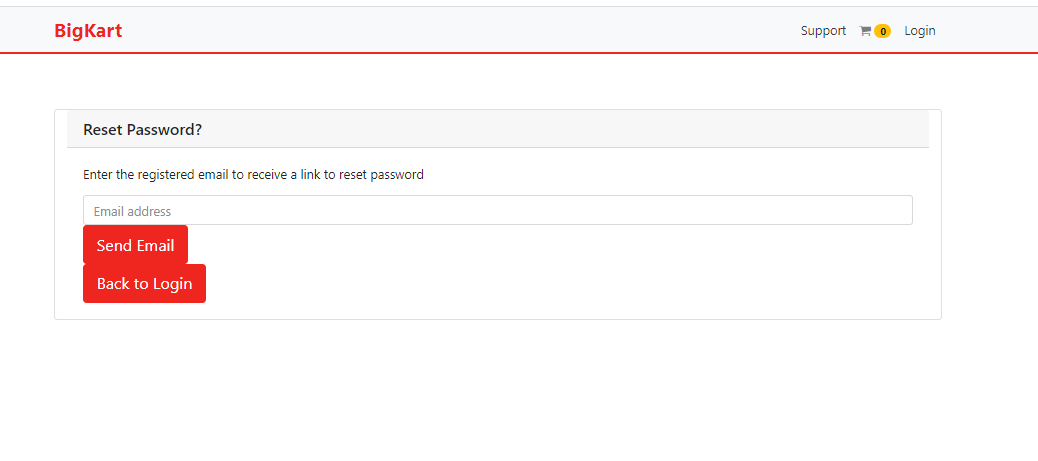


Fig 7.11: Forgot Password

**My Profile:**

The user when logs into the application is recognized and displayed in the right corner of the navigation bar. We can navigate to user profile by using the dropdown that appears when clicked on the username.

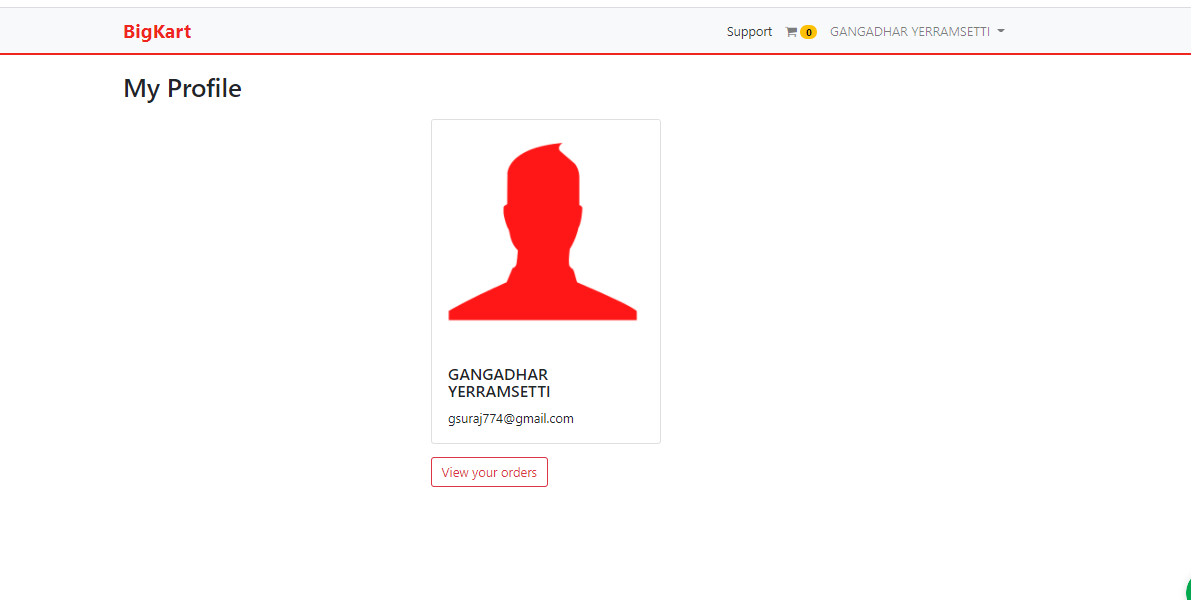
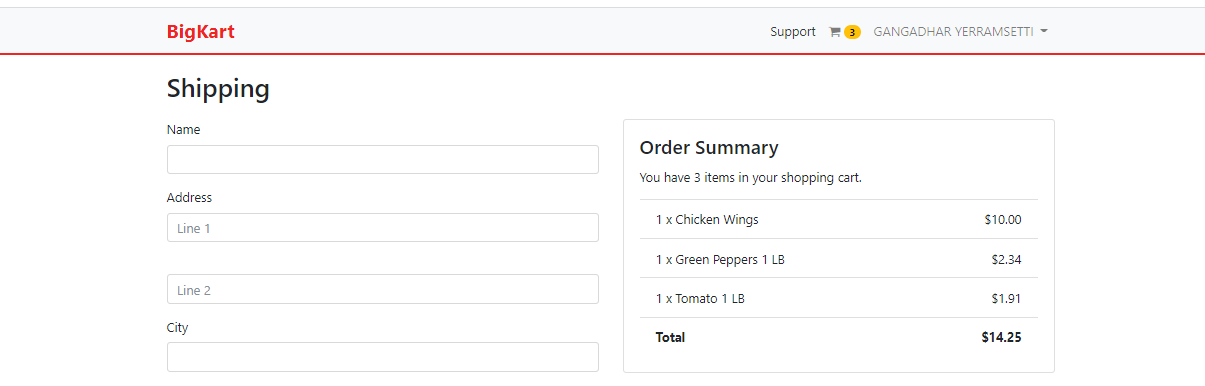


