CPS 630

Project Report Itr3-4

Team 20

Technical Report for Smart Customer Services (S.C.S) Web application

Team Members:

Name	Student Number
Yasser Ammar Abady	500911995
Saif Ahmed Alvi	500834841
Tusaif Azmat	500660278

Tasks Done By Each Team Member:

Name	Tasks Done	Percentage
Yasser Ammar Abady	Sign in system, About Us/Contact, General debugging, frontend, Angular Js, Technical Report	33.3%
Saif Ahmed Alvi	New service, Shopping cart, Technical Report	33.3%
Tusaif Azmat	Database maintain mode pages, Technical Report, Backend, SPA, adding additional features	33.3%
Total		100%

Objectives:

In this iteration 3-4 we have applied changes on the web-application that was developed in iter-1&2 in order to cover all requirements defined in (A), (B), and (C) below (Please note that most of requirements in A and B were already developed in Iter-1&2).The web application is developed under Nodejs, XAMPP Apache, MySQL, and PHP with SPA architecture. Additionally, the web app includes open libraries such as Google Maps APIs to assist the system.

Languages:

AngularJs: One Page Architecture
CSS: used to style whole website
JavaScript: used to extract data from database
PHP: used to process requests for the forms
SQL: used to create, insert, modify, etc
Tools: Nodejs, XAMP, W3C, Notepad++

Design for Iteration-III:

A) The system should work in two separate modes: Admin-mode and User-mode as follows:

1- **"Admin-mode"**

The Admin is being able to 'Maintain" the system for all major components including client, server and databases.
The Admin can maintain the database to add /delete/search/update the data in database. The Admin can maintain the database to cover all types of data associated with the web application.

2- **"Sign Up"**

The ordinary Users are able to "Sign-in" to work with the system, only if they already registered through "Sign-up" and created a profile.

B) The system should provide Services to the ordinary Users as follows:

1- **"Service (a), (b)"**
The two services that were already defined in It-1&2: (a) Shopping and (b) Delivery

2- **"Service (c)" User can able to compare the items before purchasing and Suggestion of new ware house location.**
These are the extra Services that other team has provided as an optional services. We believe this is an interesting new Service along with extra features that makes the system to be preferable by the users. The layout and UI for Service (c) is designed such a way that it is relevant to the other UIs in the system. After using service (c) which allows user to select the items in the shopping cart and give them option to compare before purchase. The invoice is generated and displayed on the screen along with the delivery map and the system is ready for the payment process.

localhost | CPS630 Web Application T20

localhost/CPS630/Online-Store/app/#!/compare

Item #04

Price: $999.99

FROM:

2963 Britannia Rd Building C, Mississauga, ON L5N 0A2

TO:

10344 Mississauga Rd, Brampton, ON L7A 0B8, Canada

DATETIME:

2022-04-14 09:55

Rating:

Select this Item

Item #01

Price: $999.99

FROM:

2963 Britannia Rd Building C, Mississauga, ON L5N 0A2

TO:

10344 Mississauga Rd, Brampton, ON L7A 0B8, Canada

DATETIME:

2022-04-14 09:55

Rating:

3

Select this Item

Type here to search

53°F ENG US 9:56 AM 2022-04-14

localhost | CPS630 Web Application T20

localhost/CPS630/Online-Store/app/#!/checkout

Order Summary

Item #04

$999.99

FROM:

2963 Britannia Rd Building C, Mississauga, ON L5N 0A2

TO:

10344 Mississauga Rd, Brampton, ON L7A 0B8, Canada

DATETIME:

