**NIIT UNIVERSITY**

**NEEMRANA 301705, Dist. Alwar (Rajasthan)**



**Catero**

**Final Report of project**

**Course Name:** Software Engineering **Course code:** CS301

**Year:** 3nd **Semester:** 5th

**Session:** 2018-19 **Discipline:** Computer Science

Contributors:

|  |  |  |
| --- | --- | --- |
| Name | Enroll-Number | Section |
| Ajay Sharma G S | U101116FCS006 | S-1 |
| Desu Lohith | U101116ECS159 | S-2 |
| CVN Koushik | U101116FCS024 | S-2 |
| Ashish Bisht | U101116FCS015 | S-2 |
| Mitul Kumar | U101116FCS068 | S-3 |

Mentors:

*Mr. Amit Kumar Sir.*

*Mr. Chaitanya Surabhi.*

Software Specification (SRS)

CATERO

Date: 24/09/2018

Table of Contents

Table of Contents [ii](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230970)

Revision History [ii](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230971)

1. Introduction [1](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230972)

1.1 Purpose [1](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230973)

1.2 Document Conventions [1](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230974)

1.3 Intended Audience and Reading Suggestions [1](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230975)

1.4 Product Scope [1](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230976)

1.5 References [1](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230977)

2. Overall Description [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230978)

2.1 Product Perspective [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230979)

2.2 Product Functions [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230980)

2.3 User Classes and Characteristics [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230981)

2.4 Operating Environment [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230982)

2.5 Design and Implementation Constraints [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230983)

2.6 User Documentation [2](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230984)

2.7 Assumptions and Dependencies [3](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230985)

3. External Interface Requirements [3](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230986)

3.1 User Interfaces [3](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230987)

3.2 Hardware Interfaces [3](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230988)

3.3 Software Interfaces [3](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230989)

3.4 Communications Interfaces [3](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230990)

4. System Features [4](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230991)

4.1 System Feature 1 [4](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230992)

4.2 System Feature 2 (and so on) [4](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230993)

5. Other Nonfunctional Requirements [4](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230994)

5.1 Performance Requirements [4](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230995)

5.2 Safety Requirements [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230996)

5.3 Security Requirements [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230997)

5.4 Software Quality Attributes [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230998)

5.5 Business Rules [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441230999)

6. Other Requirements [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441231000)

Appendix A: Glossary [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441231001)

Appendix B: Analysis Models [5](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441231002)

Appendix C: To Be Determined List [6](file:///C:\Users\Desu.Lohith\Downloads\Online_testing.docx#__RefHeading___Toc441231003)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| **Catero Team** | **Sept 29th** | **Initial version** | **1.0** |
| **Catero Team** | **Oct 10th** | **Modified Use Case diagram** | **1.1** |

1. **Introduction**

## Purpose

Catero is a catering management system in which we will add a tasty, well-organized menu on our website and enable customers to pre-order your exquisite meals and drinks online. You can offer them different kind of packages as per occasion like wedding, birthday party, etc. Add as many products as you wish, manage orders and process payments. Clients only have to make their choice, select their preferred payment method and wait for their delivery. Catero is designed to simplify the catering order process for our customer. It can be integrated with mobile phone, tablet, and web applications. We are bringing business processes to the digital age.

## Document Conventions

We will use some acronyms throughout this SRS document. This subsection presents definitions for the terms used in this SRS document.

|  |  |
| --- | --- |
| **Term** | **Description** |
| Item | Single serving of food/beverage. |
| Menu | Representation of available items and other options. |
| Order | Consists of one or more items. |
| User | Person who orders/pays from Catero website. |
| Payment | Comprises the total cost of zero or more orders and zero or more tips. |
| Server | Backend computer that hosts the menu & orders. |
| Owner | The owner of the catering company |

The document uses the conventions like database, entity relationships, user interface, use case diagrams and bold letters to emphasis important topics and for all the major functions of the system and Times font of 12 font size indicates the complete description about the whole sub parts.

## Intended Audience and Reading Suggestions

This document is written especially for web developers, researchers, software developers and documentation writers, etc. By reading this document a reader can learn about what the project is implemented for and how it will present the basic information of project. This document has a sequential overview of whole project so if a reader reads the whole document so that reader can get a whole idea about the project.

## Product Scope

This system will consist of a website with AI Chat bot and database to store the orders ordered by the customers. The website is intended to boost the sales of the catering company and create an efficient and sustainable business model of the company. The scope of the product is high because this catering company is already popular in its region and is willing to increase its online customers.

The purpose of the Catering management system is to create a convenient and easy-to-use application for customers, trying to book the orders. The system is based on a relational database with its Catering management. We will have a database server supporting to account all the orders and dates of events as well as we give important notifications. Above all, we hope to provide a comfortable user experience along with the best pricing available.

## References

Reference for web scripting [https://www.w3schools.com](https://www.w3schools.com/)

HTML and CSS: design and build web site by Jon Duckett*.*

Fundamentals of database systems by ramez elmarsi and shamkant b.navathe.

**2. Overall Description**

## 2.1. Product Perspective

The Catero helps the owner to manage the catering process more efficiently and effectively by computerizing the orders. This system replaces the current manual and telephone process for ordering food for events through creating an online system and digitization of menu items with user friendly interface reduces the ambiguity to the user for ordering the food items. The system merges various hardware and software elements and it will evolve over several releases ultimately with chat bot integration.

## 2.2. Product Functions

Whole functions will performed through this project are:

* Registration: Customer should register before login if he wants to visit the Catero.
* Login: Customer logins with valid Username and Password.
* Customer view: Customer can view all the features like menu, gallery, bookings, contact details etc.
* Owner view: Owner can view booking orders, stocks, and maintenance of company.

## 2.3. User Classes and Characteristics

**USERS** are divided into two types:

* **Customers**

Customer will have the access to the functions present on the screen like menu, gallery, bookings, contact details etc. It is considered that the user do have the basic knowledge of operating the internet and to have access to it.

* **Registered Customers:**

Registered Customers are the users who have already registered and they can login into catero website with their credentials.

* **Unregistered Customers:**

These are the users who have not yet registered in catero website. They can fill the registration form and create their credentials and login into catero website.

* **Owner**

The Owner is expected to be familiar with the interface of the tech support system and can view booking orders, maintenance of company.

## 2.4. Operating Environment

This is a web-based system and hence will require the operating environment for a client and server GUI. The server must be able to operate unattended indefinitely. It should not need physical interaction except for upgrades and failure of hardware elements. Backup and recovery should be handled by the DBMS and operating system, or external software running on a timed backup system.

**Operating System**: Windows or UNIX based OS.

**Database**: MySQL.

## 2.5. Design and Implementation Constraints

This system is provisioned to be built on the HTML, CSS, Java Script, which is highly flexible. Decision regarding which database to use should be taken considering the fact that data being stored is large, and the appropriate data management system will yield efficient performance.

## 2.6. User Documentation

* This project comes with a user manual as a guide to its interface options.
* The details of designs, analysis and documentation will be provided along with project.
* Online contact through email and phone number will be provided along with the software.

## 2.7. Assumptions and Dependencies

* The customer and seller must have basic knowledge of computers and English language.
* Each User must have a User ID and password that will stored in encrypted format in database.
* Bot libre platform which acts as an interpreter for AIML.
* Paytm API for advance payment*.*

**3.External Interface Requirements**

There are many types of interfaces as such supported by this software system namely:

User Interfaces, Hardware Interfaces, Software Interfaces and Communications Interfaces.

## 3.1. User Interfaces

The interface will be user friendly. So that every kind of customer can place the food order easily.

Some of the Interfaces for the product would be:

* Login Page
* Registration Form
* View menu, gallery, and contacts.
* Add to cart option for saving the order.
* Payment mode

The user interface for the software shall be compatible to any browser such as google, Internet Explorer, Mozilla Firefox by which user can access to the system.

## 3.2. Hardware Interfaces

It is expected that the user will have a device (computer/mobile) having active Internet connection in order to access the catering website.

## 3.3. Software Interfaces

User only need a web browser which supports HTML5. As for developers would require having database in order to store user information.

## 3.4. Communication Interfaces

Communication function requires the Internet protocol version 4/6 in order to access the website. Protocol that will be used are HTTP/HTTPS and SMTP for communication through mail. SSL, SET, IOTP will be used for online transactions.

**4.System Features**

## 4.1 Registration

**4.1.1 Description and Priority**

It’ s purpose is to create a new customer of the Catero.

A customer must be registered if he/she wants to view menu, gallery etc.

The priority to this feature is high.

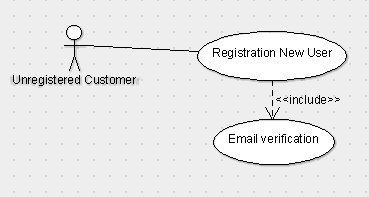
**4.1.2 Stimulus/Response Sequences**

**Stimulus:** User wants to have an account to view menu and place order.

**Response:** The system will display the form with required data to signup.

**Stimulus:** User is unregistered and enters incorrect email or phone-number while registering.

**Response:** Display an error message.



**4.1.3 Functional Requirements**

**REQ -1:** For User the required details are Name, Password, Phone number,E-mail.

**REQ-2:** For admin, details are Name, Username and Password.

**REQ-3:** After any of the above users have signed up a dedicated account has been created for the user and owner.

**4.2 Authentication**

**4.2.1 Description and Priority**

This feature will be used to Log In into the Website (only registered users can use Catero). It is done in order to prevent any misuse of Website.

The Priority for this feature is high.

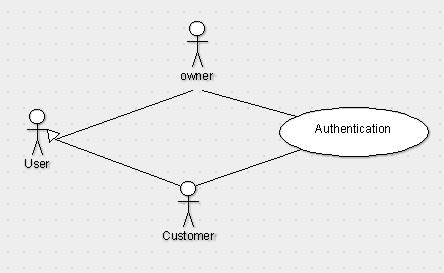
**4.2.2 Stimulus/Response Sequences**

**Stimulus:** User wants to have an account to view and place order.

**Response:** System will display a login page with Username and password.

**Stimulus:** The User is unregistered and tried to Log in or the user is registered and puts incorrect username or password.

**Response:** Display an error message.



**4.2.3 Functional Requirements**

**REQ-1:** For Customer the required details are Username and password to login.

**REQ-2:** For admin the required details are Username and password.

**4.3.View Contact Details:**

**4.3.1.Description and Priority:**

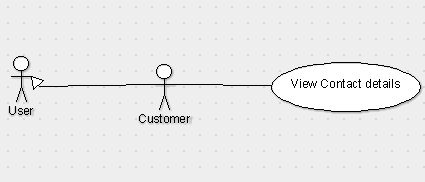
Users can view the contact details of the catering company and contact them for any clarifications.

The priority is moderate.

**4.3.2.Stimulus/Response Sequences**

**Stimulus:**  User wants to see the details of owner.

**Response:** The system will provide a new page containing details of owner.



**4.3.3 Functional Requirements**

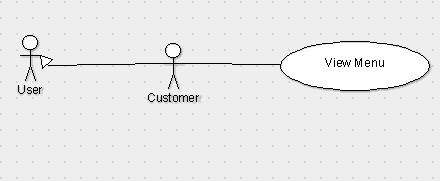
**REQ:** For user to know about the company and to clarify any queries this details will help him out.

**4.4.Standard menu:**

**4.4.1.Description and priority:**

Users can view the items list from the menu available on the home page and select their desired items and add to the cart and place order.

**4.4.2.Stimulus and response:** Customers can select the items from the standard menu which is like a menu provided at hotel.



**4.4.3.Functional Requirements:**

**REQ1:** Standard menu is required to present the customer about all the items which can be done by the catering service.

**REQ2:** Customers should be able to see the standard menu and select items from it.

**4.5.Customized Menu:**

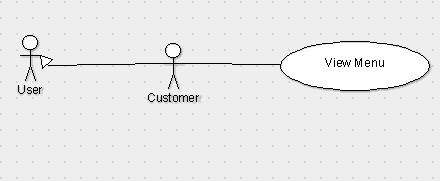
**4.5.1.Description and Priority:**

Users can view the items that are recommended by the owner according to the event and he can choose any of the desired list from the set of option.

**4.5.2.Stimulus/Response Sequences:**

**Stimulus:** Users can view the items list suggested by owner.

**Response:** It displays the list to user to pick any of the items.



**4.5.3.Functional Requirements**

**REQ:** It is useful to those customers who have zero background about food to choose according to the event helps this feature to help them.

**4.6.View Gallery:**

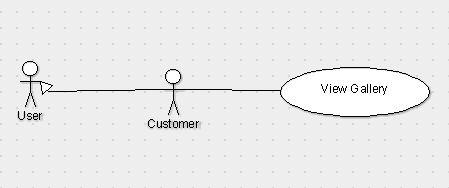
**4.6.1.Description and priority:**

Users can view the gallery which consists pictures of previous events like flower decoration, food items, DJ etc; organized by the company.

**4.6.2.Stimulus/Response Sequences:**

**Stimulus:** Customers select the gallery option to view pictures.

**Response:** The gallery must then be opened for them which contains the photographs of past events.



**4.6.3.Functional Requirements:**

**REQ**: Gallery is required to showcase the previous events organized by the catering service.

**4.7.Sample tasting:**

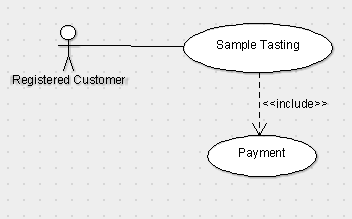
**4.7.1.Description and priority:**

Users can specify an item from their desired menu for sample tasting.

**4.7.2.Stimulus/Response Sequences:**

**Stimulus:** Customers select the few items of their choice for sample tasting.

**Response:** The selected sample tasting items should have been provided to the owner.



**4.7.3.Functional Requirements:**

**REQ1:** Payment should have been done in order to place the sample tasting.

**REQ2:** Customer is required to provide the information about the event and select the menu of his choice.

**4.8.Placing order**

**4.8.1.Description and priority:**

Can place order by choosing the required items from the menu provided for their occasion. Selected menu will be added into the cart. If the User wants to remove the item, it can be done by the remove option.

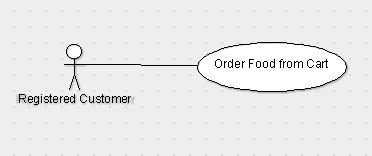
Users can also book additional activities for their event like Flower Decoration, DJ, Photography, Tattoos etc.,

This function carries high priority.

**4.8.2.Stimulus/Response Sequences:**

**Stimulus:** User wants to place the order.

**Response:** The required items will be selected and added to the cart for checkout.



**4.8.3.Functional Requirements:**

**REQ1**: The required items would be presented in the menu provided in the website.

**REQ2:** Customers should be able to design the menu of their choice.

**4.9.Payment:**

**4.9.1.Description and priority:**

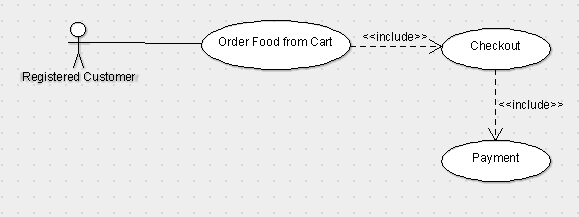
After adding the menu to the cart, user proceeds to payment mode where he can pay the advance amount through PayTm and the remaining by cash.

This function has high priority.

**4.9.2.Stimulus/Response Sequences:**

**Stimulus:** Customer selects to pay the advance payment.

**Response:** paytm information will be provided when he wants to pay the advance payment.



**4.9.3.Functional Requirements:**

**REQ1:** paytm information should be provided as soon as the customer wants to pay.

**REQ2:** In order to pay customer should have completed designing the menu of his choice.

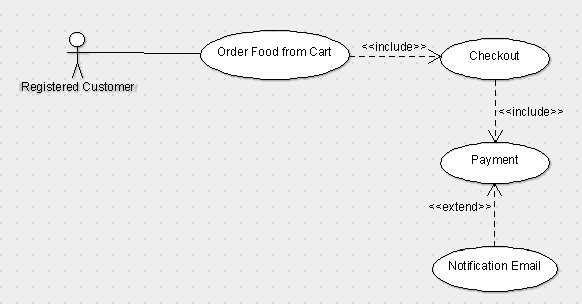
**4.10.Notifications:**

**4.10.1.Description and priority:**

After the order is placed successfully, instant notification is of order details is sent through E-mail.

**4.10.2.Stimulus/Response Sequences:**

**Stimulus and Response:** Notification must be sent to the customer after the payment has been done through the mail which the customer has provided at the time of checkout.



**4.10.3.Functional Requirements:**

**REQ1:** Customer should have completed the advance payment.

**REQ2**: Email should have been provided by the customer.

**4.11.Ask queries through Chat Bot:**

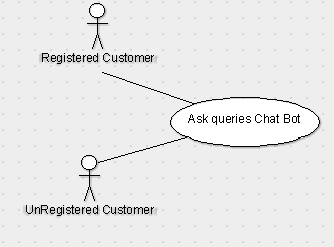
**4.11.1.Description and priority:**

Users should be able to place their queries and get appropriate answer. This function carries least priority.

**4.11.2.Stimulus/Response Sequences:**

**Stimulus:** Users ask about certain queries which they have.

**Response:** Users must be able to get their query resolved in an appropriate way possible.



**4.11.3.Functional Requirements:**

**REQ:** ChatBot is being used in order to resolve user queries instantly to clarify their doubts.

**4.12. Update Menu:**

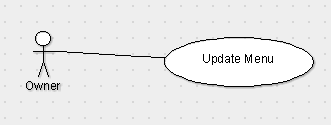
**4.12.1 Description and Priority**

 This function is for owner side of the application.If the owner decides to change the current menu, he can do so with this feature.This function carries highest priority.

**4.12.2 Stimulus/Response Sequences**

**Stimulus:** Owner updates the list of items on menu present on the owner side.

**Response:** It displays menu updated successfully and the menu visible to the customers on catero website is updated.



**4.12.3 Functional Requirement:**

**REQ-1:** Owner can add items from the current menu.

**REQ-2:** Owner can remove items from the current menu. This feature gives an option for the owner to change the menu according to particular season.

**4.13. Add/Remove multiple Categories:**

**4.13.1 Description and Priority**

This function is for owner side of the application. With this function,  owner should be able to add multiple categories to the menu. This  function carries highest priority.

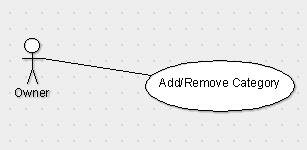
**4.13.2 Stimulus/Response Sequences**

**Stimulus:** Owner adds a new category of food into menu.

**Response:** A new category is added into the current menu.

**Stimulus:** Owner uploads an image for the new category created.

**Response:** New category with an enhanced UI is created.



**4.13.3 Functional Requirements**

**REQ-1:**  Owner should be able to Create various product categories as per their required menu like starters, drinks, desserts, etc.

**REQ-2:** Personalise each category by adding a title and representative image.

**4.14. Calendar of upcoming events:**

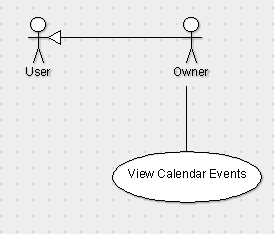
**4.14.1 Description and Priority**

All the upcoming events will appear as a calendar of events which will    allow the owner to stay updated with orders to be prepared and cater them in real time. This function is of moderate priority.

**4.14.2 Stimulus/Response Sequences**

**Stimulus:** Owner clicks on Calendar of Upcoming events.

**Response:** All the upcoming events are displayed sorted in date order.



**4.14.3 Functional Requirements**

**REQ-1:** Owner can view the orders placed by the users for particular date and get ready to prepare delicious food.

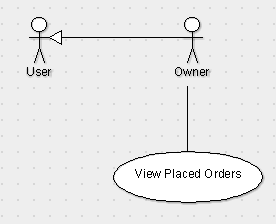
**4.15. Manage Orders:**

**4.15.1 Description and Priority**

 Owner can view the orders placed through catero website. This function carries highest priority.