Fig 7.12: User Profile

**IV: Shipping page and Checkout:**

The checkout and shipping page of the application displays the items in the cart along with the order total. The user can supply the card information along with the shipping details. All the fields here are mandatory and the user can use the “Place order” button at the bottom of the page to place the order.



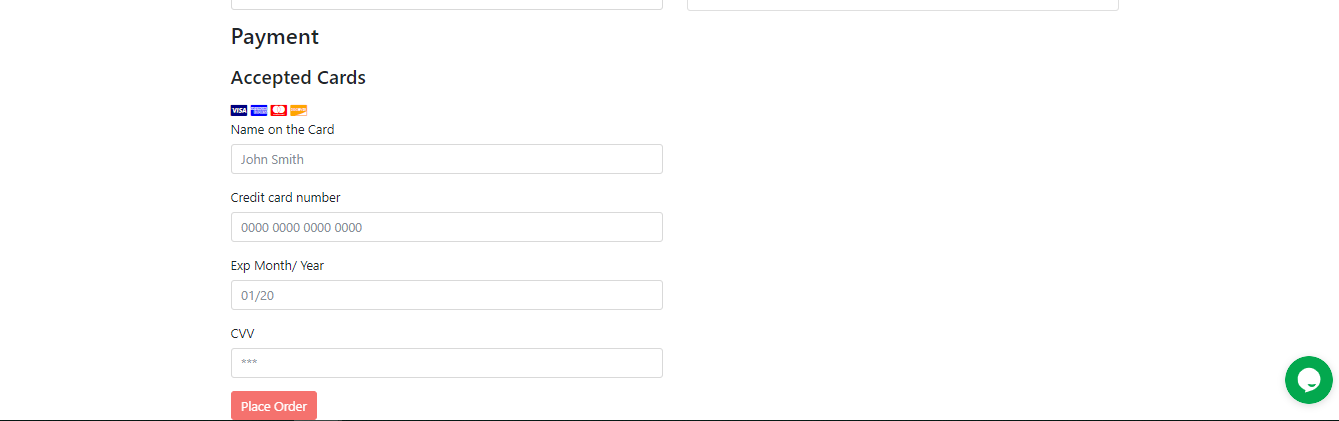


Fig 7.13: Checkout and Shipping page

**(Team Member: Sushmita Rudra)**

**III Filters and categories functionality:**

Login to the application as a user, and click on the particular category which will filter out the products accordingly. On selection of category, only products that fall under that particular category are filtered out and displayed.