2022-04-14 09:55

Payment Information

First Name: john

Last Name: smith

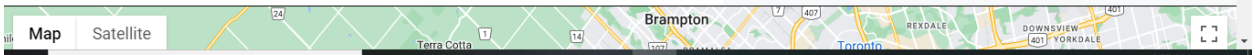
Email: js@email.com

Card Number: 4545124545781236

Expiration Date (MM/YYYY): 05/2022

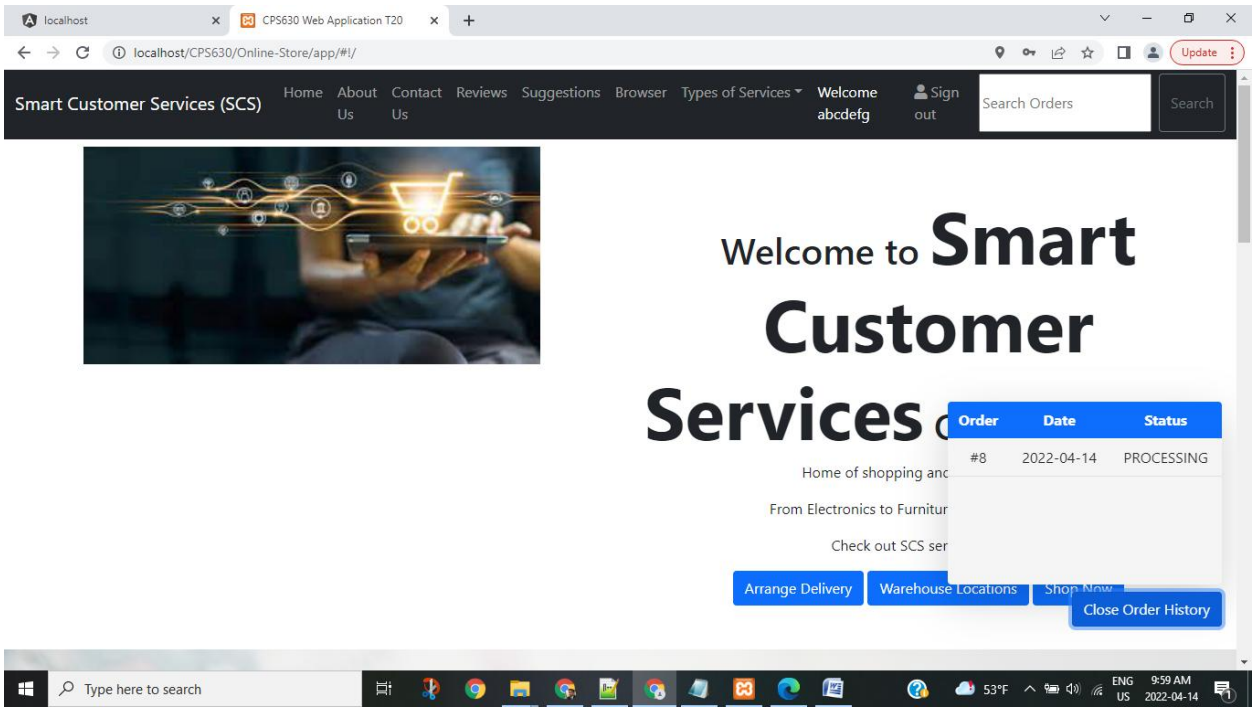
