**Architecture and Detailed Design Specification Template**

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Approver** | **Comments** |
| <date> | <Version 1> | <Your Name> | <Approver’s Name> | <First Revision> |
| 26/10/17 | Completion of all sections of the document | All team members |  |  |
| 29/10/17 | Changes made based on review by team C3. Section 2.2(Updation of analytics in detail) | N L Ramya |  |  |
| 29/10/17 | Changes made based on review by team C3 ( Reviewing test cases) | N L Ramya |  |  |

# Document Approval

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  | <Your Name> | Lead Software Eng. |  |
|  |  |  |  |
|  |  |  |  |

Contents

[**Revision History**](#_30j0zll) **1**

[**Document Approval**](#_1fob9te) **1**

[**1.0 Architectural and Component-level design**](#_2et92p0) **3**

[1.1 Management System](#_3zqktxls45z9) 3

[1.2 Cloud Computing System](#_vvqq2iqtlbmz) 4

[1.3 Real Time Monitoring System](#_ac8jqf9drj3s) 4

[**2.0 System Structure**](#_tyjcwt) **5**

[2.1 Architecture diagram](#_9f84lm80gkl4) 5

[2.2 Description for Component](#_1t3h5sf) 5

[2.3 Interaction Diagrams](#_a5oay79m0vu0) 6

[2.4 Describe usage scenarios and how you would test that](#_8aoth5onpb9e) 6

[2.5 Architectural Styles and Patterns considered and for what reason](#_17dp8vu) 6

[**3.0 User interface design**](#_7s9cyd7pour3) **7**

[3.1 ManageGuru’s Management-End Login Page](#_bl66646u0vx) 7

[3.2 ManageGuru’s Billing Entry Page](#_rocpp0v6idrk) 8

[3.3 ManageGuru’s Waiter Order Entry](#_63xvbzyjad8q) 8

[3.4 ManageGuru’s Chef Management-End](#_hx4djkq5gwz9) 9

[3.5 Other User Interfaces](#_gcu72ial01p3) 9

[**4.0 Detailed Design Approach**](#_26in1rg) **10**

[4.1 Design patterns considered and for what reason](#_lnxbz9) 10

# 1.0 Architectural and Component-level design

Our product consists of three main components:

1. Web-based Management System
2. Centralised Cloud Computing System
3. Real Time Monitoring System (a Mobile Application and a Web Application)

## **1.1 Management System**management.jpg

This forms the data aggregation end of our system. The system is designed to provide an easy hotel management. The system is divided into more simpler components, 1) a simple web interface for the waiters to take in orders based on the current menu prepared by the head chef, 2) web interface for the billing staff to get all the orders placed by a particular table, 3) web interface for Inventory management and menu management for the head chef and 4) web interface for a local manager to administrate the whole system.

All the above components are linked by a local database set up at the hotel.

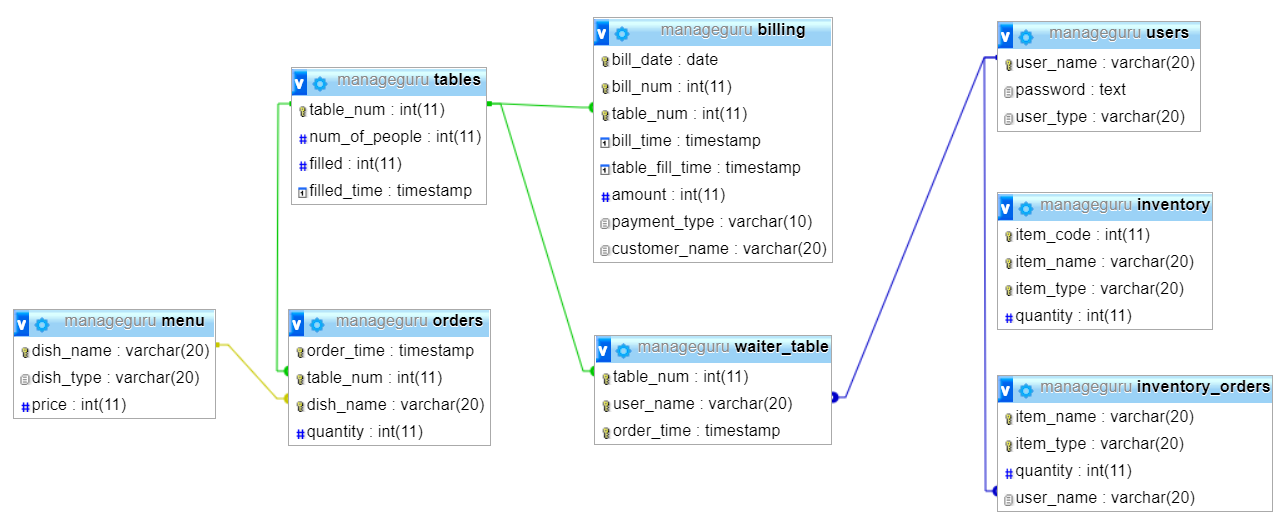


Fig 1.1 Local Database Schema

## **1.2 Cloud Computing System**

ManageGuru’s Cloud system form the central core system of our product. The updates from the management system is stored our cloud where analytics is performed on the gathered data. The cloud also runs a MySQL database server. Based on the analytics reports on the functioning of the franchises management system is provided to the owner/manager who might not be present at the location.