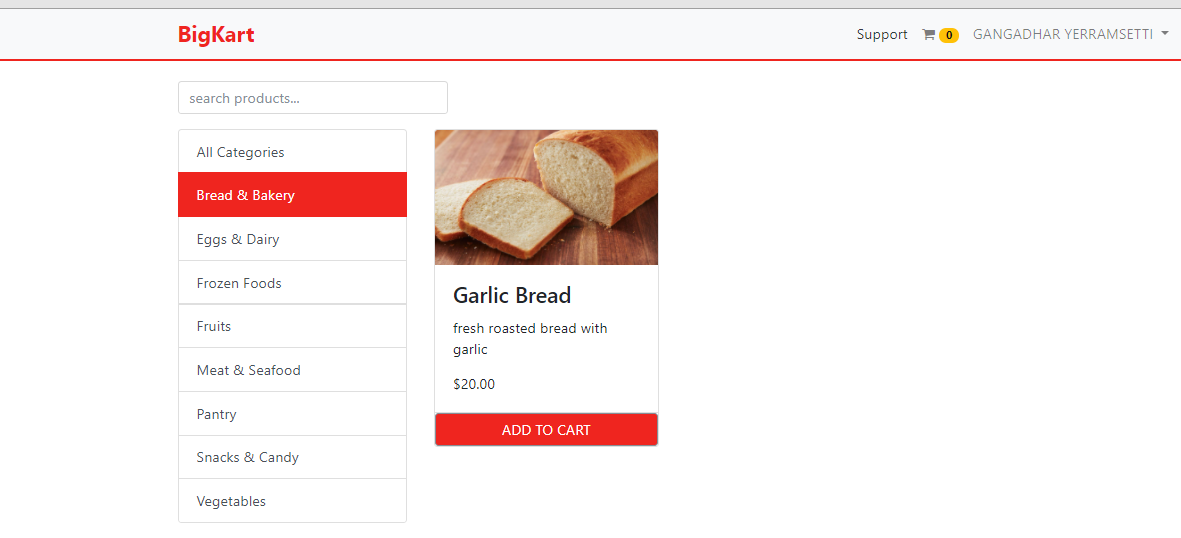


Fig 7.14: Filters and Categories

**User order history functionality:**

After placing an order, the user can view all his orders by navigating to ‘My Orders’ where the information about the date and items ordered resides.