CVC/CVV: 125

Place Order



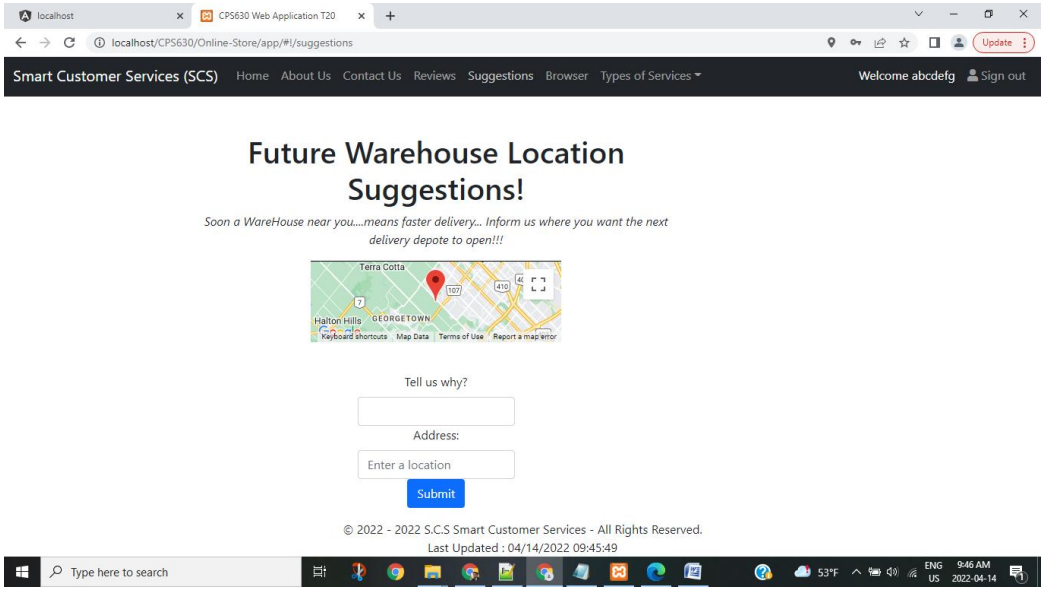
Type here to search

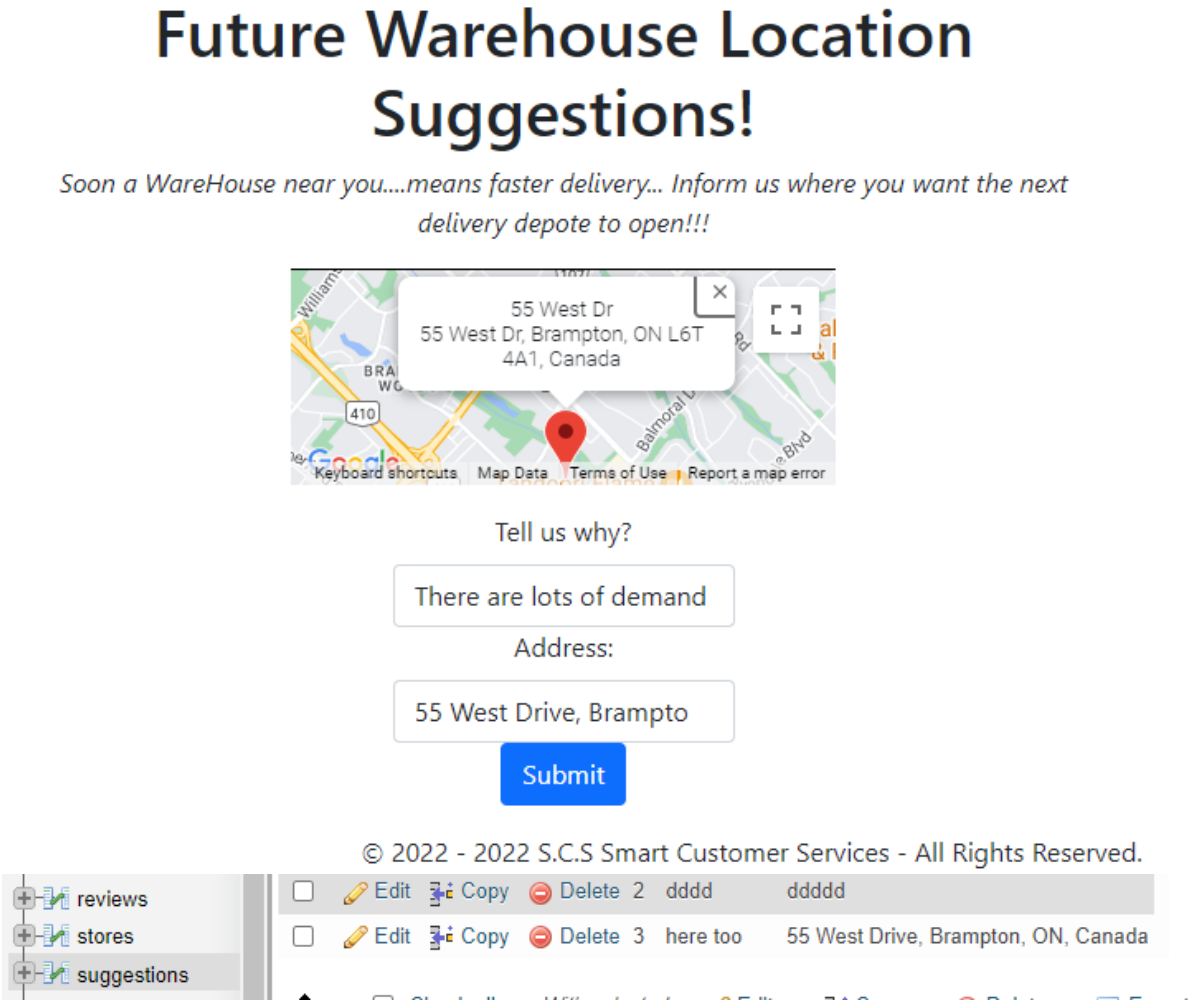
53°F ENG US 9:57 AM 2022-04-14



After purchasing the order shows up in user's order history list.