**4.15.2 Stimulus/Response Sequences**

 All the orders placed by the customers through catero website are made visible to the owner.



**4.15.3 Functional Requirements**

**REQ:**  Owner can check(view) all the latest online orders at a glance.

1. **Other Nonfunctional Requirements**

This subsection presents the non-functional requirements of Catero. The subcategories of other non-functional requirements are performance requirements, safety, security, software quality attributes.

## 5.1.Performance Requirements

* The product will be based on AWS server.
* The product will take initial load time.
* The performance will depend upon hardware components.
* Database will contain user information.
* Payment system will be ensured through Paytm gateway.

## 5.2.Safety Requirements

If there is extensive damage to the database due to sudden failure, such as a database misinterpretation or disk crash, the recovery method restores a past copy of the database that was backed up to storage and reconstructs the same database as of previous one. We will use PreparedStatement which will prevent sql injections.

## 5.3.Security Requirements

The whole system is secured, only registered user and owner can access all the data. Security systems need database storage just like many other applications. However, the special requirements of the security market mean that vendors must choose their database partner carefully.

So, it’s important that we ensure the overall web application and database does not get hacked by the hackers.

## 5.4.Software Quality Attributes

**AVAILABILITY:** The menu should be available to the customers all the time and it should be updated seasonally.

**CORRECTNESS:** The order should reach customer correctly as placed by customer and delivery should be reached from correct source.

**MAINTAINABILITY:** The owner and menu in chargers should maintain correct availability of items.

## 5.5.Business Rules

Users who visit Catero website will be able to login, view items in the menu and add desired menu to the cart and confirm their order by making payment.

Owner of the catering company will be able to view the orders ordered by users through Catero website and process the order.

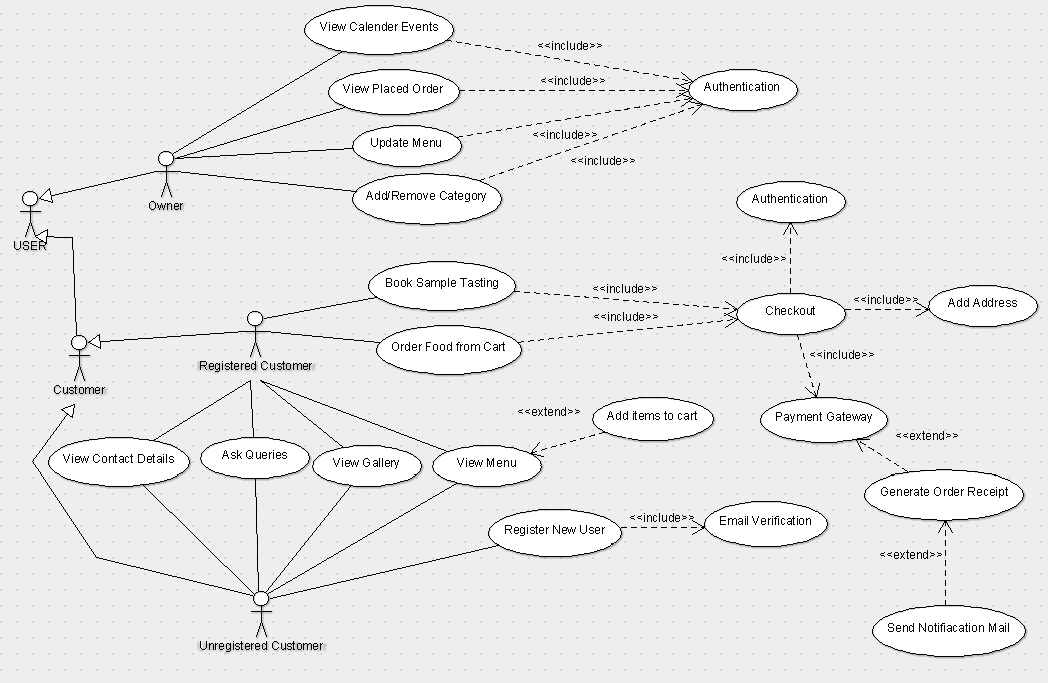
1. **Other Requirements**

All requirements are specified above, there are no other requirements to be mentioned.

**6.1.Appendix : Analysis Models**

**Use Case Diagram:**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.



Software Design Document

**Catero**

Date: 02/11/2018

**Table Of Contents**

**1. Introduction ..................................................................................** Error! Bookmark not defined.

1.1 Purpose of this document ................................................................. **Error! Bookmark not defined.**

1.2 Scope of the development project .................................................... **Error! Bookmark not defined.**

1.3 Definitions, acronyms, and abbreviations ........................................ **Error! Bookmark not defined.**

1.4 References ........................................................................................ **Error! Bookmark not defined.**

1.5 Overview of document ..................................................................... **Error! Bookmark not defined.**

**2. Conceptual Architecture/Architecture Diagram ......................** Error! Bookmark not defined.

2. Structure and relationships ............................................................... **Error! Bookmark not defined.**

2.1 Admin’s side ......................................................................... **Error! Bookmark not defined.**

2.2 User’s side ......................................................................... **Error! Bookmark not defined.**

2.3 User interface issues side ......................................................................... **Error! Bookmark not defined.**

**3. Logical Architecture (Class Diagram, Sequence Diagram, State Diagram)**

**3.1 Logical Architecture Description .............................................**

**3.1.1 Class Diagram Explanation .............................................**

**3.1.2 Sequence Diagram Explanation .............................................**

3.1.2.1 Class name: Login Page ........................................................................... **defined.**

3.1.2.2 Class Name: Admin orders.............................................................. **not defined.**

3.1.2.3 Class Name: Receipt Generator ............................................................. **not defined.**

3.1.2.4 Class Name: Registration.................................................................. **defined.**

3.1.2.5 Class Name: About us .......................................................... **not defined.**

3.1.2.6 Class Name: FAQ....................................................................... **defined.**

3.1.2.7 Class Name: Feedback......................................................... **not defined.**

3.1.2.8 Class Name: Payment.................................................................. **defined.**

**3.1.3 State Diagram Explanation .............................................**

*3.1.3.1 SignIn -----------------------------------------------------------not defined*

*3.1.3.2 SignUp-----------------------------------------------------------not defined*

*3.1.3.3 Sample Testing-----------------------------------------------------------not defined*

*3.1.3.4 Order Placing-----------------------------------------------------------not defined*

*3.1.3.5 FAQ-----------------------------------------------------------not defined*

*3.1.3.5 Gallery-----------------------------------------------------------not defined*

*3.1.3.6 Calendar List-----------------------------------------------------------not defined*

*3.1.3.7 Update Order List-----------------------------------------------------------not defined*

*3.1.3.8 Feedback-----------------------------------------------------------not defined*

**4.0 Execution Architecture ..............................................................** Error! Bookmark not defined.

4.1 Reuse and relationships to other products ........................................ **Error! Bookmark not defined.**

**5.0 Design decisions and tradeoffs ..................................................** Error! Bookmark not defined.

**6.0 Psuedocode for components ..............................................................** Error! Bookmark not defined.

**7.0 Appendices -------------------------------------------------------------------------------Error!Bookmark not defined.**

**Software Design Specification**

**The Software Design Specification Outline**

**1. Introduction**

This document provides documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including sequence diagrams, state diagrams, class diagram and other supporting requirement information.

**1.1 Purpose of this document**

This Software Design Specification (SDS) provides an overview of the proposed Online Catering Management system project design.  It will encompass in detail the basic outline of our project and represent a basis for the development process.  This will also allow critical analysis of the logical and functional aspects of the design before any commitment is made to actual code.

**1.2 Scope of the development project**

The Catero website is intended to boost the sales of the catering company and create an efficient and sustainable business model of the company. The scope of the product is high because this catering company is already popular in its region and is willing to increase its online customers.

The purpose of the Catering management system is to create a convenient and easy-to-use application for customers, trying to book the orders. The system is based on a relational database with its Catering management. We will have a database server supporting to account all the orders and dates of events as well as we give important notifications. Above all, we hope to provide a comfortable user experience along with the best pricing available.

**1.3 Definitions, acronyms, and abbreviations**

IEEE: Institute of Electrical and Electronics Engineers

SDS: Software Design Specification

**1.4 References**

1.4.1 R. S. Pressman, Software Engineering: A Practioner’s Approach, 5th Ed, McGraw-Hill, 2001.

1.4.2 IEEE SDS template

**1.5 Overview of document**

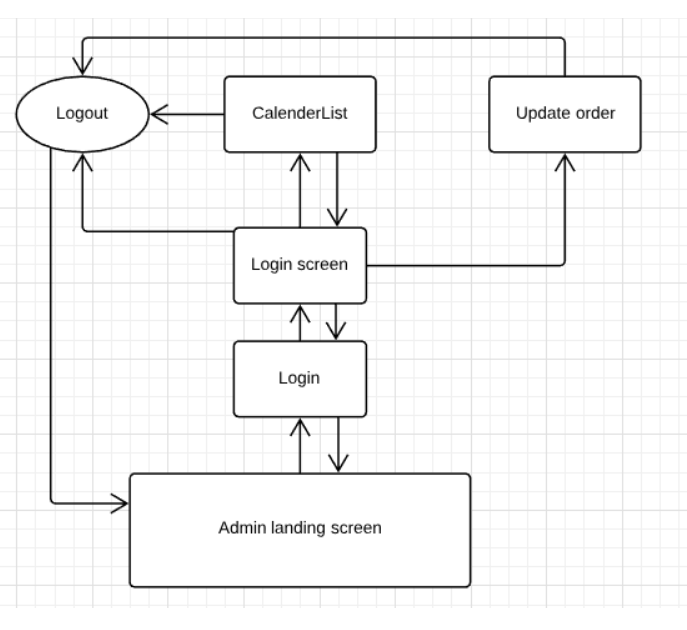
This SDS is divided into seven sections with various sub-sections. The sections of the Software Design Document are:

1. Introduction: describes about the document, purpose, scope of development project definitions and abbreviations used in the document.
2. Conceptual Architecture: describes the overview of components, modules, structure and relationships and user interface issues.
3. Logical Architecture: describes Logical Architecture Description and Components.
4. Execution Architecture: defines the runtime environment, processes, deployment view.
5. Design Decisions and Trade-offs: describes the decisions taken along with the reason as to why they were chosen over other alternatives.

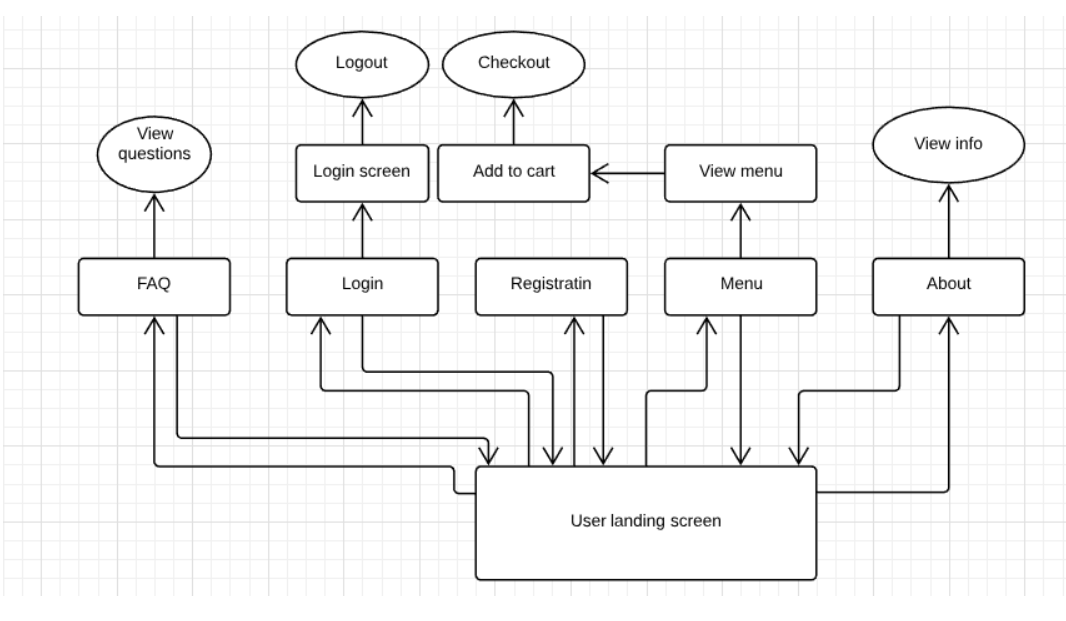
**2. Structure and relationships**

Make clear the interrelationships and dependencies among the various components. Structure charts can be useful here. A simple finite state machine can be useful in demonstrating the operation of the product. Include explanatory text to help the reader understand any charts.

**2.1. Admin’s side:**



**2.2. User’s side:**

****

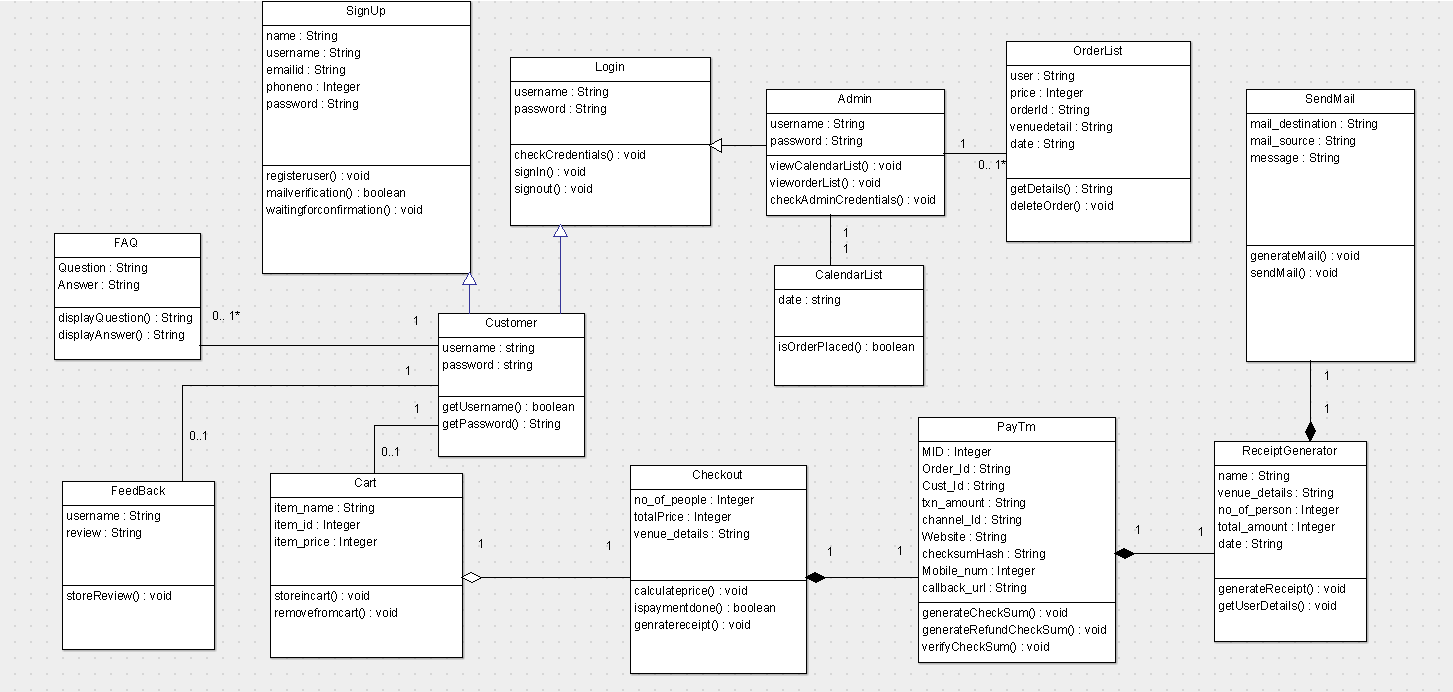
**2.3 User interface issues**

This section will address User Interface issues as they apply to the following hypothetical users of the Online Catering Management System (Catero).

* Let’s assume User A is well versed with using websites over internet. User also know English language. Catero will be using simple words like “Sign-In”, “Sign-Up”, “Feedback”, “Contact”, “Add to cart” etc.
* Let’s assume User B is an old person who isn’t versed with using internet, websites or any other such technology. If B want’s to place an order online he/she may ask someone to open the website and can place an order even through contacting the vendor. The contact information of vendor is given on the home page.

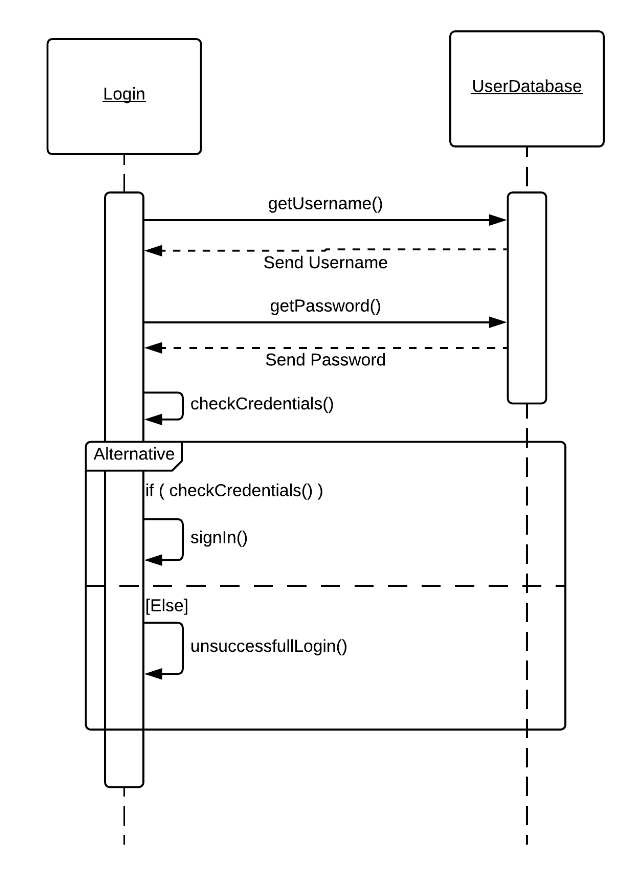
**3. Logical Architecture (Class Diagram, Sequence Diagram, State Diagram)**