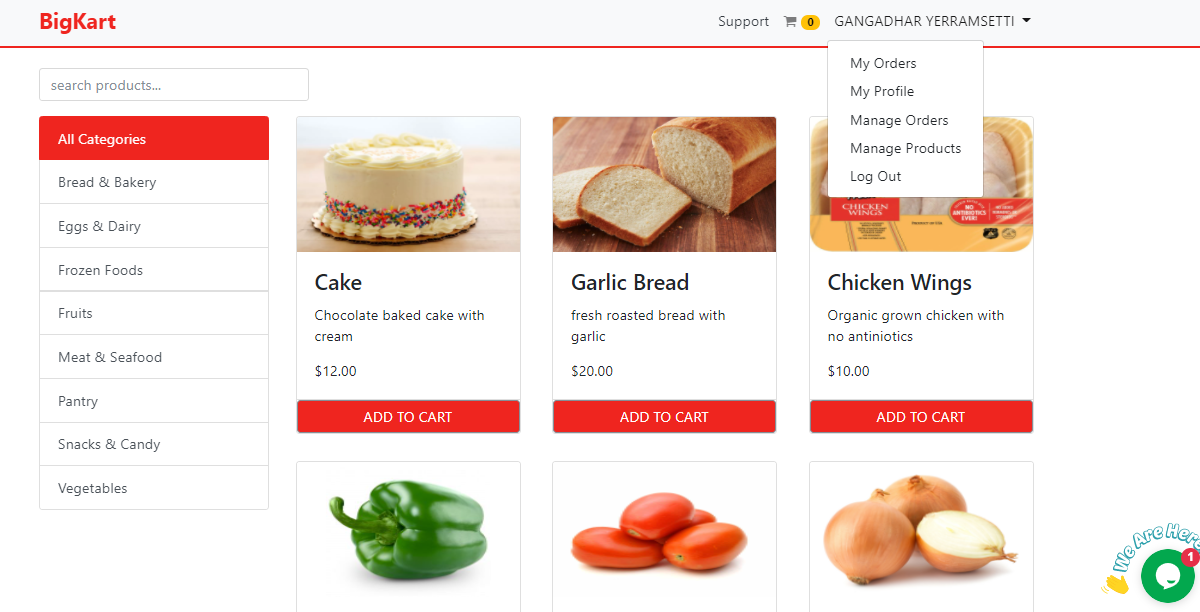


Fig 7.15: User Order History -1

The items ordered and quantity of items is also displayed.

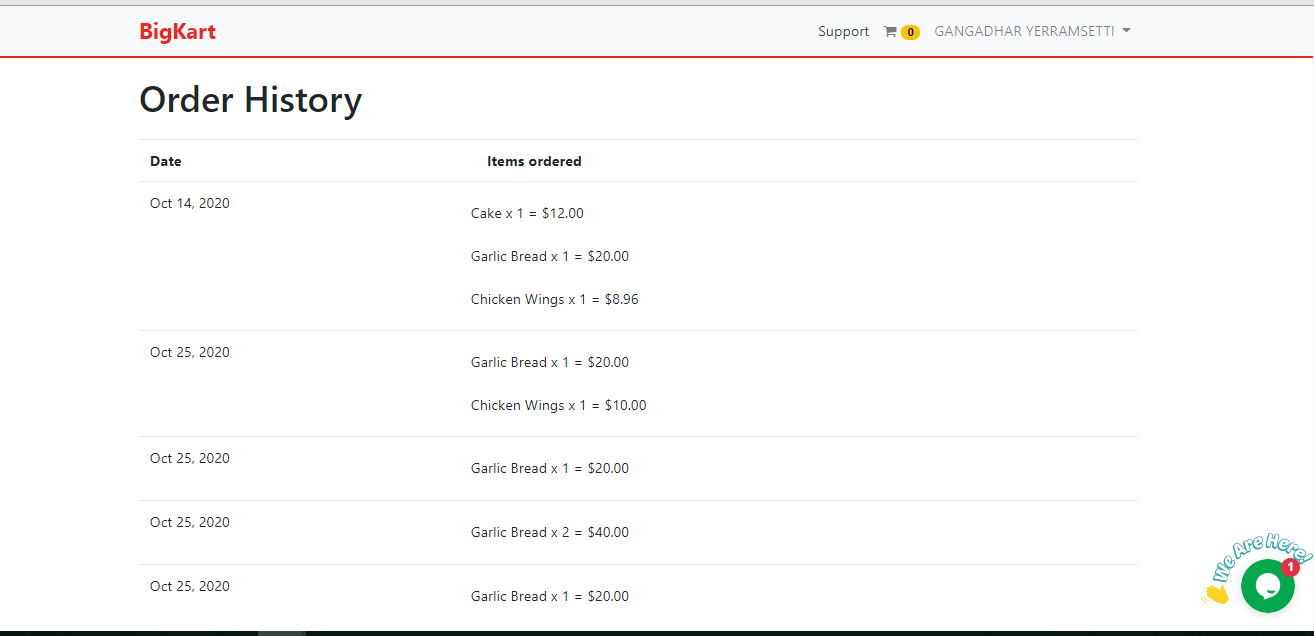


Fig 7.16: User Order History -2

(Team Member: Omkar Abhiteja Badda)

1. **Product Display Component:**
2. Login to the application with user credentials and home page will get displayed with a wide variety of products. Select the desired products by clicking on the Add to Cart button.