Following is also a new service we have added along with the one mentioned above:

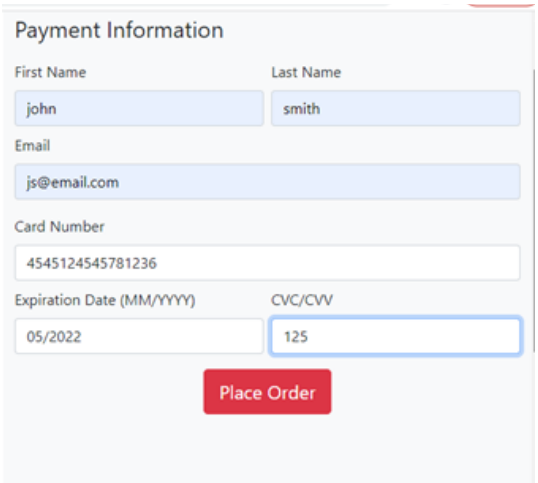




As you can see the suggestion is added to the database.

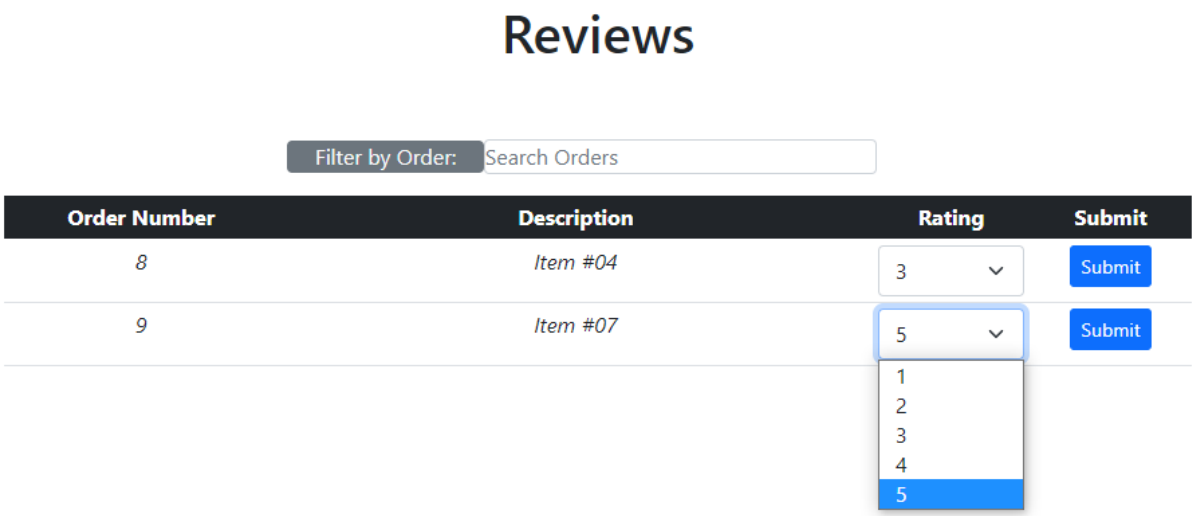
3- **"Payment"**

User can able to enter the credit card number and the banking information to do the purchase and see the banking transaction result (assuming that it is always a successful transaction).