**Class diagram:**

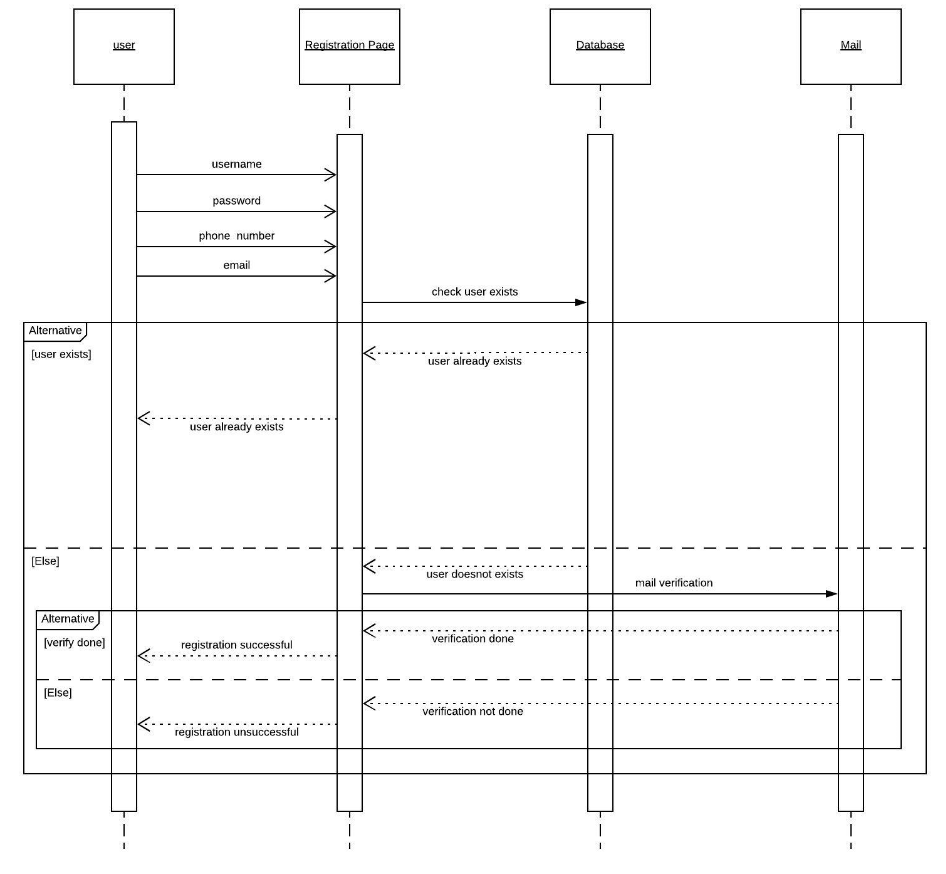
****

**Sequence diagram:**

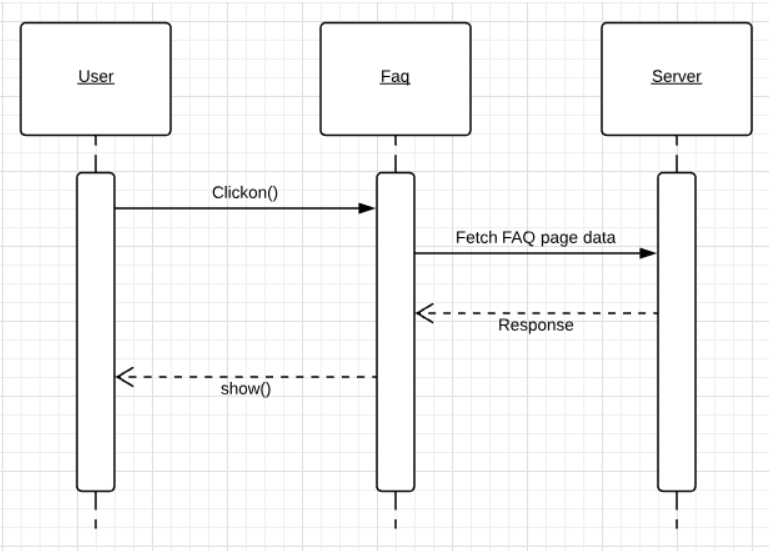
**Sequence diagram: Login Page**

****

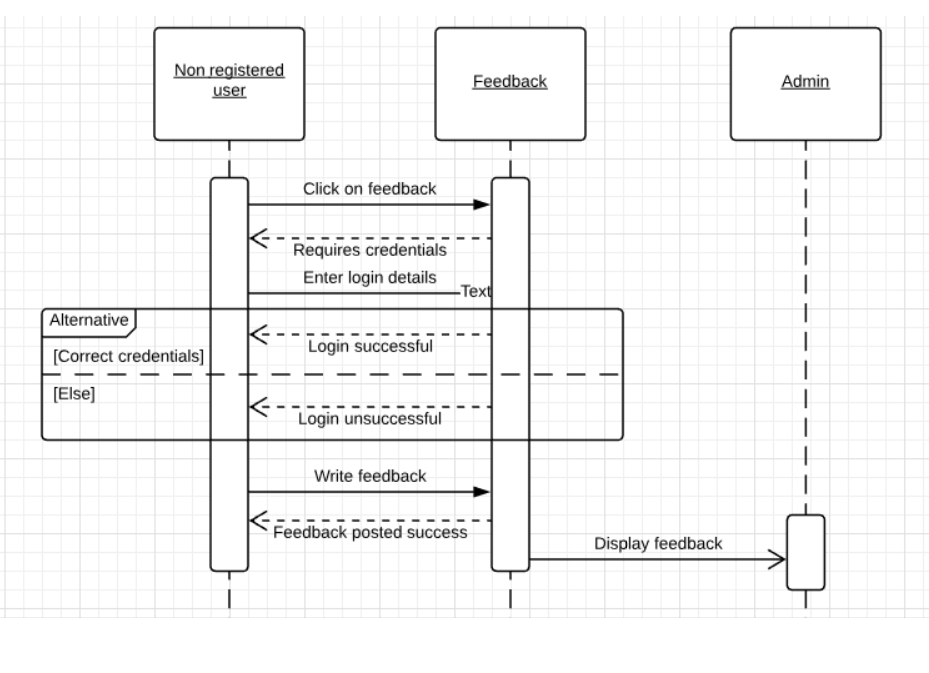
**Sequence diagram: Registration Page**

****

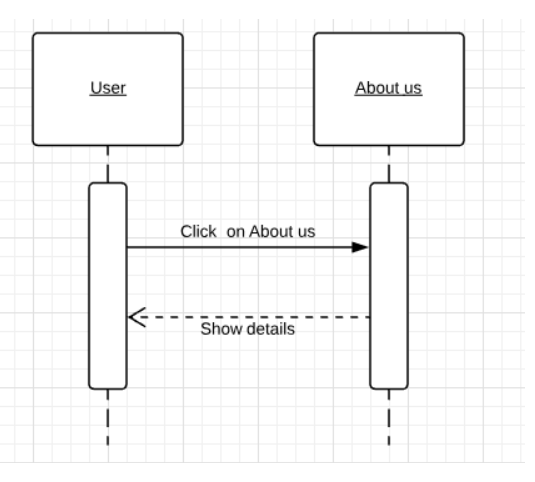
**Sequence diagram: FAQ Page**

****

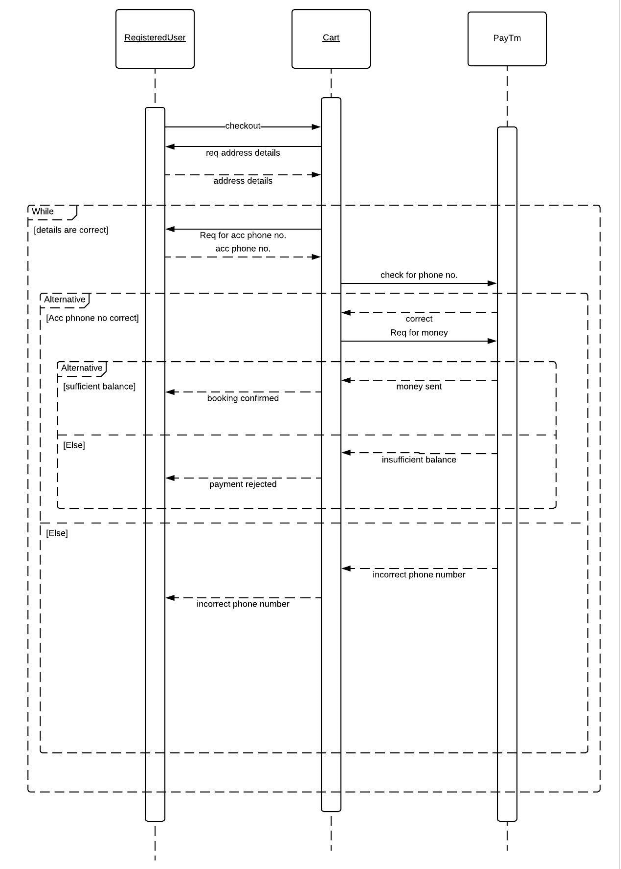
**Sequence diagram: Feedback Page**

****

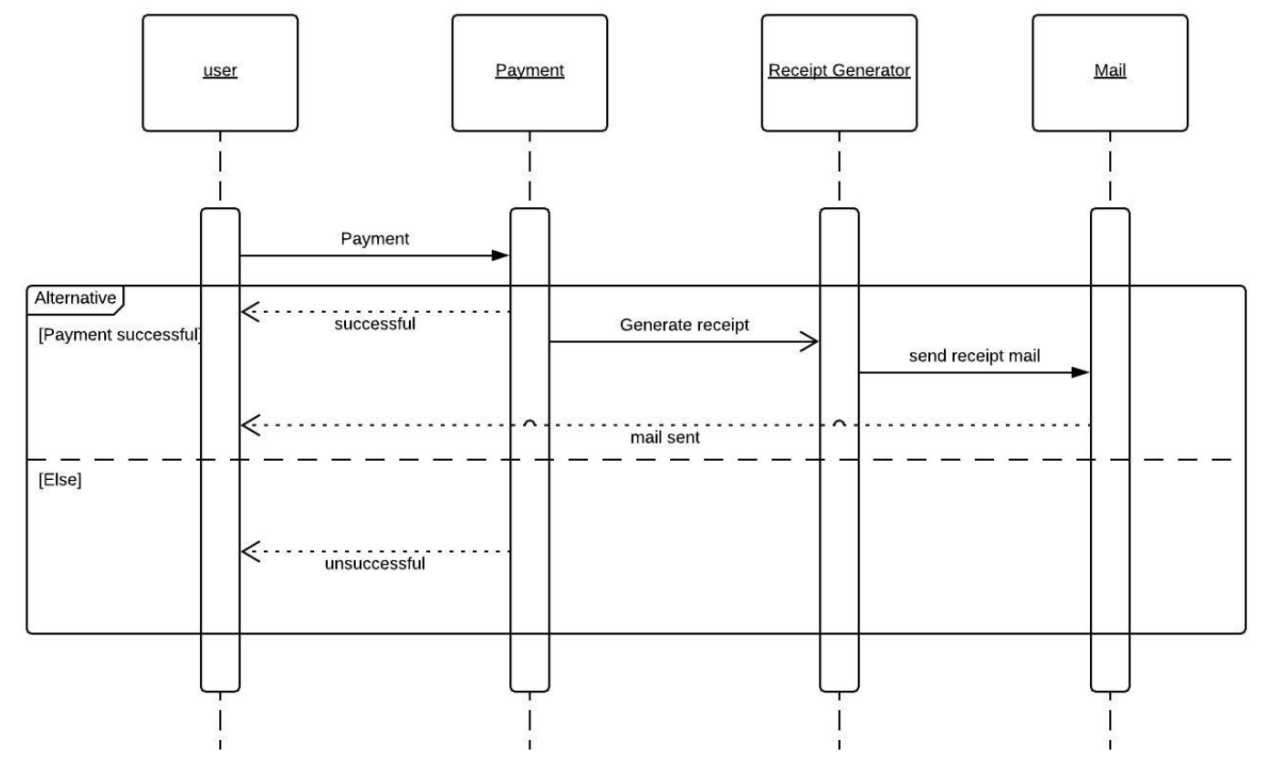
**Sequence diagram: About us**

****

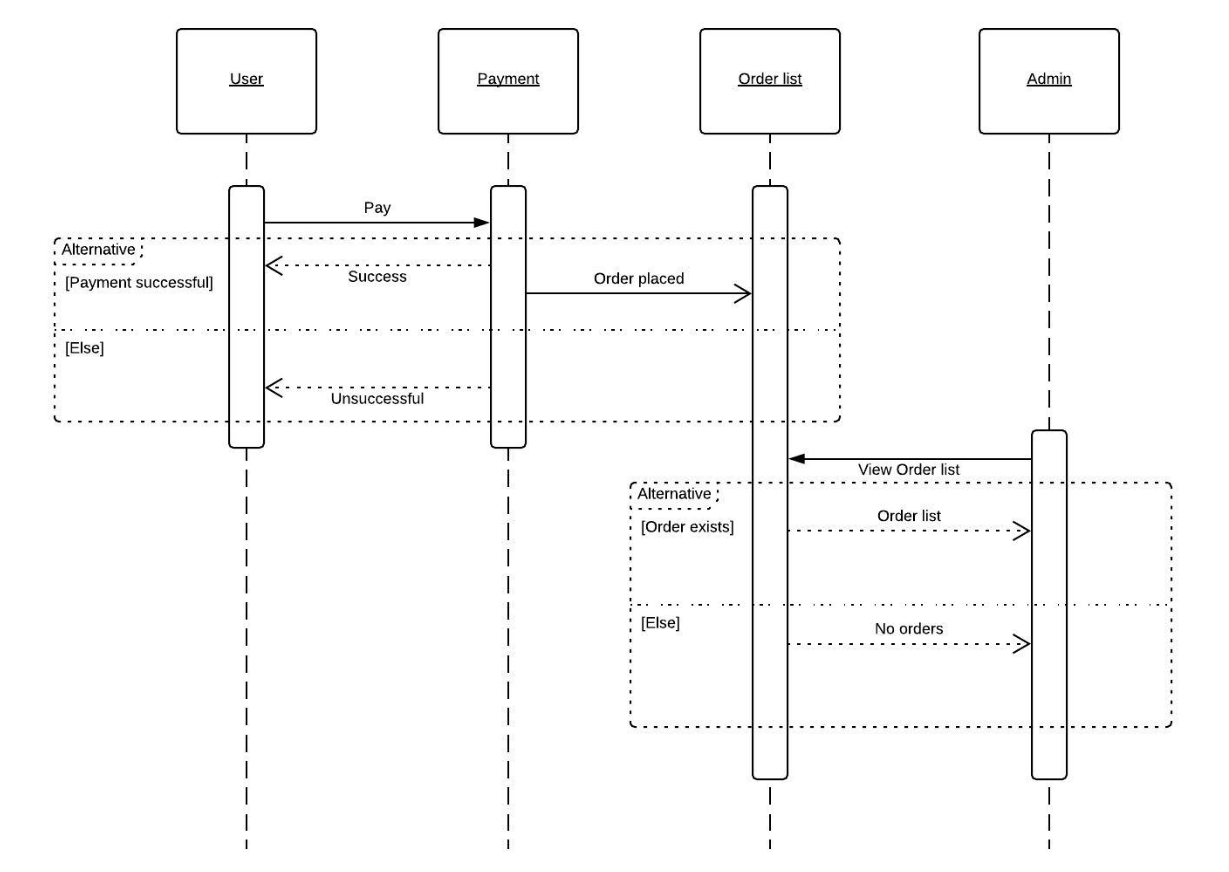
**Sequence diagram: Payment**

****

**Sequence diagram: Receipt Generator**

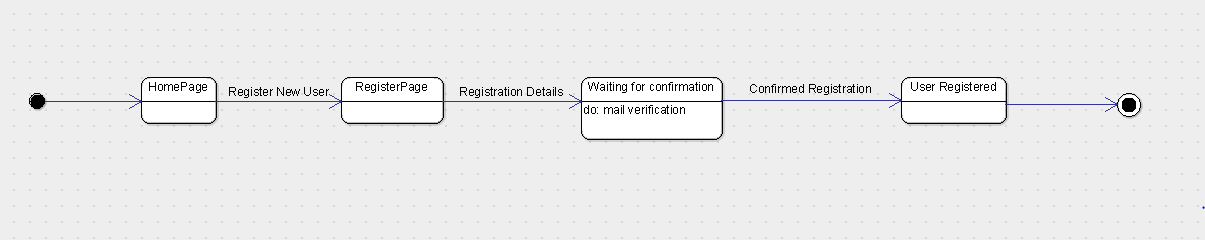
****

**Sequence diagram: Admin Order List**

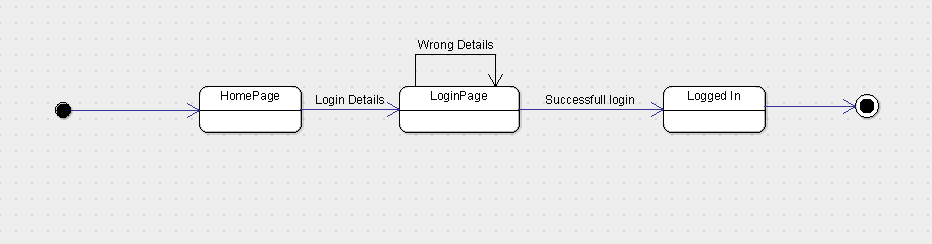
****

**State diagrams:**

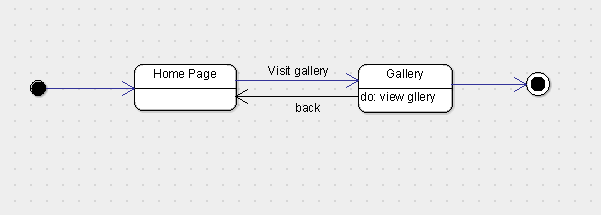
**State diagram: Registration Page**

****

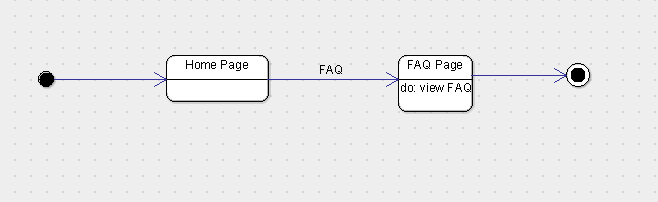
**State diagram: Login**

****

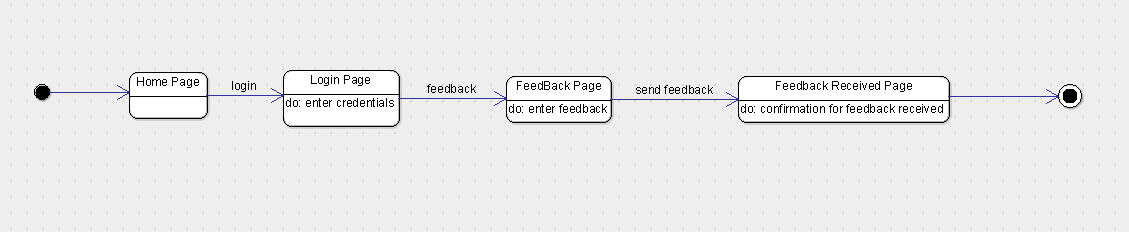
**State diagram: Gallery**

****

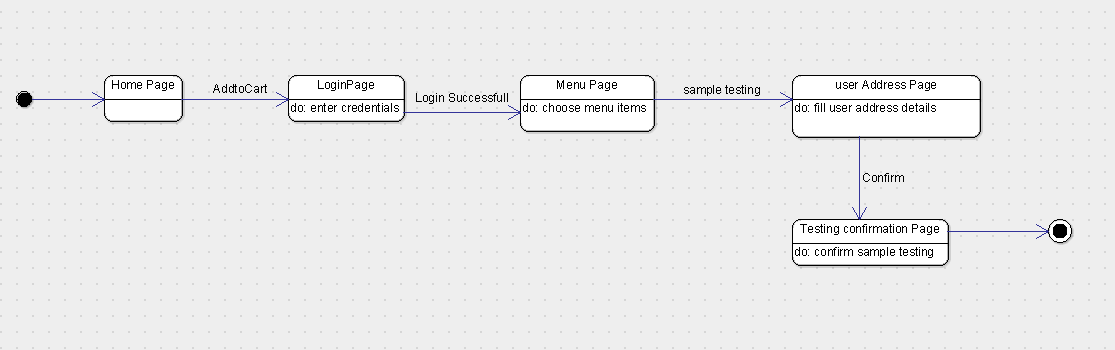
**State diagram: FAQ**

****

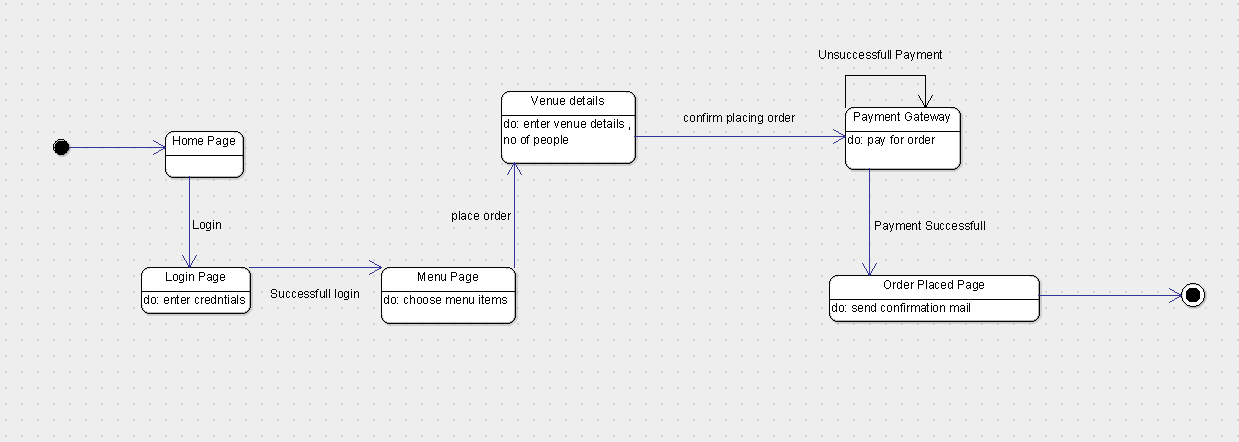
**State diagram: Feed back**

****

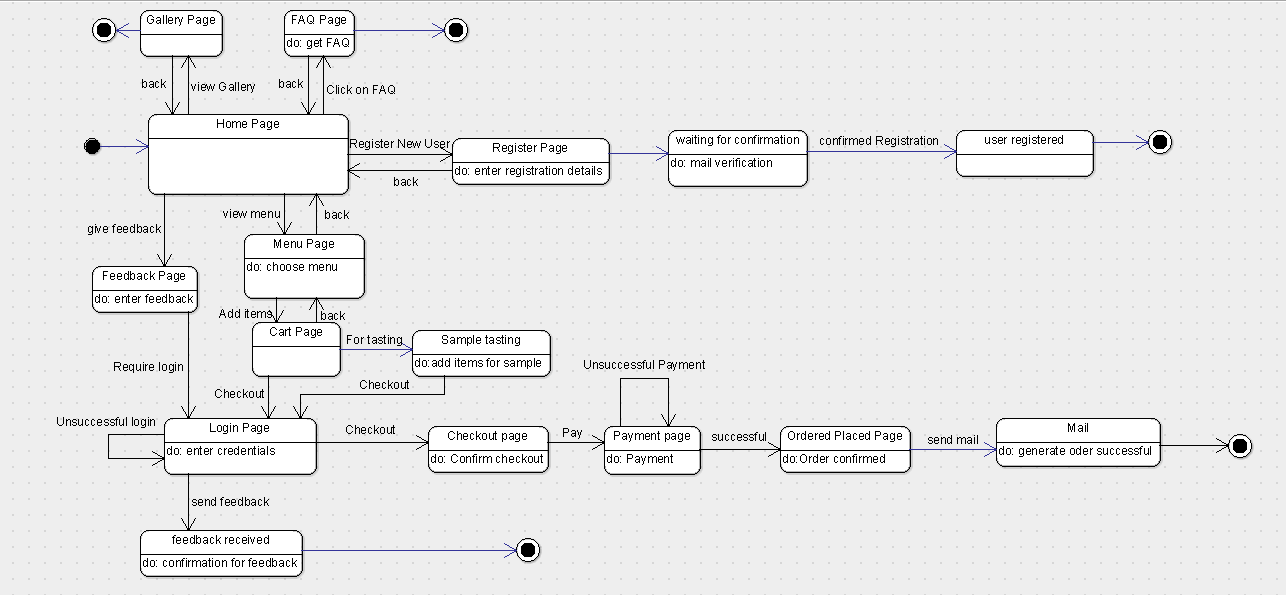
**State diagram: Sample Tasting**

****

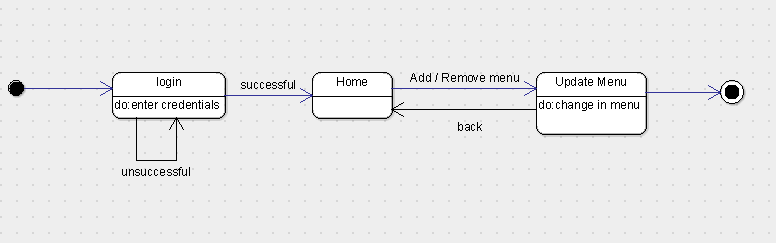
**State diagram: Place Order**

****

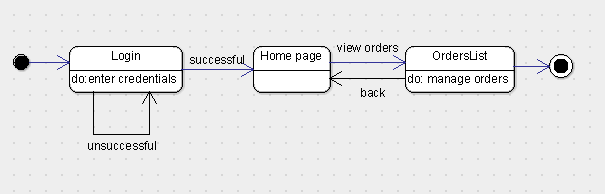
**State diagram: User’s Side (full)**

****

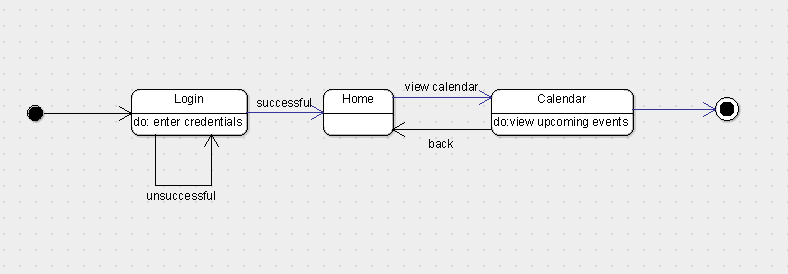
**State diagram: Admin Update Menu**

****

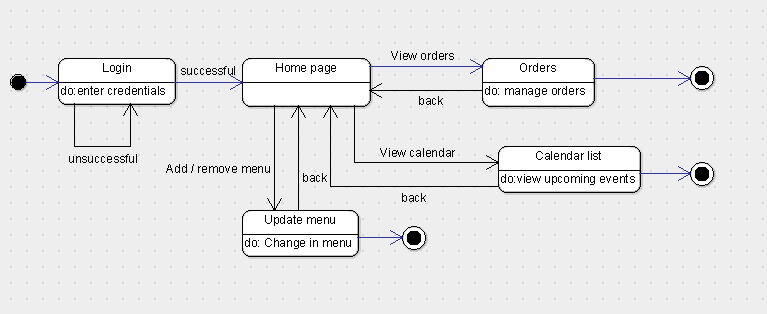
**State diagram: Admin Orders**

****

**State diagram: Admin Calendar’s List**

****

**State diagram: Admin’s Side (full)**

****

**3.1 Logical Architecture Description**

Discuss some details(generic) of Logical Architecture

**3.1.1. State Diagram**

Initial state is being shown by starting with a black dot. Final State is being shown by the black dot surrounded by an empty circle.

**3.1.3.1 Sign In:** On clicking the Sign In button on Home page, it lands on Login Page, which requires the input data from the user. Correct login credentials will lead the login to be successful making the user logged in for taking actions. Incorrect login credentials will redirect the user to same page with an error message.

**3.1.3.2 Sign Up:** On clicking the Sign Up button on Home page, it lands on Register Page, where the user has to input all the fields and click on register button. After the user has click on register button a link will be sent to the user’s mail account which on clicking will take the user to confirmation page where the user will be registered and confirm user for further operation’s onsite. If any field is left blank the user will be given prompt to fill the blank field.

**3.1.3.3 Sample Testing:** On clicking Menu button the user will be redirected to menu page where all the menu items are shown and user can select from given menu and add items to cart. Clicking on sample testing will lead to sign in page which if correctly done will lead to payment page for sample testing. After paying the payment for sample test the user is eligible for sample testing.

**3.1.3.4 Order Placing:** On clicking on Menu button the user will be redirected to menu page where all the menu items are shown and user can select from given menu and add items to cart. Clicking on checkout will lead to sign in page which if correctly done will lead to payment page for the order placed. After the payment is received by vendor the user will be given a prompt that his/her order has been placed. A mail with order receipt will be sent to user’s and vendor’s email id.

**3.1.3.5 FAQ:** On clicking on FAQ button on menu page it will redirect to FAQ page where some of the common questions of the users will be answered.

**3.1.3.5 Gallery:** On clicking on Gallery button on Menu page will redirect to Gallery page where some images are shown.

**3.1.3.6 Calendar List:** On clicking on sign in and after admin has been authorized he/she can see the list of events that will take place according to calendar.

**3.1.3.7 Update Order List:** After successful sign in the owner can update the order list which contains the deleting of order’s that has been completed or cancelled.

**3.1.3.8 Feedback:** Feedback option is available on home page. User will need to sign in first in order to give a feedback. After the submit button is clicked on feedback page the user will be given a prompt that his/her feedback has been recorded.

**3.1.2 Sequence Diagram :**

Arrow line signifies there is a send message taken place. Response is being shown by dotted arrows.

**3.1.2.1 Login Page:** It allows the users to login with their Username and password that are being registered in the database already. It loops being on same page until the correct information is not given.

**3.1.2.2 Admin orders:**

It allows the admin(owner) to view the orders list if they are present after user successfully completes payment and places order. If there are no orders present then order list without any orders(empty list) will be displayed.

**3.1.2.3 Receipt Generator:**

It is used to generate the receipt after the customer successfully makes payment. Then the generated receipt is sent through mail. If the successful payment is not done then receipt for the order will not be generated.

**3.1.2.4 Registration:**

Users will be able to register in the website with the register option available and after they are successfully registered their information is stored in the database so that the next time they enter the website there is no need to register again instead they can directly login.

**3.1.2.5 About us:**

It helps users to know information about the catering service provided and to know how the service operates and detailed information about the catering service.

**3.1.2.6 FAQ:**

FAQ’s are provided by the catering service to know how the service basically operates and information regarding how they manage the catering work allotted to them.

It may contain information about what are all the thing that to be informed to the catering service in prior to the event.

**3.1.2.7 Feedback:**

Feedback option is being provided which is used to share the experience of the clients with the catering service and is useful for the catering service to improve their customer satisfactory rate.

It is basically available for the users who are logged in and are genuine users which makes this function more reliable

**3.1.2.8 Payment:**

Payment option is available which allows users to pay the advance amount.

It can be done only by the registered user after adding items to the cart then heading toward checkout and providing the required details like venue address and phone number. The payment is processed based upon the items the user have chosen and is carried out through paytm gateway.

**3.2 Class name: Login**

**Description**: This class allows the user to enter the system by authenticating the entered credentials.

**3.2.1 Method 1:signIn()**

**Input:** SignIn through user name and password

**Output:** User landing page of login successful

**Method Description:**

This method allows user to login into Catero website with their user name and password. It redirects to CheckCredentials() method to verify the entered user name and password.

**3.2.2 Method 2: CheckCredentials()**

**Input:** user name and password

**Output:** Authentication of user name and password

**Method Description:**

This method checks whether the details entered by the user are correct or not. If the details entered are found incorrect it redirects the user back to the Sign In page.

**3.2.3 Method 3:signOut()**

**Input:** Click on Sign out button

**Output:** Landing to Home page after successful logout.

**Method Description:**

This method allows the user to log out of the catero website after the completion of placing the order.

**3.3 Class name: SignUp**

**Description**: This class allows the user to register in order to enter into the catero website.

**3.3.1 Method 1:registerUser()**

**Input:** name, user name, password, emailId, phone\_no

**Output:** User landing page(Sign In) of registration successful.

**Method Description:**

This method allows user to register on Catero website with their name, user name, password, emailId and phone\_no. After successful registration, user will be redirected to login page in order to make the booking of food.

**3.3.2 Method 2: waitingforconfirmation()**

**Input:** emailId, url

**Output:** Handles sending verification url to the user

**Method Description:**

This method is responsible for sending verification url to the user and waits for the user confirmation in order to successfully complete the Sign Up process of catero website.

**3.3.3 Method 3:mailVerification()**

**Input:** emailId, url

**Output:** Landing to Sign In page after successful verification.

**Method Description:**

This method checks whether the email confirmation is done by the user or not to complete the registration process.

**3.4 Class name: Feedback**

**Description**: This class is utilised for storing the feedback reviews provided by the customers who have utilised the services of the Catero.

**3.4.1 Method 1:storeReview()**

**Input:** user name, review

**Output:** Handles storing of review written by user.

**Method Description:**

This method allows user to give feedback of the services utilised and stores them so that the owner can view them and enhance the quality of the food being prepared.