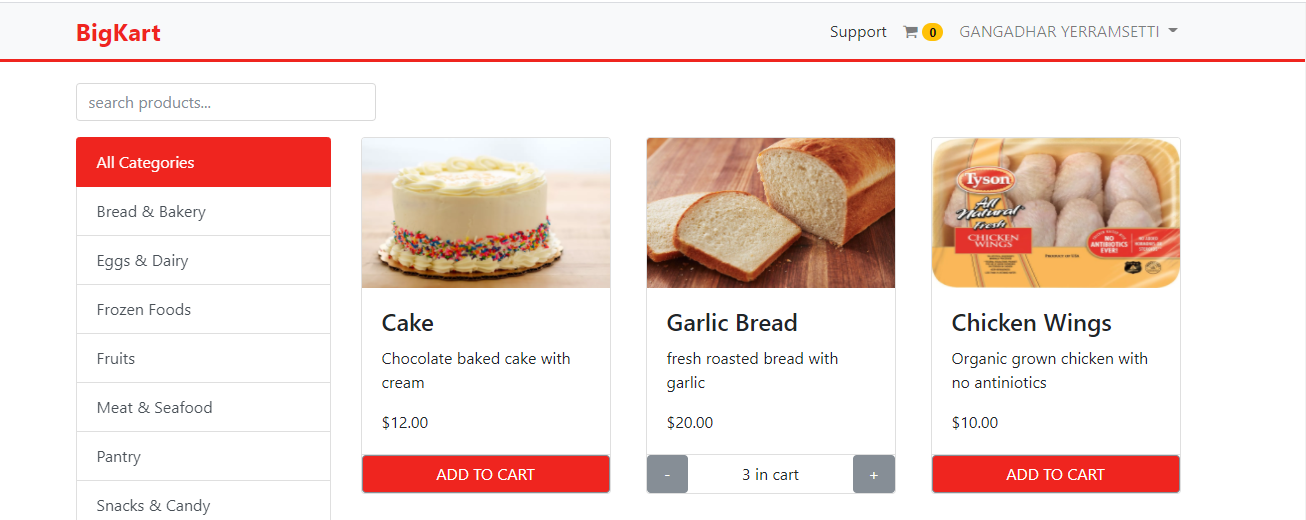


Fig 7.17: Product display

1. Click on the cart button on the top and verify the items present in the cart. Increase or decrease the quantity as you require. The price will also get updated simultaneously.

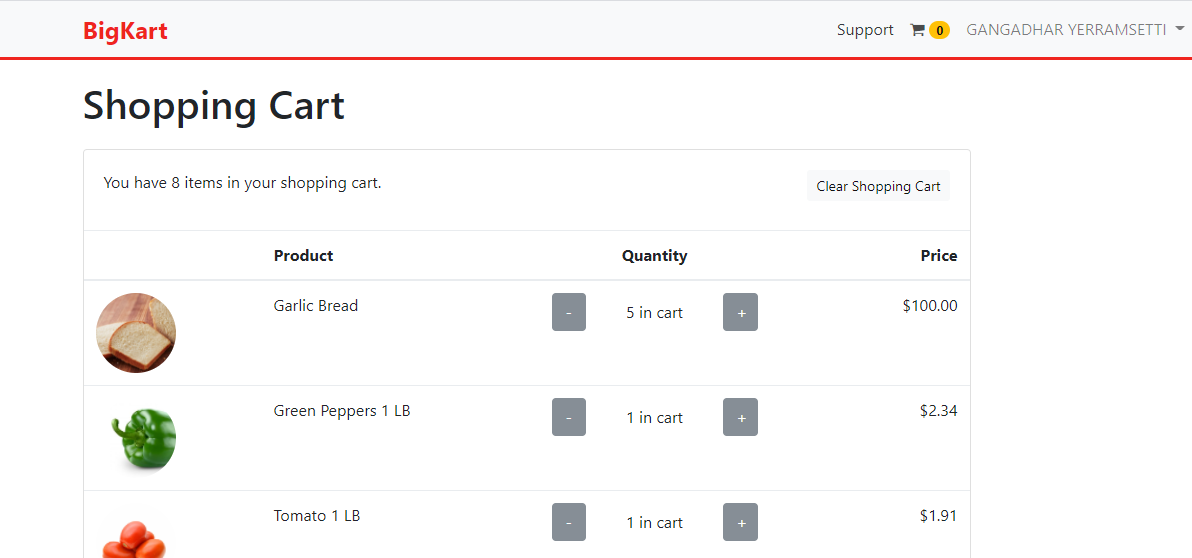


Fig 7.18: Manage Shopping Cart

1. Click on the clear shopping cart field so that all the products will get deleted and the homepage will again get displayed normal.