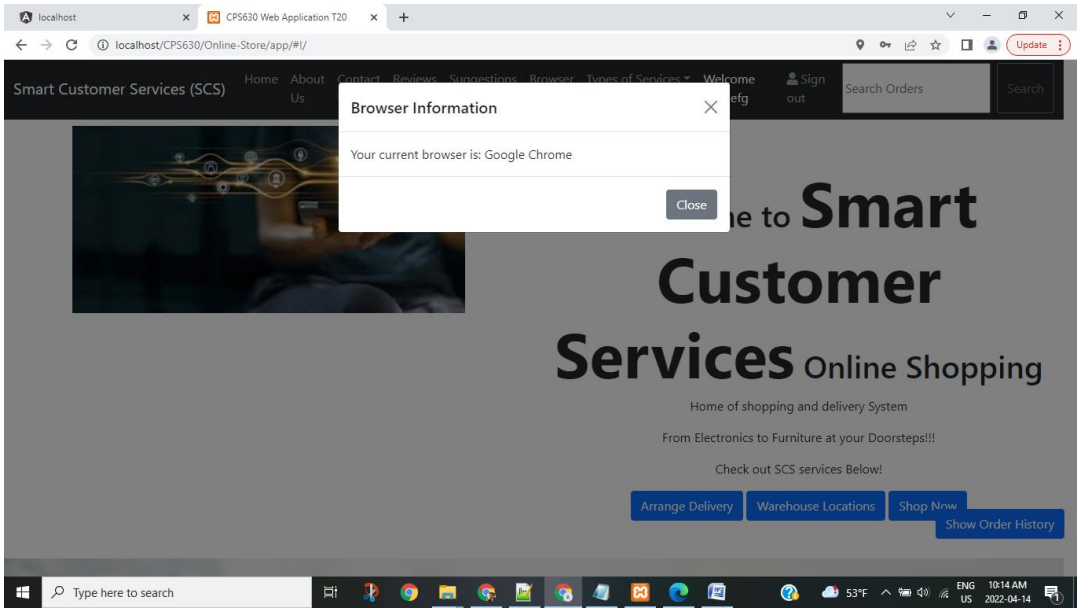
4- **“Review”**

User can able to write a short review and define a ranking number (RN) to the service they received and to the items that they purchased (1-5, from lowest to highest). The “RN” and “Review” for all items are saved in a separate table in the database table Reviews. The “RN” and “Review” for each item then added to each item right away and user could see it.



5- **“Browser”**

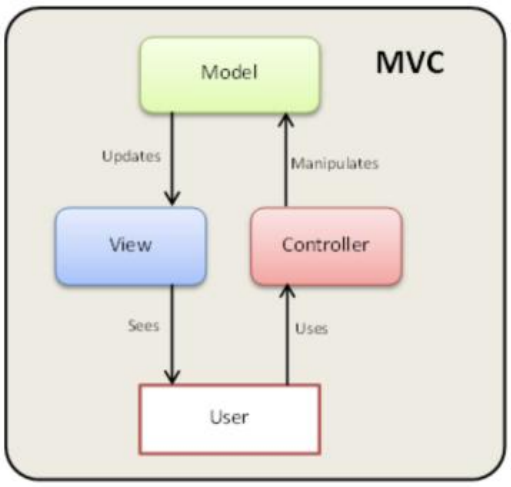
Our application can be open and view on different browsers: Firefox, Internet Explorer, and Chrome with the same layout. The Browser information can be view by the user as pop-up and also while using the system.



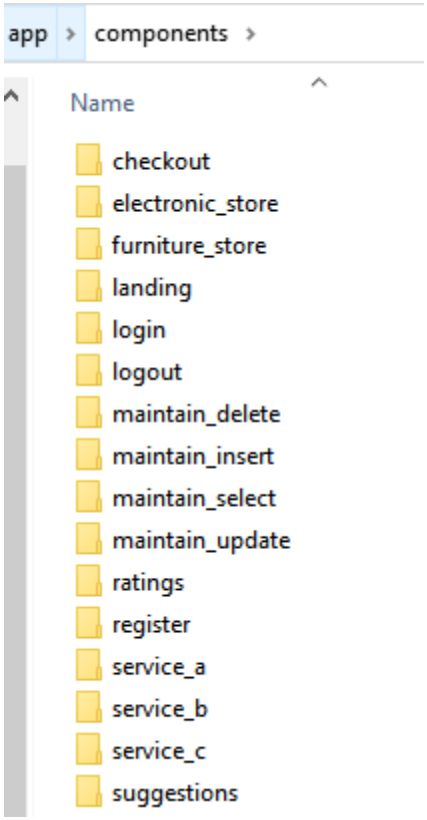
C) Single-Page-Application “SPA” architecture.