**3.5 Class name: FAQ**

**Description**: This class is utilised for displaying frequently asked questions and apt answers for those questions.

**3.5.1 Method 1:displayQuestion()**

**Input:** click on FAQ button

**Output:** Launch the activity

**Method Description:**

This method allows user to view the frequently asked questions by the other users while ordering the catering services through Catero website.

**3.5.2 Method 1:displayAnswer()**

**Input:** click on FAQ button

**Output:** Launch the activity

**Method Description:**

This method allows user to view the answers to the frequently asked questions by the other users while ordering the catering services through Catero website.

**3.6 Class name: AddtoCart**

**Description**: This class allows the user to use all the functionalities regarding Shopping Cart.

**3.6.1 Method 1:additemtocart()**

**Input:** item\_name, item\_id, item\_price

**Output:** Handles adding item to cart.

**Method Description:**

This method allows the user to add items into cart from the menu displayed. After adding the items to cart, user can view the cart and proceed to checkout.

**3.6.2 Method 2: storeincart()**

**Input:** item\_name, item\_id, item\_price

**Output:** Handles storing the items added in cart

**Method Description:**

This method is responsible for storing the menu items added into the cart by the user. When the user clicks on the cart button, he will be displayed with all the items added into cart.

**3.6.3 Method 3:removefromcart()**

**Input:** item\_name, item\_id, item\_price

**Output:** Handles removing the items from cart

**Method Description:**

This method allows the user to remove (or) delete the items added from cart.

**3.7 Class name: Checkout**

**Description**: This class contains methods that make the user to check out of the catero system.

**3.7.1 Method 1:calculateprice()**

**Input:** no\_of\_people, totalPrice,venue\_details

**Output:** Handles the calculation of the total price to be paid .

**Method Description:**

This method calculates the total cost to be paid by the customer according to order placed per no\_of\_people, totalPrice of the menu selected and venue.

**3.7.2 Method 2: isPaymentdone()**

**Input:** no\_of\_people, totalPrice,venue\_details

**Output:** Returns a boolean value regarding Payment done.

**Method Description:**

This method checks whether the payment is done or not by the customer.

**3.7.3 Method 3:generatereceipt()**

**Input:** no\_of\_people, totalPrice,venue\_details

**Output:** Handles generation of receipt

**Method Description:**

This method is used for the generation of receipt after the payment is done by the customer.

**3.8 Class name: UserDatabase**

**Description**: This class is required for parsing through the User Database for login verification purpose.

**3.8.1 Method 1:getUsername()**

**Input:** username, password

**Output:** Handles the verification of Login credentials from database.

**Method Description:**

This method checks if the username entered is present in the usernames on the database.

**3.8.2 Method 2: getPassword()**

**Input:** username, password

**Output:** Handles the verification of Login credentials from database.

**Method Description:**

This method checks if the password entered matches with the username entered in the database.

**3.9 Class name: CalendarList**

**Description**: This class enables owner to view the calendar list of upcoming events sorted in date wise manner.

**3.9.1 Method 1: isOrderPlaced()**

**Input:** date

**Output:** Handles the creation of Calendar list of upcoming events.

**Method Description:**

This method is responsible for the creating Calendar list of upcoming events in date wise manner on owner side of the application.

**3.10 Class name: ReceiptGenerator**

**Description**: This class is responsible for the generation of Receipt after the order is placed by the user

**3.10.1 Method 1: getUserDetails()**

**Input:** no\_of\_people, totalPrice, venue\_details, name, date

**Output:** Returns the order details of the user.

**Method Description:**

This method returns the booking details of the order placed by the user.

**3.10.2 Method 2:generatereceipt()**

**Input:** no\_of\_people, totalPrice, venue\_details, name, date

**Output:** Handles generation of receipt

**Method Description:**

This method is used for the generation of receipt after the payment is done by the customer.

**3.11 Class name: SendMail**

**Description**: This class is responsible for sending of Receipt after the order is placed by the user through mail.

**3.11.1 Method 1: generateMail()**

**Input:** message

**Output:** Handles generation of mail content

**Method Description:**

This method is responsible for generating the mail to be sent to the user after the successful placing of order.

**3.11.2 Method 2:sendMail()**

**Input:** message, mail\_source , mail\_destination

**Output:** Handles sending of receipt through mail

**Method Description:**

This method is used for the sending the receipt generated through mail after the payment is done by the customer.

**3.12 Class name: OrderList**

**Description**: This class is responsible for generating the order details after the order is successfully placed.

**3.12.1 Method 1: getUsername()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Handles generation of user details

**Method Description:**

This method is responsible for generating the user details of the order placed.

**3.12.2 Method 2:getPrice()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Handles generation of total price

**Method Description:**

This method is used for getting the total price of order placed.

**3.12.3 Method 3:getOrderPlaced()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Handles getting placed orders

**Method Description:**

This method is used for getting the details of placed orders.

**3.12.4 Method 4:getVenuedetails()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Handles generation of venue to be delivered at for the placed order

**Method Description:**

This method is used for getting the delivery details of the location where food should be delivered.

**3.12.5 Method 5:getDate()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Handles generation of date of the order placed

**Method Description:**

This method is used for getting the date of the order placed in order to sort all the orders in date wise manner.

**3.12.6 Method 6:deleteOrder()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Handles deleting the order

**Method Description:**

This method is used for the deletion of orders by the owner which are already delivered.

**3.12.7 Method 7:viewPlacedOrder()**

**Input:** user, price, orderplaced, venuedetails, date

**Output:** Launch the activity

**Method Description:**

This method is used for viewing all the placed orders on owner side of the application.

**3.13 Class name: PayTm**

**Description**: This class is responsible for payment through PayTm.

**3.13.1 Method 1: generateCheckSum()**

**Input:** M\_id, order\_id, cust\_id, Txn\_amt, channel\_id, website, checkSumHash, mobile\_no

**Output:** Handles payment through PayTm

**Method Description:**

This method is responsible for generating checksum to be sent for Paytm server.

**3.13.2 Method 2:generateRefundCheckSum()**

**Input: Input:** M\_id, order\_id, cust\_id, Txn\_amt, channel\_id, website, checkSumHash, mobile\_no

**Output:** Handles payment through PayTm

**Method Description:**

This method is used for getting the response from the paytm server.

**3.13.3 Method 3:verifyCheckSum()**

**Input: Input:** M\_id, order\_id, cust\_id, Txn\_amt, channel\_id, website, checkSumHash, mobile\_no

**Output:** Handles payment through PayTm

**Method Description:**

This method is used for the verification of the checksum generated.

**3.14. Admin:**

This class is responsible for logging of admin and some of the admin actions.

3.14.1. Method 1: checkAdminCredentials()

Input: None

Output: None

Description: This Method checks admin credentials so that admin can login.

3.1.32 Method 2: viewCalendarList()

Input: None

Output: None

Description: This method is responsible for showing upcoming events to admin via notification on Calendar on Admin dashboard.

3.1.33 Method 3: viewOrderList()

Input: None

Output: None.

Description: This method is responsible for showing all the Placed Order to admin.

**4.0 Execution Architecture**

Runtime environment required is any device like Laptop/Mobile which can open web browser having active internet connection, Eclipse as a deployment platform.

**4.1 Reuse and relationships to other products**

NIL

**5.0 Design decisions and tradeoffs**

The design decision to use two layouts separately for admin and for customers is to provide encapsulation. It may have been possible to get all the information on one screen. However, using two screens will keep the data of admin separate from the data being accessed by Cusomers.