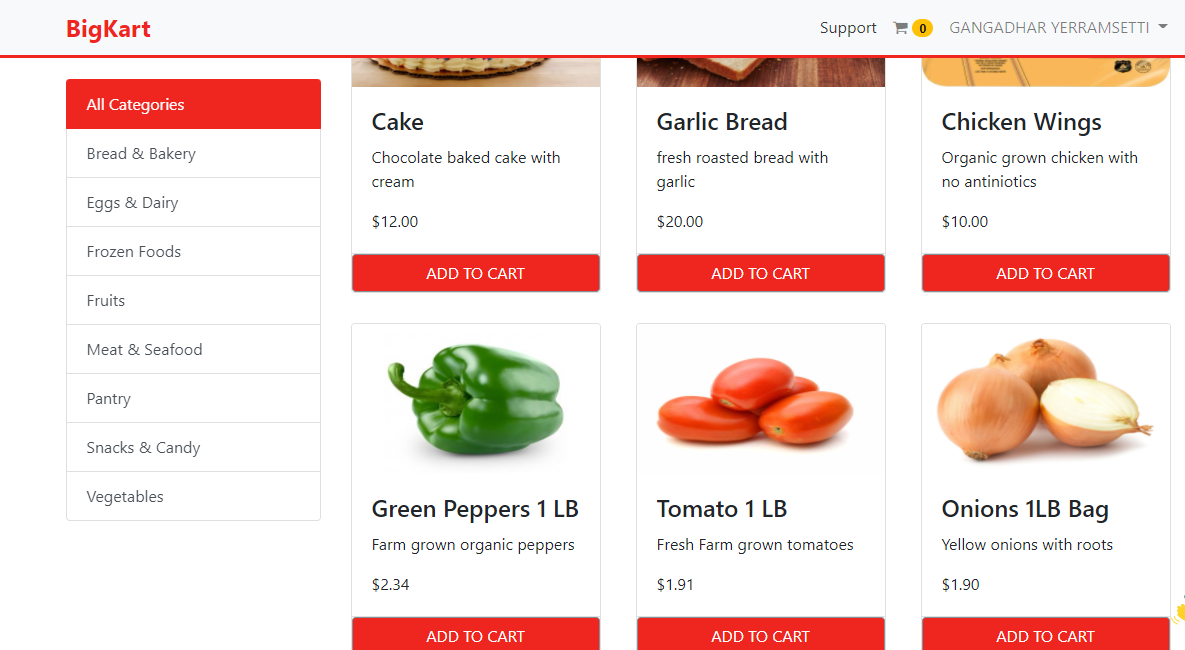


Fig 7.19: Home Page after clearing cart

1. **Live Chat and Support page:**
2. Every page of the application has an icon at the bottom right with a small message “We are here!” and by clicking on it user will be able to chat with the admin