Design your system to work based on the “SPA” architecture using either of the following JavaScript **Frameworks: AngularJS.**

First we have defined the layout of our application in separate pages (Views), and then we defined the elements & contents for each page (data Model) that is in the SPA, and finally we defined all events, processes (Controller) to apply sync changes on the SPA (Views, Model).
AngularJS is a full-featured SPA framework that supports the principles behind the Model-View-Controller architecture. MVC design pattern is used to separate the user's view or template from the actual logic of the application. Below is the diagram showing how MVC in Angular works.



The Single Page Application consists of 16 separate pages all routed by a single application using AngularJS. An example snippet and folder structure is shown below:




```
app.config(function($routeProvider, $locationProvider) {

        $routeProvider
        .when('/', {
        templateUrl : './components/landing/home.php',
        controller : 'HomeController'
        })
        .when('/login', {
        templateUrl : './components/login/login.php',
        controller : 'LoginController'
        })
        .when('/register', {
        templateUrl : './components/register/register.php',
        controller : 'RegisterController'
        })
        .when('/ratings', {
        templateUrl : './components/ratings/ratings.php',
        controller : 'ReviewsController'
        })
        .when('/suggestions', {
        templateUrl : './components/suggestions/suggestions.html'
        })
        .when('/service_a', {
        templateUrl : './components/service_a/vehicle.html',
        controller : 'VehicleController'
        })
        .when('/service_b', {
        templateUrl : './components/service_b/service_b.html',
        controller : 'BController'
        })
        .when('/service_c', {
```

The main application has a primary controller and using the $rootScope, we can control certain components from being shown on the screen depending on the current and next routes. For example:

```
app.run(function($rootScope) {
        $rootScope.$on("$locationChangeStart", function(event, next, current) {
                if (next.endsWith('#!/') || next.endsWith('/#about') || next.endsWith('/#contact') || next.endsWith('/#top')) {
                        $('#order_query').addClass("d-flex")
                        $('#order_query').show();
                } else {
                        $('#order_query').removeClass("d-flex")
                        $('#order_query').hide();
                }

        });
        $rootScope.$on("$locationChangeSuccess", function(event, newUrl, oldUrl) {
                console.log("went from " + oldUrl + " to " + newUrl);
                if (oldUrl.endsWith('login') && newUrl.endsWith('app/')) {
                        $rootScope.reloadHeader = true;
                        console.log("making reloadHeader true");
                        window.location.reload();
                }
        });
```

Other controllers are used to make HTTP Requests with our PHP server such as retrieving a list of supported Warehouse Stores:

```
app.controller('CController', function($scope, $http) {
    $scope.message = 'Hello from C';
    $http.get('../server/get_shops.php')
        .then( function (response) {$scope.shops = response.data.records;})
});
```

Using a controller, we can also retrieve/organize information embedded in the HTML such as form information and further POST it to the PHP server such as when creating new orders:

```
app.controller('FurnitureController', function($scope, $http, $timeout, cartService, checkoutInfoService) {
a = new Date();
    document.getElementById('date_1').valueAsDate = a;
    document.getElementById('time_1').value = a.toLocaleTimeString('fr-FR', {hour: '2-digit', minute:'2-digit'});
    document.getElementById('date_2').valueAsDate = a;
    document.getElementById('time_2').value = a.toLocaleTimeString('fr-FR', {hour: '2-digit', minute:'2-digit'});
    //get shop info
    var obj = {};
    obj["id"] = 2;
    $http.post('../server/get_shop_info.php', obj).then( function (response) {
        $scope.shopinfo = response.data;
    });
    //get products
    var obj = {};
    obj["category"] = "furnitureshop";
    $http.post('../server/get_products.php', obj).then( function (response) {
        $scope.products = response.data.records;
    });
    //make draggable?
    $timeout(function() {
        setDraggable();
    }, 1000); // 1 seconds
    $scope.submitCart = function() {
        $scope.checkoutInfo = {
            'start_address' : document.getElementById('start_address_1').value,
            'end_address' : document.getElementById('end_address_1').value,
            'date' : document.getElementById('date_1').value,
            'time' : document.getElementById('time_1').value,
            }
        $scope.productInfo = {
          'product_info' : document.getElementById("cart").childNodes[3].children[1].children[0].innerHTML,
          'product_price' : document.getElementById("cart").childNodes[3].children[1].children[1].innerHTML,
          'product_img' : document.getElementById("cart").childNodes[3].children[0].src,
          'product_id' : document.getElementById("cart").childNodes[3].id,
```