**6.0 Pseudocode for components**

**Login Page**

**Pseudo-Code:**

Input: email,password

Output: Redirects to Home Page after successful login

1. protected void doPost(HttpServletRequest request, HttpServletResponse response) {
2. String u = get EmailId from HTML page
3. String p = get Password from HTML page
4. Mydao md = new Mydao(); creates an object md of Mydao class
5. TRY
6. String name = md.CustomerLogin(u, p);
7. IF name not equal to null then
8. Redirect to index.jsp
9. ELSE
10. RequestDispatcher rd = Dispatches the current page to index.jsp
11. request.setAttribute("msg", "Login Fail,try again");
12. forwards the request to index.jsp file
13. END ELSE
14. END IF
15. CATCH Handles exception

**Register Class**

**Pseudo-Code:**

Input: FirstName, LastName, Gender, EmailId, mobileno, password

Output: Redirects to Login Page after successful Registration

1. protected void doPost(HttpServletRequest request, HttpServletRespons e response) {
2. String firstname=get FirstName from HTML page
3. String lastname=get LastName from HTML page
4. String gender=get Gender from HTML page
5. String emailid=get Email\_id from HTML page
6. String mobileno=get MobileNo from HTML page
7. String password=get Password from HTML page
8. customerbean e=new customerbean(); Creates object e of customerbean class
9. e.setFirstname(firstname); sets the firstname of customer
10. e.setLastname(lastname); sets the lastname of customer
11. e.setGender(gender); sets the gender of customer
12. e.setEmailid(emailid); sets the emailid of customer
13. e.setMobileno(mobileno); sets the mobileno of customer
14. e.setPassword(password); sets the password of customer
15. Mydao m=new Mydao(); creates object for dao class
16. initializes x=m.insertCustomer(e);
17. IF x not equal to 0 then
18. RequestDispatcher rd=dispatches request to login.jsp
19. Sets the msg attribute with value Account Created...
20. rd.forward(request, response);
21. END IF

**Admin Login**

**Pseudo-Code:**

Input: uid,pwd

Output: Redirects to Admin Home Page after successful login

1. protected void doPost(HttpServletRequest request, HttpServletResponse response){
2. PrintWriter out = response.getWriter();
3. String u =get uid from HTML page
4. String p = get pwd from HTML page
5. Mydao md = new Mydao();
6. Initialize x = md.AdminLogin(u, p);
7. IF x not equal to 0 then
8. HttpSession session = request.getSession(); Creates a session for the user
9. session.setAttribute("user", u); User submitted is set in the session
10. redirect response to AdminHomePage.jsp
11. END IF
12. ELSE
13. RequestDispatcher rd = dispatch request to Index.jsp
14. sets the msg attribute with value Login Fail,try again
15. rd.forward(request, response); forwards to index.jsp file
16. END ELSE
17. }

**Add to Cart**

**Pseudo-Code:**

Input: pid,price,quantity

Output: Adds the selected product to cart

1. protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2. String pid, price, quantity = get pid, price, quantity from HTML page
3. initialize ip to null
4. HttpSession hs=request.getSession();Creates a session for the user
5. String user=get FirstName from HTML page
6. IF(user==null)
7. ip=new Mydao().ipAdd();
8. int x=new Mydao().addtocart(pid,quantity,ip);
9. IF x not equal to 0 then
10. Make ArrayList<addproductbean> object(list) = new Mydao().ViewAllProduct();
11. RequestDispatcher rd= Dispatches to shopss.jsp
12. sets the msg attribute with value item added to cart
13. request.setAttribute("data", list);
14. rd.forward(request,response); ); forwards shopss.jsp file
15. END IF
16. ELSE
17. initializes x=new Mydao().addtocart(pid,quantity,user);
18. IF x =! 0 then
19. Make ArrayList<addproductbean>object( list) = new Mydao().ViewAllProduct();
20. RequestDispatcher rd= Dispatches to shopss.jsp
21. request.setAttribute("msg", "item added to cart");
22. request.setAttribute("data", list); Sets the data attribute with list
23. forwards request to shopss.jsp file
24. END IF
25. END ELSE
26. END IF

**Place Order**

**Pseudo-Code:**

Input: FirstName, user,amount

Output: Redirects to OrderMail upon successful placing of order

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. HttpSession hs=request.getSession();
3. String user = get FirstName from HTML page;
4. String Emailid= get user from HTML page;
5. **long** id=System.*currentTimeMillis*();
6. String orderid="Od-Id"+(id+"").substring(5);
7. get amount from HTML page
8. **double** amount=Double.*parseDouble*(request.getParameter("amount"));
9. **initialize** x=**new** Mydao().placeOrder(user,amount,orderid);
10. IF x =! 0 then
11. RequestDispatcher rd=Dispatch to ordermail
12. request.setAttribute("Emailid", Emailid); Sets the Emailid attribute
13. request.setAttribute("orderid", orderid); Sets the orderid attribute
14. request.setAttribute("FirstName",user); Sets the FirstName attribute
15. forwards request to ordermail
16. END IF
17. }

**Check Email**

**Pseudo-Code:**

Input: id

Output: checks if the email already exists or not

1. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. PrintWriter out=response.getWriter(); creates Print writer object out
3. String email = request to get **id** from HTML page
4. String m=checkEmail(email); stores the returned value by calling the checkEmail method
5. out.println(m); Already exist or available
6. }
7. **public** String checkEmail(String email)
8. {
9. Initializes msg to null
   1. **TRY**
   2. {
      1. Initializes x to 0
      2. Class.*forName*("com.mysql.jdbc.Driver");
      3. con=DriverManager.*getConnection*("jdbc:mysql://127.0.0.1:3306/zappy", "root", "lohith@123"); Create database connection
      4. String sql = "select \* from customerdetails where Emailid=?"; sql query to select all details of a particular customer
      5. PreparedStatement ps = con.prepareStatement(sql);
      6. ps.setString(1, email);
      7. ResultSet rs = ps.executeQuery(); Execute statement query
      8. **WHILE**(rs.next())

X=1;

* + 1. **IF** x = 1 then

msg="<font color=red>Already Exist</font>";

* + 1. **ELSE**

msg="<font color=green>Avaliable</font>";

* 1. **Catch** handles Exception

1. return msg;
2. }
3. }

**Show Cart**

**Pseudo-Code:**

Input: FirstName

Output: Displays the cart

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
   * 1. **initialize** ip, user to n**ull**;
     2. **initialize** count=0;
     3. HttpSession hs= request.getSession(); creates session request
     4. String userid=(String)hs.getAttribute("FirstName");
     5. **IF**(userid==**null**)

ip=**new** Mydao().ipAdd();

* + 1. **ELSE**
    2. ip=userid

1. Make ArrayList<addproductbean> object(list) = **new** Mydao().showCart(ip);
2. count=**new** Mydao().count(user);
3. **IF** list not equal to null then
   1. RequestDispatcher rd=dispatches cart.jsp
   2. request.setAttribute("ip", ip); sets ip attribute
   3. request.setAttribute("data", list); sets data attribute to list
   4. request.setAttribute("count", count); sets count attribute
   5. rd.forward(request, response); forwards to cart page
4. END IF
5. **ELSE**
   1. RequestDispatcher rd=Dispatches cart.jsp
   2. Sets the msg to your cart is Empty
   3. rd.forward(request, response); forwards to another cart file
6. END ELSE

}

**Show Product**

**Pseudo-Code:**

Input: None

Output: Displays the Menu Items

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. Make ArrayList<addproductbean> object(list) = **new** Mydao().ViewAllProduct();
3. RequestDispatcher rd=Dispatches shopss.jsp
4. sets data attribute to list
5. forwards the request to shopss.jsp file
6. }

**View Product**

**Pseudo-Code:**

Input: None

Output: redirects to viewallproducts.jsp file

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. Make ArrayList<addproductbean> object(list) to Mydao().ViewAllProduct();
3. RequestDispatcher rd= Dispatches to viewallproduct.jsp
4. sets data attribute value to list
5. rd.forward(request, response); forwards to another viewallproducts page
6. }

**View Product Desciption**

**Pseudo-Code:**

 Input: pid

Output: displays the description of the product

1. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. **int** p= get pid from HTML page
3. Make ArrayList<addproductbean> object(list) to new Mydao().singleproduct(p);
4. **IF** list not equal to null then
5. RequestDispatcher rd=Dispatches to single.jsp
6. request.setAttribute("data", list); sets data attribute value to list
7. rd.forward(request, response);
8. END IF
9. }

**Remove from cart**

**Pseudo-Code:**

Input: pid, FirstName

Output: Removes the selected product from cart

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. HttpSession hs = Retrive seesion object
3. String pid = get pid from HTML page
4. String user = get FirstName from HTML page
5. **IF** list not equal to null then
6. **Initialize** x = **new** Mydao().remove(pid, user);
   1. **IF** x =! null then
   2. Redirect to ShowCart
   3. **else**
   4. Redirect to ShowCart
   5. **else**
   6. String ip = **new** Mydao().ipAdd();
   7. **initializes** x = **new** Mydao().remove(pid, ip);
7. **IF** not equal to null then
   1. Redirect to ShowCart
8. **else**
   1. Redirect to ShowCart
      1. }
9. }

**Order Placed**

**Pseudo-Code:**

Input: None

Output: Displays the placed orders to the owner

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. PrintWriter out=response.getWriter();
3. HttpSession hs=get session object
4. String user= get FirstName from HTML page
5. Make object of ArrayList<ordertable> o= **new** Mydao().DetailAfterPlacingOrder(user);
6. **IF** o != null then
   1. RequestDispatcher rd= Dispatches to OrderPlaced.jsp
   2. Sets data attribute to o
   3. rd.forward(request, response); forwards the request
7. END IF
8. **ELSE**

out.println("error");

1. END ELSE
2. }

**Logout**

**Pseudo-Code:**

Input: none

Output: Invalidates the session and redirects to home page

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. HttpSession session=request.getSession(); creates session object
3. now clearing the session object
4. session.invalidate(); it logouts the account
5. response.addHeader("pragma", "no-cache");
6. response.addHeader("cache-control", "no-store");
7. response.addHeader("expire", "0"); expires the session immediately
8. returing to home page
9. RequestDispatcher rd = Dispatches to index.jsp
10. rd.forward(request, response); forward to another jsp file
11. }

**ForgotPassword**

**Pseudo-Code:**

Input: EmailId

Output: New password is sent to the entered EmailId

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. email = get email\_id from HTML page
3. pass = **new** Mydao().ForgetPassword(email);
4. **if** pass not equal to null then
   * 1. to = email; sends email to particular customer
     2. msg = some suitable message with username
     3. from = "catero@gmail.com"; Sender's email ID needs to be mentioned
     4. **final** String username, password = XXXX; "XXXX" signifies user details
     5. host = creates host
     6. Properties props = **new** Properties();
     7. props enables auth, host and sets port number
     8. Session session = Session.*getInstance*(props, **new** javax.mail.Authenticator() {
     9. **protected** PasswordAuthentication getPasswordAuthentication() {
     10. **return** **new** PasswordAuthentication(username, password);
5. END IF
6. });
7. t**ry** block
   * 1. Message message = **new** MimeMessage(session);
     2. Message obj sets from, recipients, subject, text
     3. Transports the message
     4. Sets msg attribute to “Your Password Has Been Sent” and Dispatches login.jsp
8. **catch** handles messaging exception
9. **ELSE**

sets msg to Email Id does not exsist

1. END ELSE
2. }

**Add Product**

**Pseudo-Code:**

Input: productname,category,productprice,weight,description,image

Output: Adds products to the menu

1. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. Initiliazes productname, category, productprice, weight, description, image =**null**;
3. **boolean** isMultipart = ServletFileUpload.*isMultipartContent*(request);
4. **if** !isMultipart then
5. **return**;
6. Make factory obj for DiskFileItemFactory
7. Sets threshold size and upload size to max
8. **try** then
9. List fileItems = upload.parseRequest(request);
10. Iterator i = fileItems.iterator();
    1. **while** ( i.hasNext())
       1. {
       2. FileItem fi = (FileItem)i.next();
       3. **if** fi.isFormField () then

Checks for productname, category, productprice, weight, description

* + 1. else
       1. {
       2. String fieldName = fi.getFieldName();
          1. **If** fieldName.equals("ima") then
          2. Makes servlet config object sc
          3. image=fi.getName();
    2. File f = **new** File(sc.getServletContext().getRealPath("/")+"imagesap") ; Makes file obj
       1. **If** !f.exists() then
          1. f.mkdir();
    3. File = **new** File(sc.getServletContext().getRealPath("/")+"imagesap/"+image) ;
       - 1. fi.write( file ) ;
         2. out.println("Uploaded Filename: " +image + "<br>");
         3. System.***out***.println("PATH="+file.getPath());
  1. END WHILE
     1. END IF
     2. END ELSE
     3. END IF

1. **catch** Handles exception
2. addproductbean e=**new** addproductbean();
3. e object adds the productname category productprice weight description image
4. Mydao m=**new** Mydao(); creates database object m
5. Adds the product in the database and displays Product added
6. }
7. }

**Delete Product**

**Pseudo-Code:**

Input: productname

Output: Selected product is deleted and displays all available products

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. PrintWriter out=response.getWriter(); creates writer object out
3. String acn=request.getParameter("productname"); Gets product name
4. **try** block
   1. Register the database driver by using
   2. Class.*forName*("com.mysql.jdbc.Driver");
   3. Create database connection
   4. Connection con=DriverManager.*getConnection*("jdbc:mysql://127.0.0.1:3306/zappy", "root", "lohith@123");
5. Deletes product name from the database
6. PreparedStatement ps=con.prepareStatement("delete from addproduct where productname=?");
7. ps.setString(1, acn);
8. Execute statement query ps.executeUpdate();
   1. **if** x not equal to null then
      1. RequestDispatcher rd=Dispatches to viewallproduct.jsp
      2. Make ArrayList<addproductbean>object( list) for ViewAllProduct();
      3. Set data value attribute to list
      4. request.setAttribute("msg", "Product Deleted of ProductName is = "+acn);
      5. forwards the request to view all products page
   2. END IF
9. con.close();
10. END TRY
11. **Catch Handles e**xception
12. }

**OrderMail**

**Pseudo-Code:**

Input: EmailId, FirstName, oid

Output: Order Confirmation message is sent to user mail

1. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {
2. String to = gets email id of recipient
3. String sub="Order Confirmation";
4. String name=gets first name
5. String Oid= gets orderid
6. String msg= sends order placed message to the customer
7. String from = Sender's email ID needs to be mentioned
8. **final** String username = username to be mentioned
9. **final** String password = password of sender
10. String host = mention host name
11. Properties props = **new** Properties();
12. props enables auth, host and sets port number
13. Make Session object(session) = Session.*getInstance*(props, **new** javax.mail.Authenticator() {
14. **protected** PasswordAuthentication getPasswordAuthentication() {
15. **return** **new** PasswordAuthentication(username, password);
16. }
17. });
18. **try** block
    * 1. Message message = **new** MimeMessage(session);
      2. Message obj sets from, recipients, subject, text
      3. Transports the message to recipient
      4. PrintWriter out=response.getWriter(); creates writer object
      5. Displays message sent successfully
      6. Redirects to Orderplaced
19. END TRY

**catch** Handles exception

}

**7.0 Appendices (if any)**

**NIL**

Software Metrics Document

**Catero**

Date: 13/11/2018

**Web Code Metrics - Catero**

**Web Metric at System Level** Catero

Tools used for metrics:

1) Source Monitor

2)Cyvis metric tool

**1) Source Monitor**

The freeware program **SourceMonitor** lets you see inside your software source code to find out how much code you have and to identify the relative complexity of your modules. For example, you can use SourceMonitor to identify the code that is most likely to contain defects and thus warrants formal review.

|  |  |
| --- | --- |
| Baseline | Description |
| Files # | 20 |
| Lines # | 1163 |
| Statements # | 737 |
| % branches | 6.4 |
| Calls # | 340 |
| % Comments | 24.3 |
| Classes # | 22 |
| Methods/Class # | 2.77 |
| Avg.statements/methods | 6.15 |
| Max complexity | 14\* |
| Max depth | 6 |
| Avg .depth | 1.51 |
| Avg.complexity | 1.79\* |

|  |  |
| --- | --- |
| Invalidate.java | Description |
| Lines # | 42 |
| Statements # | 24 |
| % branches | 0.0 |
| Calls # | 9 |
| % Comments | 33.3 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 3 |
| Max complexity | 1\* |
| Max depth | 2 |
| Avg .depth | 0.92 |
| Avg.complexity | 1.00\* |

|  |  |
| --- | --- |
| OrderHistory.java | Description |
| Lines # | 43 |
| Statements # | 24 |
| % branches | 0 |
| Calls # | 7 |
| % Comments | 34.9 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 2 |
| Max complexity | 1\* |
| Max depth | 2 |
| Avg .depth | 0.67 |
| Avg.complexity | 1.00\* |

|  |  |
| --- | --- |
| OrderMail.java | Description |
| Lines # | 80 |
| Statements # | 56 |
| % branches | 1.8 |
| Calls # | 28 |
| % Comments | 20.0 |
| Classes # | 2 |
| Methods/Class # | 3 |
| Avg.statements/methods | 4 |
| Max complexity | 3\* |
| Max depth | 3 |
| Avg .depth | 1.03 |
| Avg.complexity | 1.67\* |

|  |  |
| --- | --- |
| OrderPlaced.java | Description |
| Lines # | 50 |
| Statements # | 31 |
| % branches | 6.5 |
| Calls # | 11 |
| % Comments | 28.0 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 4 |
| Max complexity | 3\* |
| Max depth | 3 |
| Avg .depth | 1.03 |
| Avg.complexity | 1.67\* |

|  |  |
| --- | --- |
| PlaceOrder.java | Description |
| Lines # | 50 |
| Statements # | 31 |
| % branches | 3.2 |
| Calls # | 16 |
| % Comments | 28.0 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 5 |
| Max complexity | 2\* |
| Max depth | 3 |
| Avg .depth | 1.26 |
| Avg.complexity | 1.33\* |

|  |  |
| --- | --- |
| RemoveFromCart.java | Description |
| Lines # | 54 |
| Statements # | 33 |
| % branches | 18.2 |
| Calls # | 15 |
| % Comments | 25.9 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 6 |
| Max complexity | 7\* |
| Max depth | 4 |
| Avg .depth | 1.67 |
| Avg.complexity | 3\* |

|  |  |
| --- | --- |
| ShowCart.java | Description |
| Lines # | 62 |
| Statements # | 41 |
| % branches | 9.8 |
| Calls # | 18 |
| % Comments | 22.6 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 7.67 |
| Max complexity | 5\* |
| Max depth | 3 |
| Avg .depth | 1.46 |
| Avg.complexity | 2.33\* |

|  |  |
| --- | --- |
| ShowCart.java | Description |
| Lines # | 62 |
| Statements # | 41 |
| % branches | 9.8 |
| Calls # | 18 |
| % Comments | 22.6 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 7.67 |
| Max complexity | 5\* |
| Max depth | 3 |
| Avg .depth | 1.46 |
| Avg.complexity | 2.33\* |

|  |  |
| --- | --- |
| ShowProduct.java | Description |
| Lines # | 41 |
| Statements # | 23 |
| % branches | 0 |
| Calls # | 7 |
| % Comments | 34.1 |
| Classes # | 1 |
| Avg.statements/methods | 2 |
| Max complexity | 1\* |
| Max depth | 2 |
| Avg .depth | 0.70 |
| Avg.complexity | 1.00\* |

|  |  |
| --- | --- |
| Statements # | 24 |

|  |  |
| --- | --- |
| ViewRegisteredCustomer.java | Description |
| Lines # | 42 |
| Statements # | 24 |
| % branches | 0.0 |
| Calls # | 7 |
| % Comments | 33.3 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 2 |
| Max complexity | 1\* |
| Max depth | 2 |
| Avg .depth | 0.67 |
| Avg.complexity | 1\* |

|  |  |
| --- | --- |
| ViewProduct.java | Description |
| Lines # | 41 |
| Statements # | 23 |
| % branches | 0 |
| Calls # | 7 |
| % Comments | 34.1 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 2 |
| Max complexity | 2\* |
| Max depth | 2 |
| Avg .depth | 0.7 |
| Avg.complexity | 1\* |

|  |  |
| --- | --- |
| AddProduct.java | Description |
| Lines # | 127 |
| Statements # | 90 |
| % branches | 14.4 |
| Calls # | 57 |
| % Comments | 9.4 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 20.67 |
| Max complexity | 14\* |
| Max depth | 6 |
| Avg .depth | 2.51 |
| Avg.complexity | 5.33\* |

|  |  |
| --- | --- |
| AddToCart.java | Description |
| Lines # | 64 |
| Statements # | 43 |
| % branches | 9.3 |
| Calls # | 25 |
| % Comments | 20.3 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 8.33 |
| Max complexity | 5\* |
| Max depth | 4 |
| Avg .depth | 1.84 |
| Avg.complexity | 2.33\* |

|  |  |
| --- | --- |
| AddminLogin.java | Description |
| Lines # | 51 |
| Statements # | 32 |
| % branches | 6.3 |
| Calls # | 15 |
| % Comments | 27.5 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 5 |
| Max complexity | 3\* |
| Max depth | 3 |
| Avg .depth | 1.25 |
| Avg.complexity | 1.67\* |

|  |  |
| --- | --- |
| CustomerGmail.java | Description |
| Lines # | 41 |
| Statements # | 23 |
| % branches | 0 |
| Calls # | 9 |
| % Comments | 34.1 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 3 |
| Max complexity | 1\* |
| Max depth | 2 |
| Avg .depth | 0.96 |
| Avg.complexity | 1\* |

|  |  |
| --- | --- |
| CustomerLogin.java | Description |
| Lines # | 56 |
| Statements # | 35 |
| % branches | 8.6 |
| Calls # | 17 |
| % Comments | 26.8 |
| Classes # | 1 |
| Methods/Class # | 3 |
| Avg.statements/methods | 6.33 |
| Max complexity | 4\* |
| Max depth | 4 |
| Avg .depth | 1.72 |
| Avg.complexity | 2\* |

**2) Cyvis metric tool**

Cyvis metric tool has been used for the project ‘Catero’ to calculate the complexities of the web application.