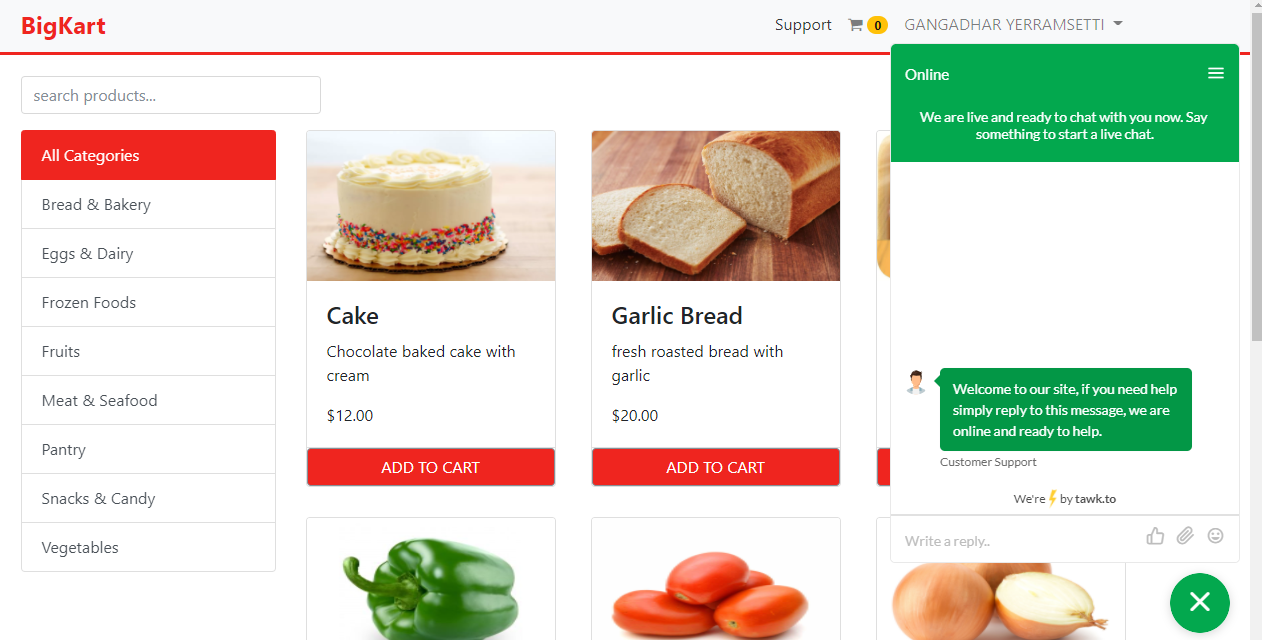


Fig 7.20: Live Chat – User End - 1

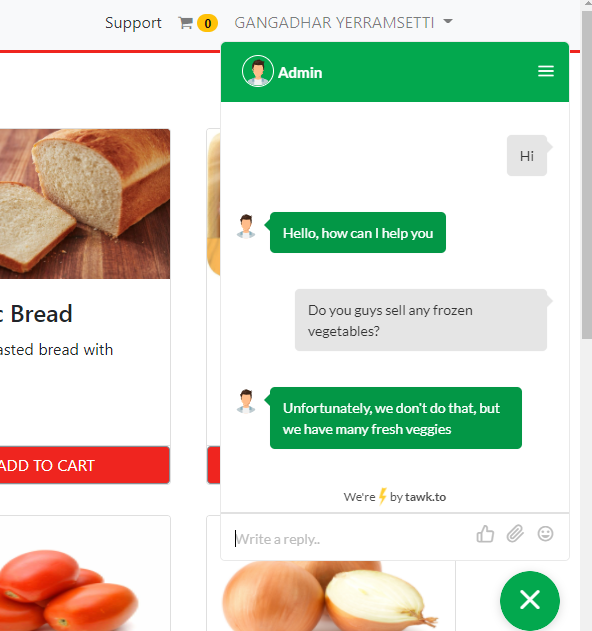
1. **Users can post their questions and admin will be able to answer them.** 