Finally, with AngularJS we can create Services like the Factory Recipes to share code and information across views within our apps. An example is as follows:

```
app.factory('checkoutInfoService', function() {
  var checkoutInfo = [];

  var addcheckoutInfo = function(newObj) {
      checkoutInfo.push(newObj);
  };

  var getcheckoutInfo = function(){
      return checkoutInfo;
  };
  var getNumOfCheckoutInfo = function(){
      return checkoutInfo.length;
  };
  var clearInfo = function() {
      checkoutInfo = [];
  }
  var clearFirst = function() {
      checkoutInfo.shift();
  }
  var clearSecond = function() {
      checkoutInfo.pop();
  }
  return {
    addcheckoutInfo: addcheckoutInfo,
    getNumOfCheckoutInfo: getNumOfCheckoutInfo,
    clearInfo: clearInfo,
    clearFirst: clearFirst,
    clearSecond: clearSecond,
    getcheckoutInfo: getcheckoutInfo
  };
});
```

Implementation:

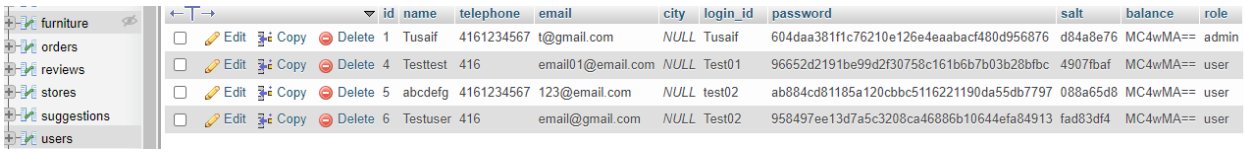
With regards to MVC integration using AngularJS, the Views are routed by the main application and the controllers associated with each route is used to handle user input, retrieve information and make GET/POST requests to the database. For example, when a page is loaded, the controller retrieves necessary information from the database and uses the $scope object to update the View. A simple example would be as follows:

```
obj["category"] = "electronics";
$http.post('../server/get_products.php', obj)
.then( function (response) {$scope.products = response.data.records;})
```

Design for Iteration-IV:

1. Security:

The security issues that we were primarily concerned were with regards to the Integrity Security Principles. This includes data such as a user's login password and their payment information. For the former, we chose to apply a MD5 hash as well as salting the hash on the user's password. As MD5 does not allow for decryption this meant that it would be a safe method to verify the user's password against the hashed value stored in our database to verifying the integrity of the user's password and more importantly to prevent the decryption if the data were to be leaked. For the latter, since we need to retrieve payment information such as a credit card number, we thought it would be more suitable to encode the data using base64 encryption (text to binary). Furthermore, our Maintain pages are set to only allow users with an Admin role to access. This means that no ordinary user can view sensitive information nor modify it.



				id	name	telephone	email	city	login_id	password	salt	balance	role
Edit	Copy	Delete	1	Tusaif	4161234567	t@gmail.com	NULL	Tusaif	604daa381f1c76210e126e4eaabacf480d956876	d84a8e76	MC4wMA==	admin
Edit	Copy	Delete	4	Testtest	416	email01@email.com	NULL	Test01	96652d2191be99d2f30758c161b6b7b03b28bfbc	4907fbaf	MC4wMA==	user
Edit	Copy	Delete	5	abcdefg	4161234567	123@email.com	NULL	test02	ab884cd81185a120cbbc5116221190da55db7797	088a65d8	MC4wMA==	user
Edit	Copy	Delete	6	Testuser	416	email@gmail.com	NULL	Test02	958497ee13d7a5c3208ca46886b10644efa84913	fad83df4	MC4wMA==	user