The following document contains the report generated by Cyvis metric tool which consists tables of the following:

* Total number of classes, total number of methods in each class and in whole project.
* Total number of lines that each class consists of.
* Total cyclomatic Complexities for each method in a class.
* Graphical representation of above data.

**CyVis Output**

controller,AddProduct,doPost,15,290,,doGet,3,12,,<init>,1,14,

controller,Forgetpassword,doGet,5,162,,doPost,3,8,,<init>,1,6,

controller,OrderMail,doGet,4,141,,doPost,3,8,,<init>,1,6,

controller,AddToCart,doGet,6,118,,doPost,3,8,,<init>,1,6,

controller,ShowCart,doGet,5,96,,doPost,3,8,,<init>,1,6,

controller,CustomerRegistration,doPost,4,90,,doGet,3,12,,<init>,1,6,

controller,CheckEmail,checkEmail,4,71,,doPost,3,21,,doGet,3,8,,<init>,1,6,

controller,CustomerLogin,doPost,5,81,,doGet,3,12,,<init>,1,6,

controller,DeleteProduct,doGet,5,82,,doPost,3,8,,<init>,1,6,

controller,PlaceOrder,doGet,4,81,,doPost,3,8,,<init>,1,6,

controller,RemoveFromCart,doGet,6,71,,doPost,3,8,,<init>,1,6,

controller,AdminLogin,doPost,4,61,,doGet,3,12,,<init>,1,6,

controller,OrderPlaced,doGet,4,48,,doPost,3,8,,<init>,1,6,

controller,ViewProductDescription,doPost,4,40,,doGet,3,12,,<init>,1,6,

controller,CustomerGmailLogin,doGet,3,42,,doPost,3,8,,<init>,1,6,

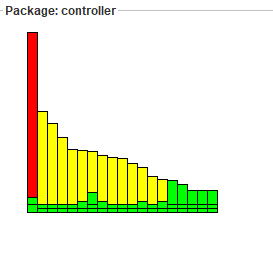
controller,InvalidateJava,doGet,3,35,,doPost,3,8,,<init>,1,6,

controller,ViewProduct,doGet,3,24,,doPost,3,8,,<init>,1,6,

controller,ViewAllRegisteredCustomer,doGet,3,24,,doPost,3,8,,<init>,1,6,

controller,ShowProduct,doGet,3,24,,doPost,3,8,,<init>,1,6,

**Diagramatic Representation:**



**Test Document**

**Manual testing**

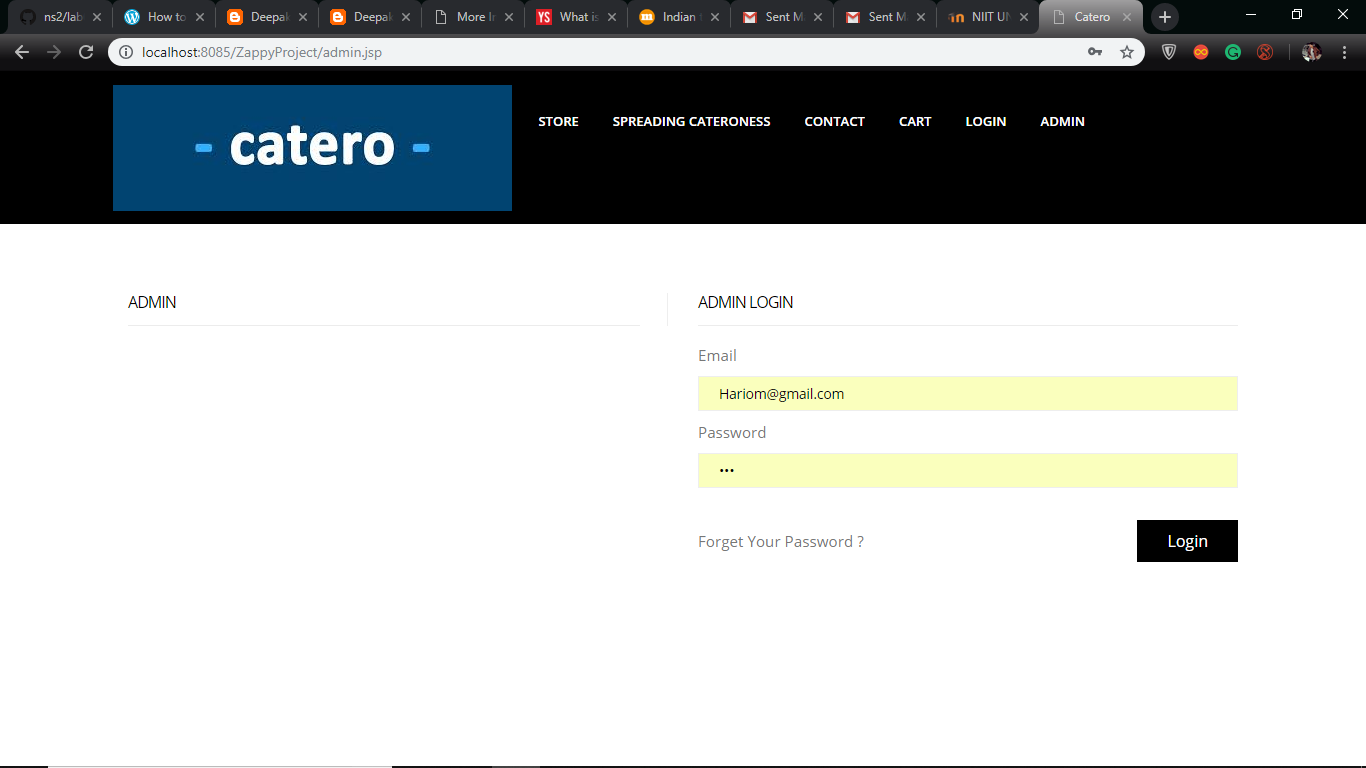
Introduction to test plan: This document is intended to give a complete planning of a systematic strategy for software testing of Catero website. Catero website is composed of numerous features. This test plan is actually designed to ensure those features work up to the mark. Both directly and indirectly affected elements will be addressed here. Manual Testing will be done to ensure the correctness of various parts of the code using test cases generated by the tester. In following table, we will discuss the functions and its expected and real output of our project.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Requirement ID | Class module name | Method name | Input parameters | Expected Output | Real Output | Status(pass/fail) |
| 1 | Customer registration |  | First name, Last name, Gender, Email id, phone no, password | Registration success (Redirects to Login page) | Registration success (Redirects to Login page) | pass |
| 2 | Check email |  | New Email id | Displays email id available in green | Shows that email id is available in green colour | pass |
| 3 | Check email |  | Existing email id | Displays email id not available in red | Shows that email id is not available in red colour | pass |
| 4 | Customer Login |  | Valid Email id, Password | Login success(redirect to home page) | Login success and displays home page | pass |
| 5 | Customer Login |  | Invalid Email id, Password | Login not successful and stays on the same page | Login not successful and remains on same page | Pass |
| 6 | Customer Gmail Login |  | Gmail id, Password | Redirects to login page shows continue to login | Return back to login page and shows continue option | pass |
| 7 | Show product |  | NO input parameters are required | Successful retrieval of menu items from the db. | Displays the existing menu items | pass |
| 8 | Add to cart |  | Pid, Price, Quantity | Successful addition of desired items to cart | Successfully added to cart | pass |
| 9 | Show cart |  | No parameters required | Displays the cart with the selected menu items | Shows the items present in cart | pass |
| 10 | Invalidate |  | NO input parameters are required | Redirects to home page after successful logout | Redirects to home page after successful logout | pass |
| 11 | Place order |  | Email id, pid, quantity, amount | Stores the parameters in the db after successful order placement | Redirects to order placed page | pass |
| 12 | Admin Login |  | Valid Email id, Password | Login success(redirect to home page) | Login success and displays home page | pass |
| 13 | Admin Login |  | Invalid Email id, Password | Login not successful and stays on the same page | Login not successful and remains on same page | pass |
| 14 | Add Product |  | Product name, category, price, weight, description, image url | Successful addition of product in the database | Successful insertion of product in database | pass |
| 15 | View product |  | No parameters required | Shows all the existing menu items | Displays available menu items | pass |
| 16 | Order history |  | None | Fetches all the orders from database | Shows all the orders | pass |
| 17 | View all registered customer |  | NO input parameters are required | Retrieves all the registered customers from database | Displays all the customers | pass |
| 18 | Delete product |  | NO input parameters are required | Removes product from the database | Successful deletion of menu item | pass |

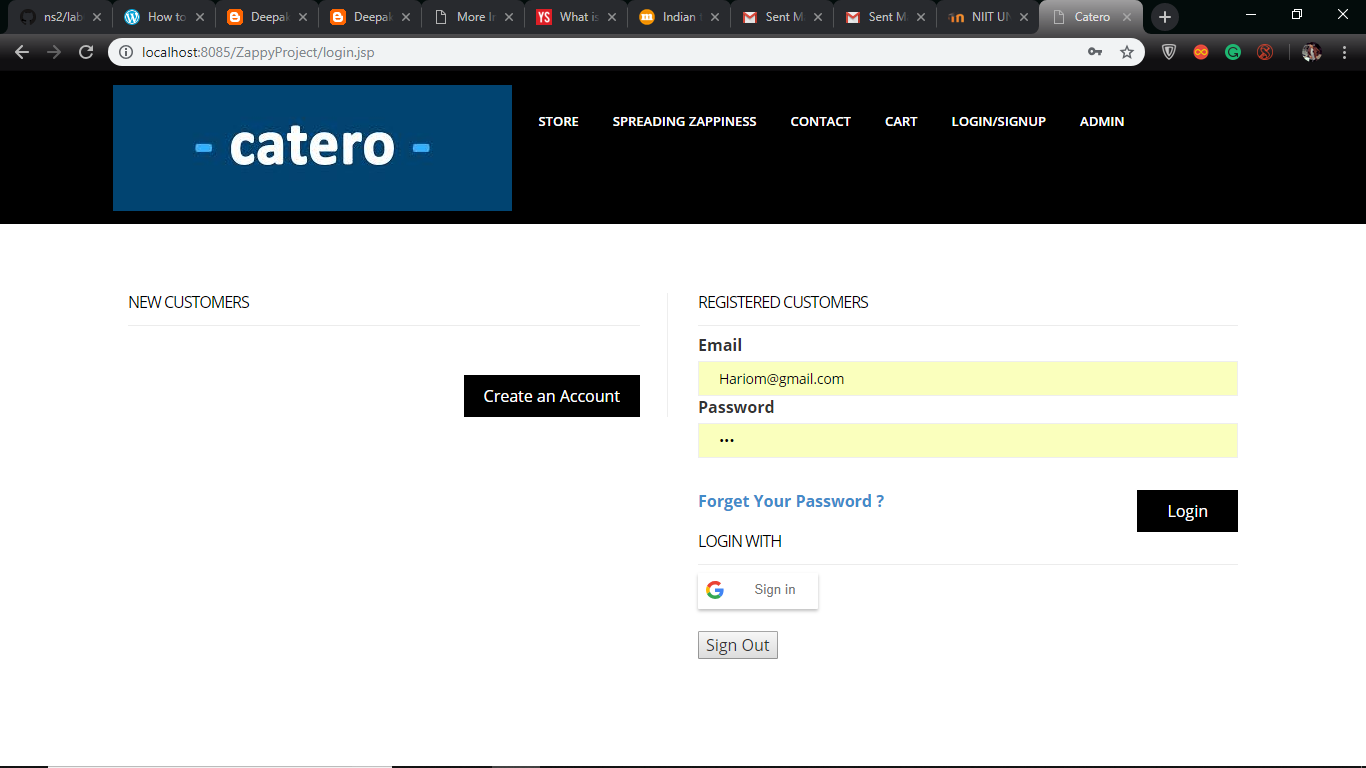
**Pass/Fail Criteria**: The system should satisfy all the functional requirements. Each feature to be tested will be evaluated against its requirement. The pass or fail of a test depends on whether the system meets with all the particular post conditions.