Fig 7.20: Live Chat – User End - 2

1. **Admin can chat with multiple users at the same time.**

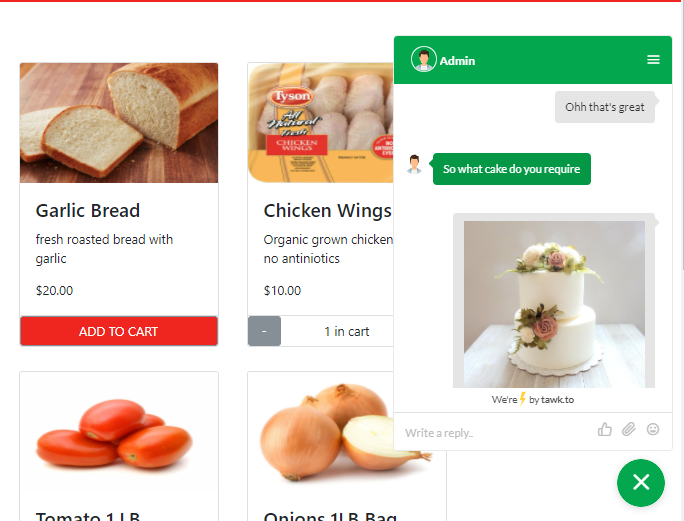
****

Fig 7.21: Live Chat – User End - 3

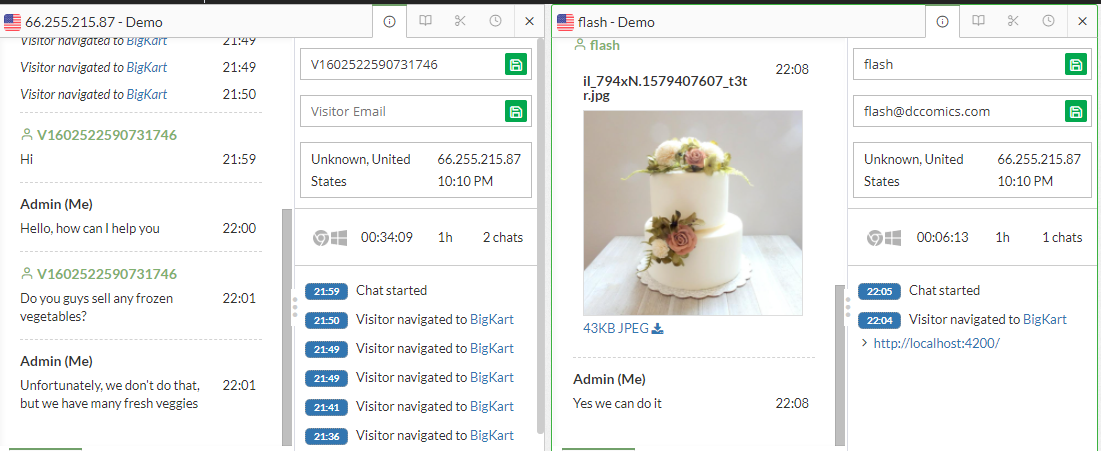


Fig 7.22: Live Chat – Admin Side

1. **Contact page displays information related to the frequently asked questions.**

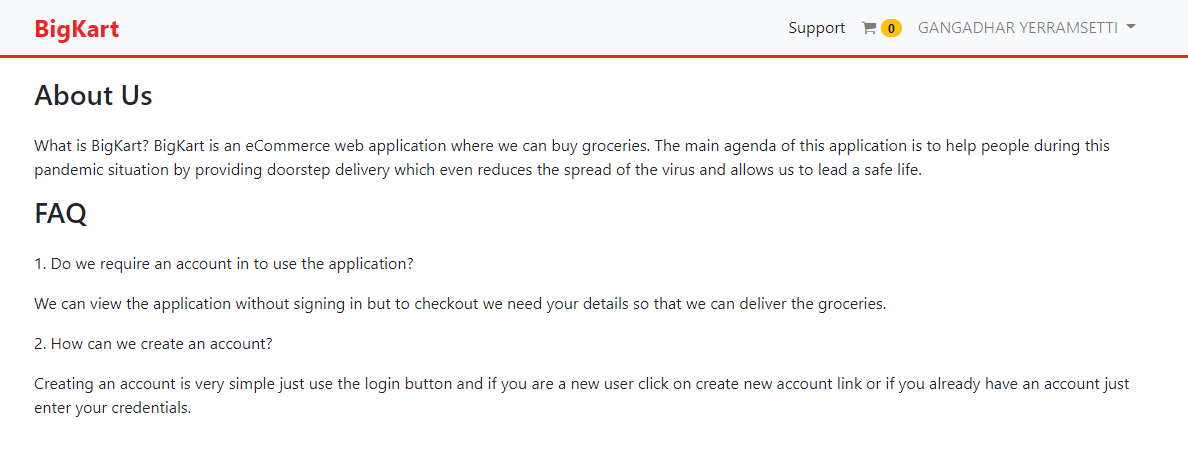


Fig 7.23: Support and Help