Manual testing screen shots

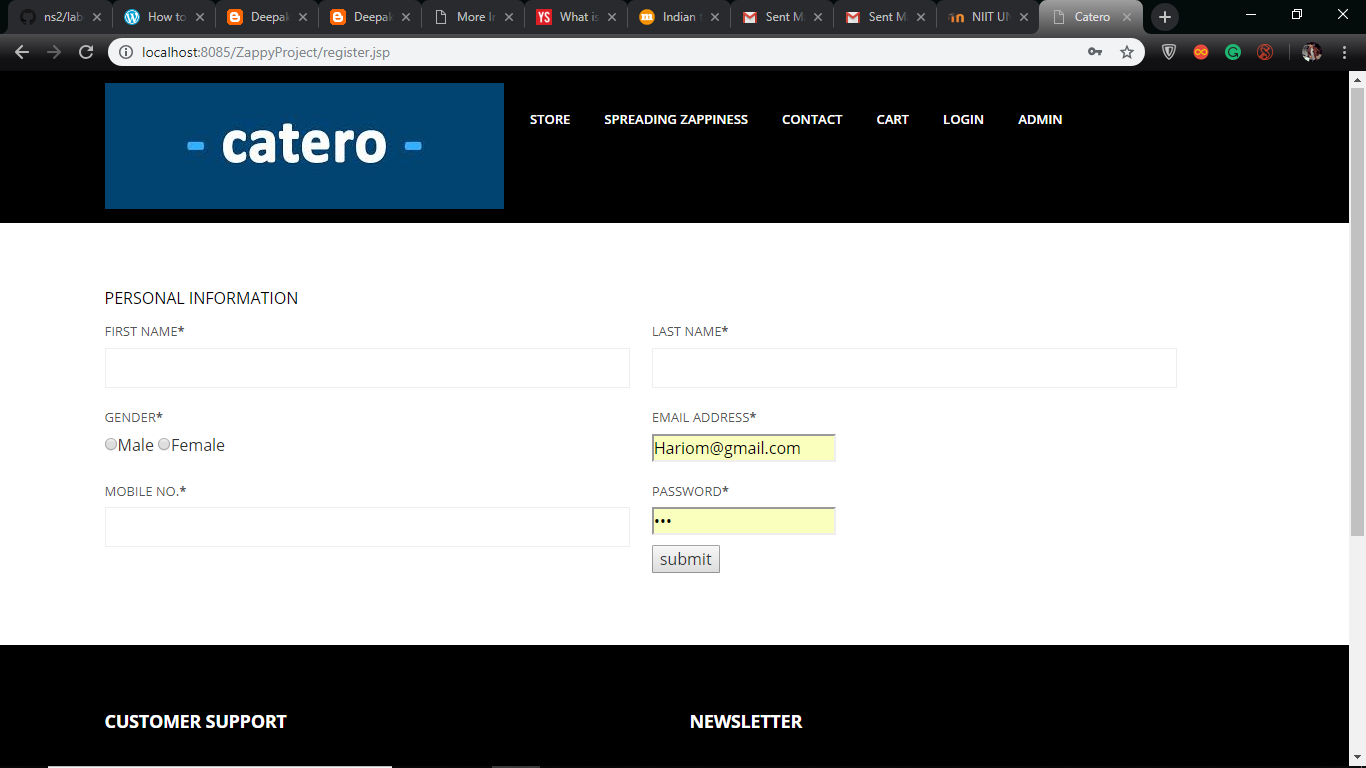
Test case for Admin login



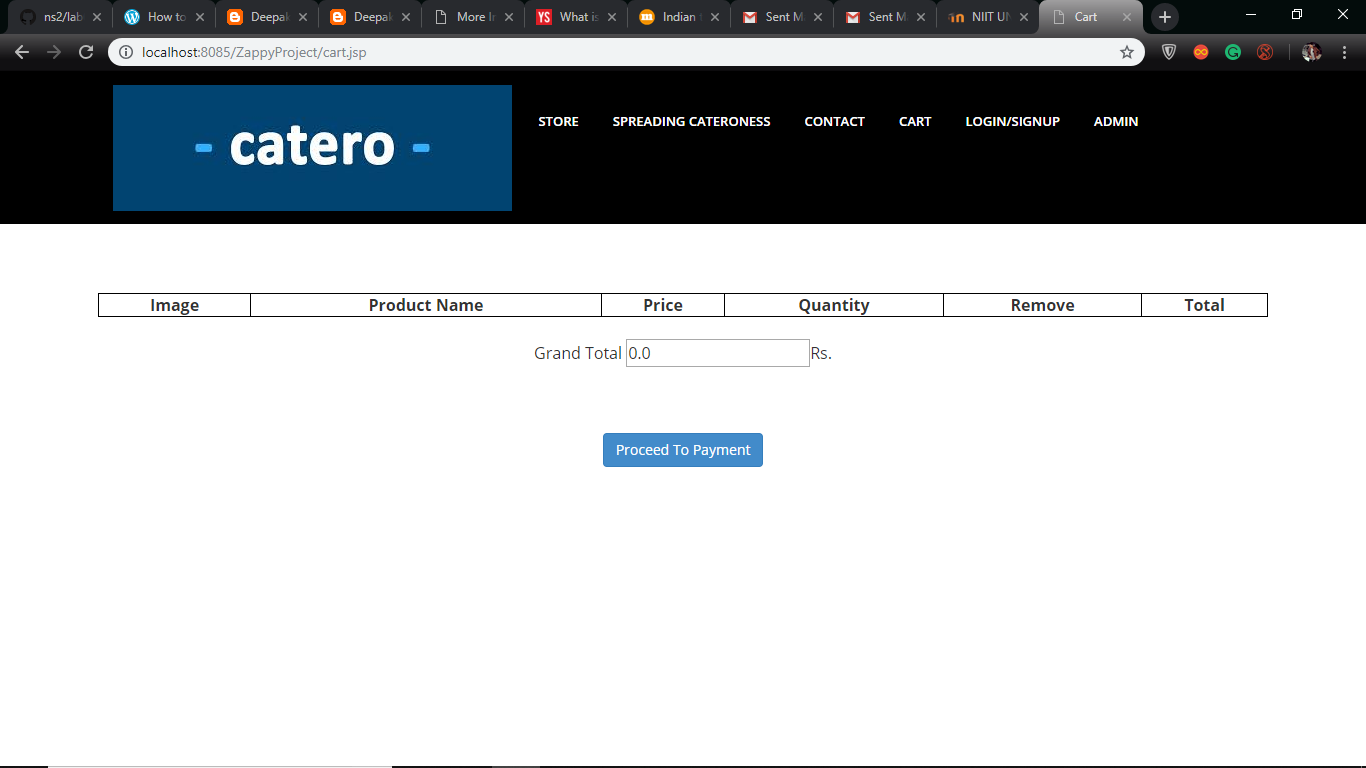
Test case for customer login

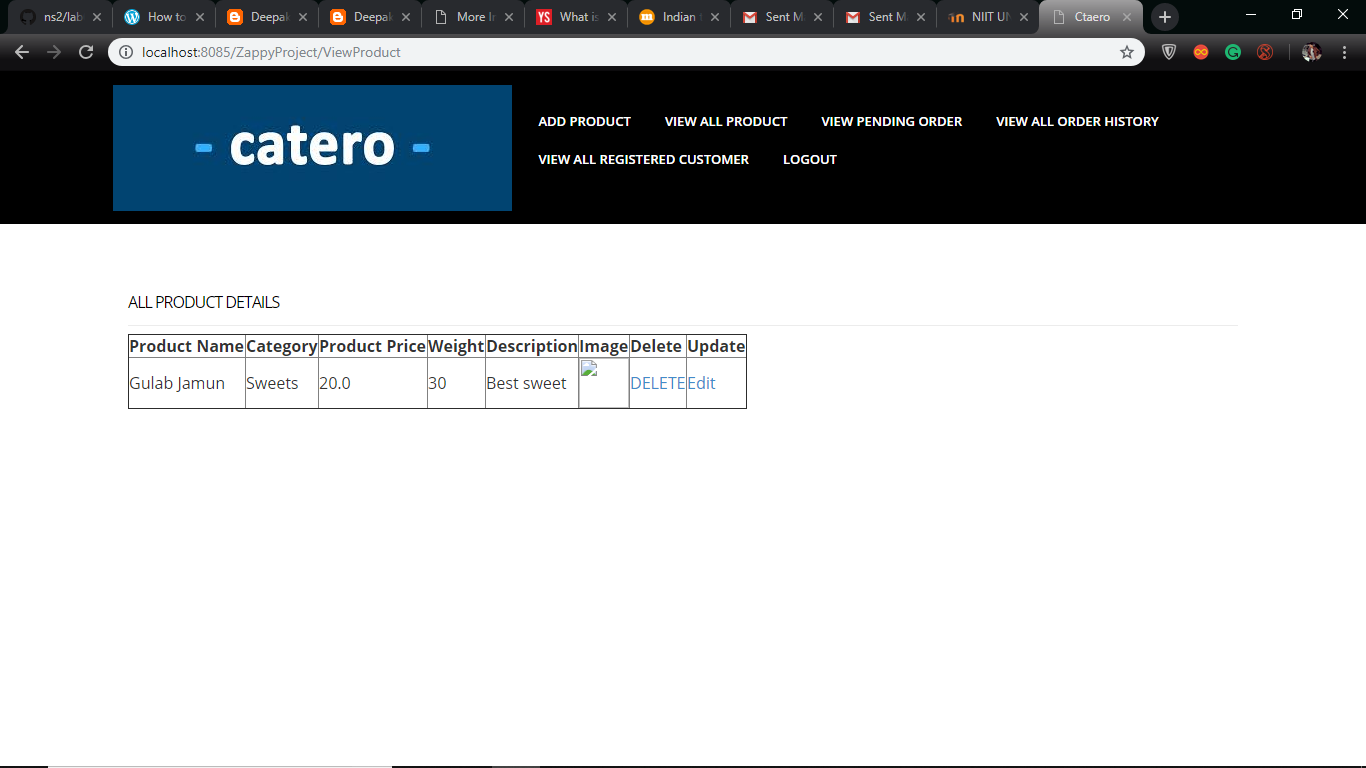


Test case for customer registration

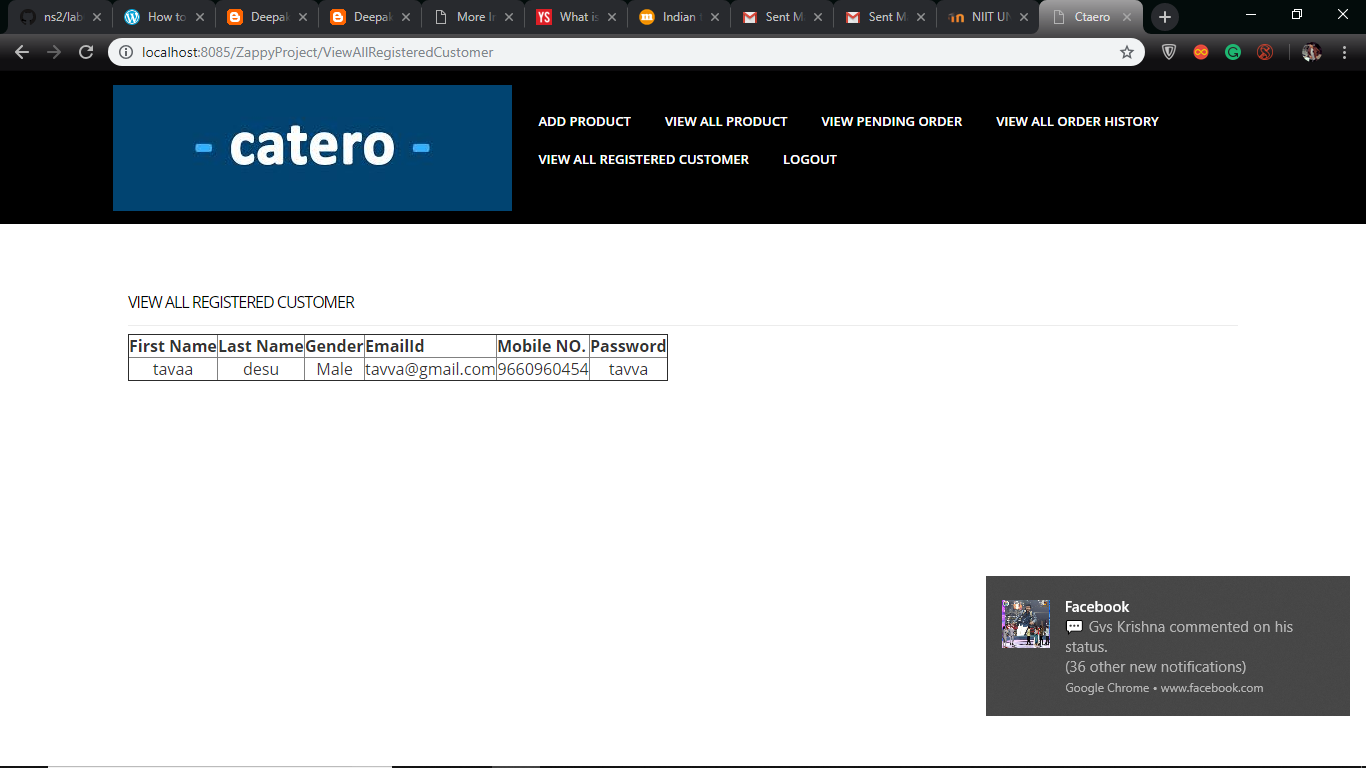


Test case for Add to cart



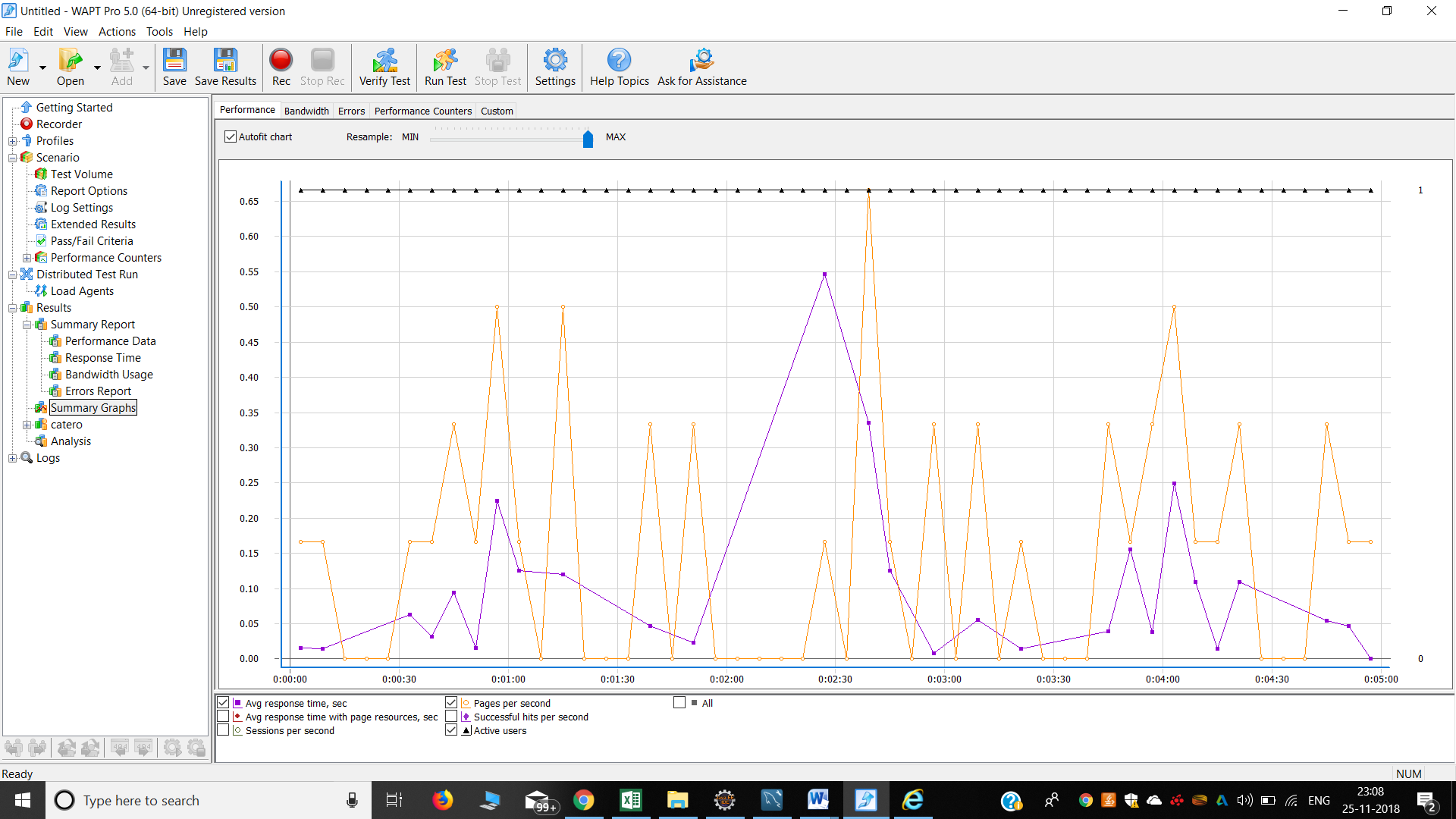
Test case for admin product details

Test case for registered customers

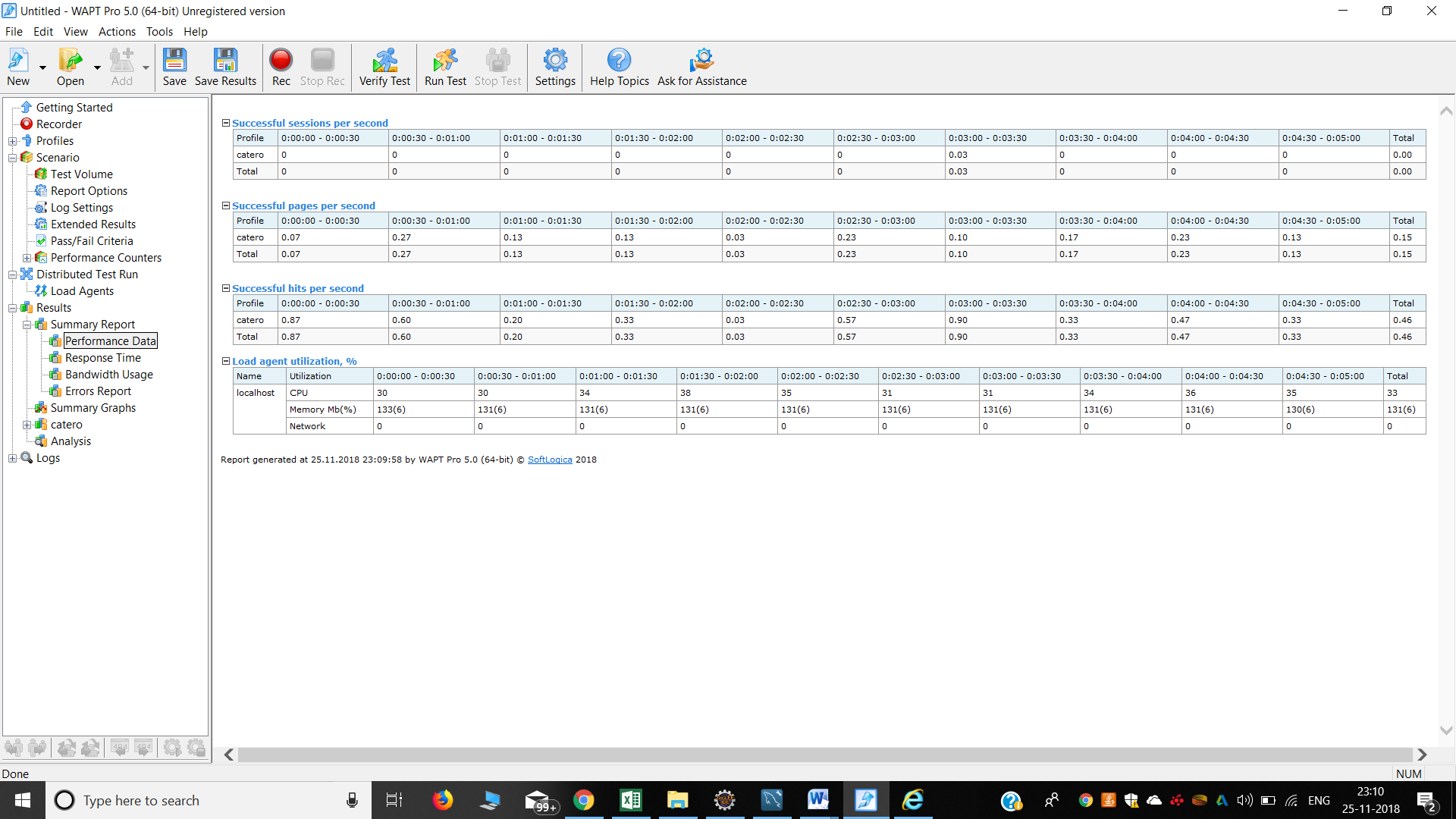


Performance testing of our whole website

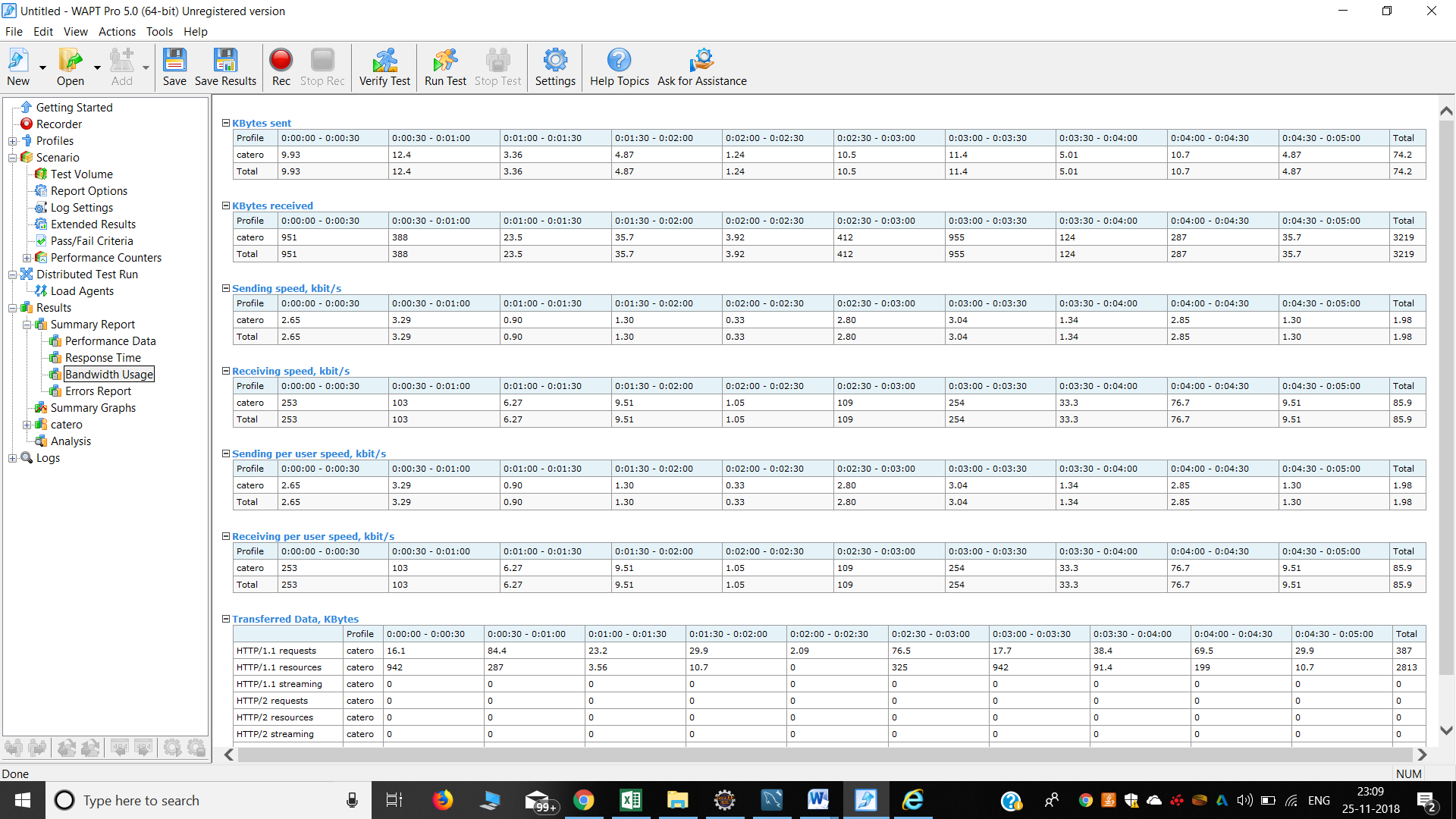
Summary graphs



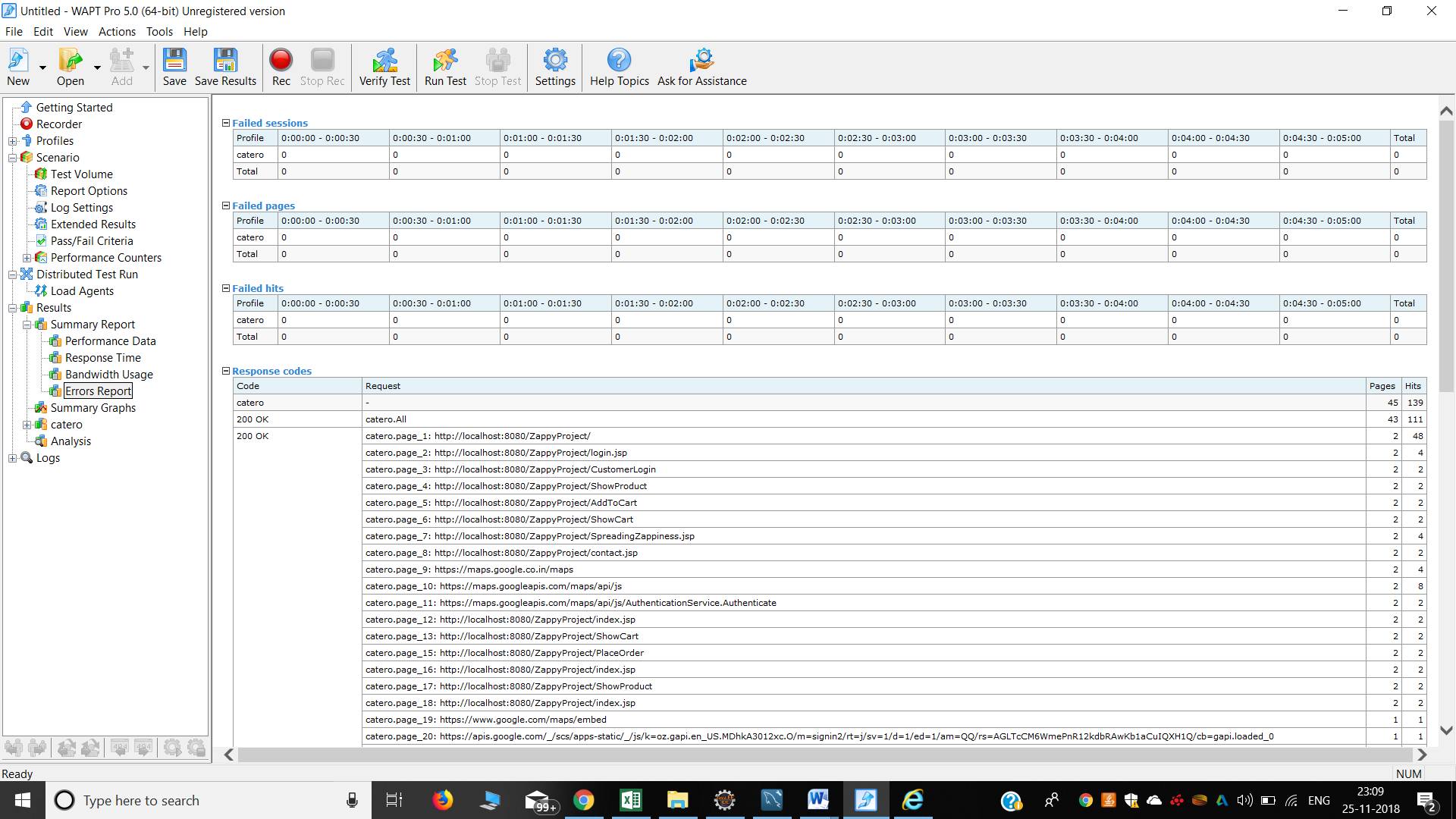
Performance data

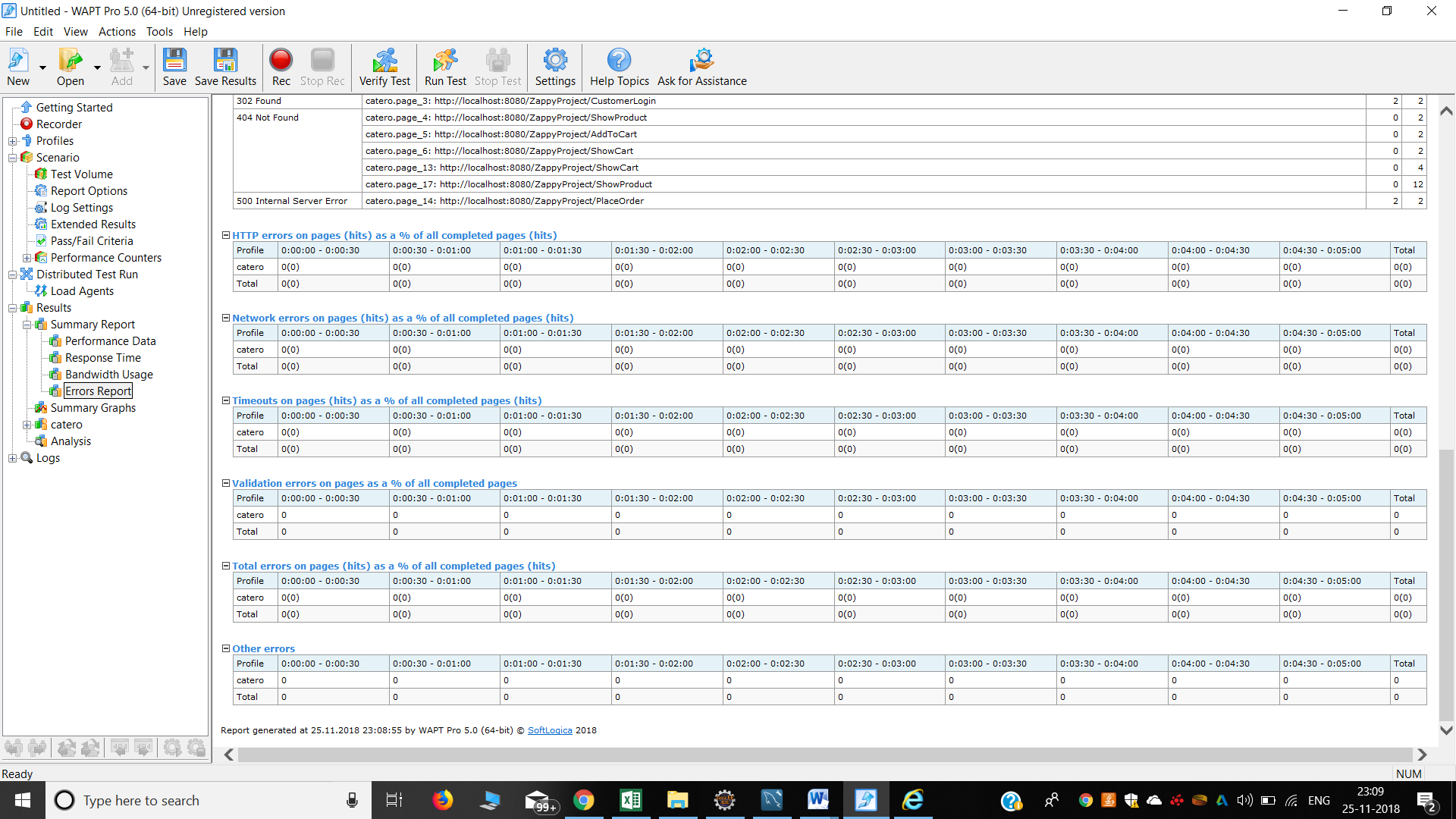
****

**Bandwidth usage**

****

**Error reports**

****

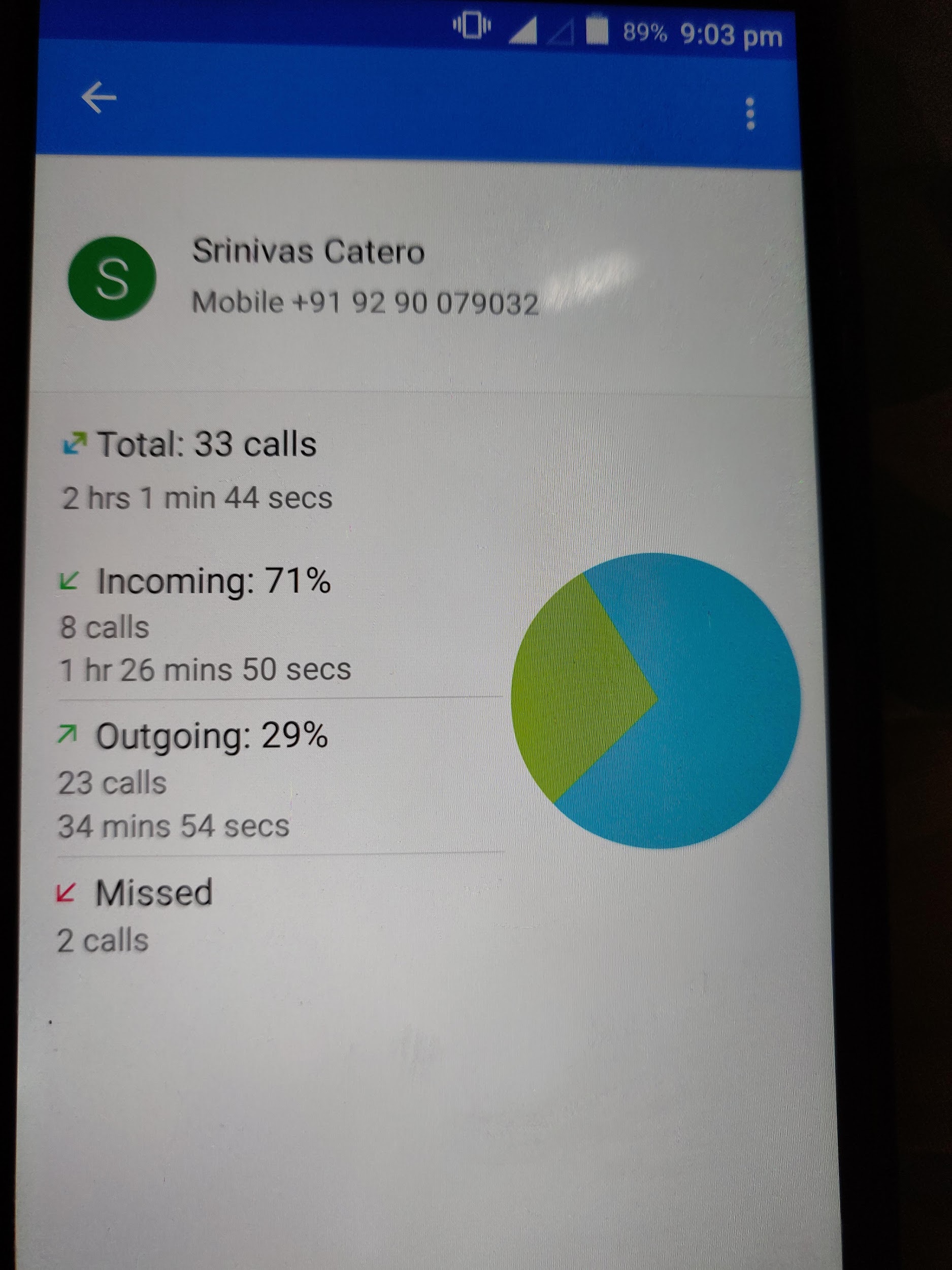
****

**Minutes of meeting with customer**

The Catero team worked on a commercial purpose to meet the need of our Customer who wants to start a catering website for their catering service which is being run successfully from many years.

The development team have always been actively interacting with our client since the very beginning of our project. We’ve been interacting with our customer through phone calls and emails and Skype calls. We also shared with him the details of every improvement at every point. We were also glad that our customer is excited with what we are doing and he is looking forward to get started with website of his own of his catering service. Here below we are sharing some of the screenshots and email.

**Phone Calls log with Client:**

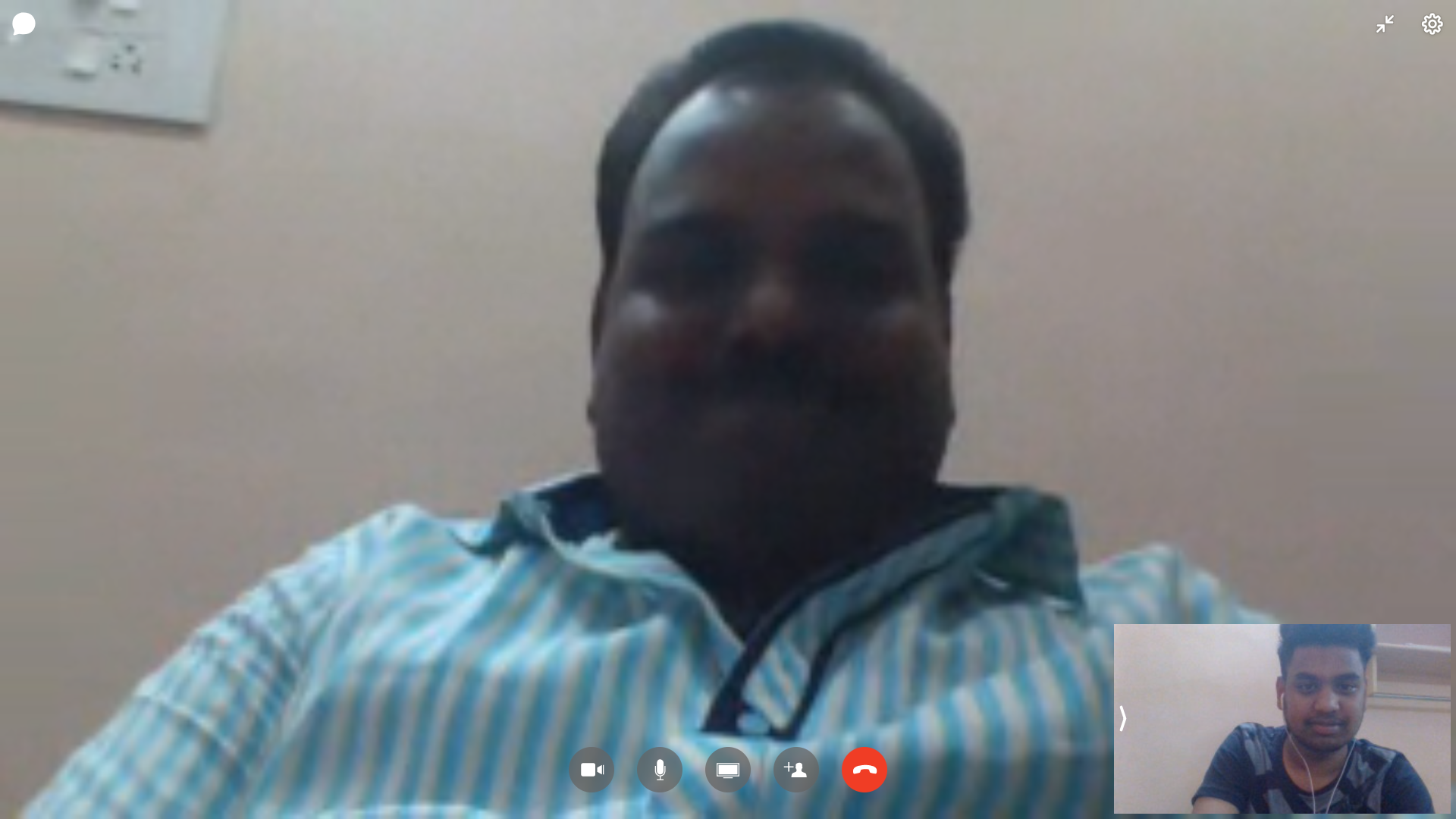


**Screenshot of Mail Confirmation by Client:**



**Video call screen shots with customer**





**Video link of customer about Catero**

<https://drive.google.com/file/d/1CPZ_Y1J-D67HJSCdSCgXCSObQhgqb-4A/view?usp=drivesdk>

**Nature Of The Customer**

There are always positive and negative feedback from our customer according to his need. As we are the development team and the project is just being developed by us and it is developed for our customer who intends to use this web application for his business growth. In this way we have only one customer for our project technically , with whom we regularly interact. As far as his nature is considered , he expected a decent work from us and which would turn out a good catering order platform who intends to order food from online. We generalized all the requirements of a customer ordering which we wanted to include as features of our web app before we started our work and let our customer know it , so that , we could add any features he’d say beforehand to the blueprint and make our work easier. Our customer was quite happy with the included features and waved a green flag.

At every step , we let the customer know all the developments we’ve been making and unlike many customers , he interacted with us technically and non-technically , giving us a push to develop our project and let us know his expectations over the user interface and the smoothness of the web application in every part clearly . His interaction helped us a lot in maintaining the standards as we were clear about what our customer wanted.

**Testimonials from Customers/Users**

It's a very useful website that I found in recent times, UI looks decent, it would be better if online payment option is available,overall it's fine - Lokesh Mamidi Btech 3rd year CSE

The web application looks good and is very useful for customers who wants to customise and order required menu which serves their purpose.- Bhanu Prakash Oleti Btech 3rd year CSE

This app is very useful for customers who wants catering service and the website is very easy to use by any kind of person. – Harihanth Julakanti Btech 4th year

**Feedback form**

* How easy was it to install our software?

Extremely easy Very easy Moderately easy Slightly easy Not at all easy

* How quick was the installation process for our software?

Extremely quick Very quick Moderately quick Slightly quick Not at all quick

* How user-friendly is our software's interface?

Extremely user-friendly Very user-friendly Moderately user-friendly Slightly user-friendly Not at all user-friendly

* How successful is our software in performing its intended task?

Extremely successful Very successful Moderately successful Slightly successful Not at all successful

* Overall, are you satisfied with the performance of our software and our team , neither satisfied nor dissatisfied with it, or dissatisfied with it?

Extremely satisfied Moderately satisfied Slightly satisfied Neither satisfied nor dissatisfied Slightly dissatisfied Moderately dissatisfied Extremely dissatisfied

* How likely are you to recommend our software and the team to others?

Extremely likely Very likely Moderately likely Slightly likely Not at all likely

* How approachable our team was

Extremely approachable Very approachable moderately approachable very difficult to meet don’t respond  on time

How do you rate the technical competence of the team

Extremely Talented Moderately TalentedNot talented at all

 b

Would you like to continue with our team in future

Would love to work Will at least give a thought Will not consider at all

a

The project was done as a course project of CS 301. Would you like to participate in selecting the team from pool of students next time for another product?

Would love to work Will at least give a thought Will not consider at all

a

How much business time is saved by the tool/product developed by our team compared to your traditional way of doing the business(e.g. without this product/tool/software)

 0-10 % 10-30% 30-60% 60-80% more than 80%

c

Do you think the product is helpful to increase the business turnover

Quite sure It is likely to do so Don’t know/can’t say It was waste of time. No improvement at all.

a

How much percentage of your expectations /requirements are satisfied

100% 70-90 % 40-60% below 40%

b

If you were the course supervisor how would you evaluate the project?

A+A B B+ C+ C D+ D Below D 

b

How would you rate the sincerity of the team

Any other suggestion/feedback/comment

extremely sincere Good level of sincerity is there Moderate level of sincerity Not sincere at all

**Tools and Technologies used during the Catero development**

Tools for Documentation like SRS

* Microsoft Word document
* Snipping Tool

Tools for Design document

* Argouml
* Lucid Chart

Technologies

* HTML
* CSS
* Java Script
* JAVA
* Servlets
* JSP
* Spring

Tools used for software specification

* GITHUB
* Eclipse
* My SQL

Tools for communication

* Gmail
* Whatsapp
* Messenger
* Phone calls

Tools for metrics

* Cyvis
* Source Monitor

**Novelty Of The Project**

The following article provides the novelty of our project idea and the solution developed as part of our project work in the course Software engineering. Our project is developing a dynamic website for a Catering management system that is accessible to users from both the computer over a Stable web connection. Having a website is an absolute necessity for most businesses today. This is particularly true for restaurants and catering services due to an emerging trend in the industry that is helping to increase sales: online ordering. Customers will often visit a website to learn all about it, including the menu.

Statistics Are Overwhelming: Digital Orders Are Rising. To stay current, we developed a website for our customer in order to increase its sales as the catering service is already so popular so this might add up to their business and help increase in their sales and profits. This will be helpful for most the users and catering service.

We have kept fixed menus as well as there can be a customisable menu which we think users will like pretty much since they can order different items which can satisfy to their standard and requirement.

**Sophistication Value Of The Project**

1. **Google SignIn**: This was one complex task where we had to burn some midnight oil for setting up Google SignIn using google console api by generating Oauth2 authentication.
2. **Geographic location:** The task of setting up location of our clients office was tedious in the first place but with constant efforts and motto of not giving up we made that happen.
3. **Forgot Password**: We’ve at first thought of sending a password reset to the user if he forgets his password ,through a mail. But after dwelling on it and trying to apply , there were some kind of constant errors appearing after many corrections which we were unable to solve it.
4. **Image Retrieving And Uploading To Database:** Though retrieving normal data was not a hard task , the retrieval and uploading of images turned out not to be so easy .
5. **Null Point Exception:** This kind of exception occurred very often in every part of code where a string is used . But , we searched about it and after spending sometime over it on some technical websites like stack overflow, we were able to solve it.

**6) Coding Using JSP And Servlet:** After all , coding in jsp and servlets didn’t turn out to be an easy task compared to languages like php and others employed by other groups .Though it was hard , we chose it cause we believed it would give the perfect result for our expectations . We dealt with some errors at the beginning and faced trouble with understanding the working .But at last , we are successful.

**Applicability of the Project**

The purpose of this phase is to present a detail description about the applicability of our project Catero. The purpose of the Catering management service is to create a convenient and easy-to-use application for customers, trying to book the orders. Catering services have become a very popular thing for most of the people these days as catering services are being utilised by everyone these days. By keeping this thing in mind we developed a website for our client who has catering service which might help him in growing his business and the user can do the following things.

Users can book catering service from their place itself  and the catering service serves the purpose of those who have functions or parties or marriages and helps them in catering.

Users can explore the different menu’s available in the website and chose amongst them or can customise their own menu according to their requirement and then the Catering service comes in contact with the customer and helps them in managing their order.