The system as said is a centralised system, so the owner/manager uses a mobile application to get real time updates by connecting to our cloud (ReSTFull API’s).

## **1.3 Real Time Monitoring System**

We provide the user with a mobile application that renders different real time stats management system, any time, any place. The system has a user authentication page at start and then the application renders the updates by invoking ReSTFull API’s designed at the cloud end.

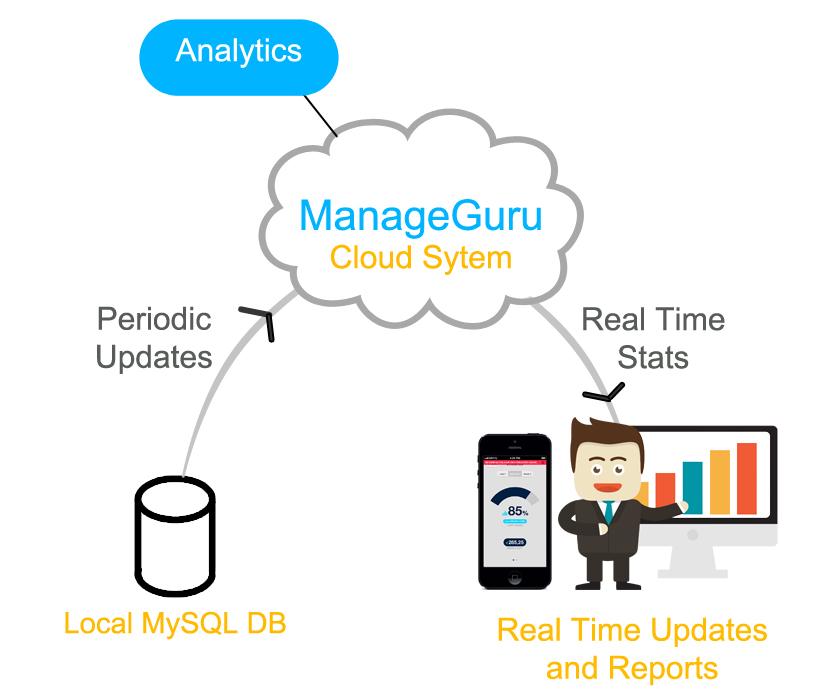


Fig 1.2 The Whole System

# 2.0 System Structure

## **2.1 Architecture diagram**

**Architecture style:** Client - Server

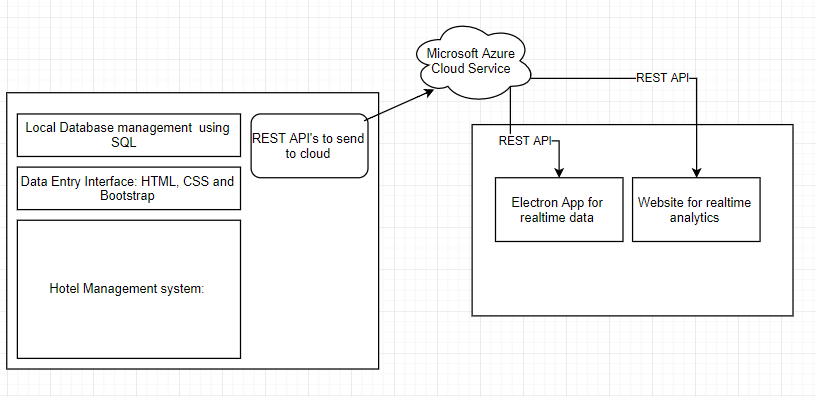


Fig 2.1 The System Architecture. Centralised Azure cloud system providing RestFull service to the both the management end and also the monitoring end.

## **2.2 Description for Component**

As discussed above the system consists of three main components, 1) Web-based Management System, 2) Centralised Cloud Computing System and 3) Real Time Monitoring System.

The ***Management System*** has a local webserver with MySQL database. It consists of multiple UI’s built using **HTML5, Bootstrap and JQuery**. The backend is developed in **PHP**.

The ***Cloud System*** being used is **Microsoft Azure**. The cloud will also contain a **MySQL database** to store the updates. Analytics will be performed on the gathered data to generate reports. **ReSTFull API**’s are built for the users to get, post and update data at the cloud. The analytics we plan to include are such as the most active waiter and the most trending dish. We also plan to have bar graphs and pie charts showing the number of orders per day and per month and also the top most ordered dishes. We are also including the number of tables filled currently and average income to the outlet of the restaurant.

The ***Monitoring System*** is a mobile/web application. The system is built so that the owner/manager can check the working of the hotel even when not present locally. It is built using **Electron framework**. A web page which shows the reports generated from the analytics.

## **2.3 Interaction Diagrams**

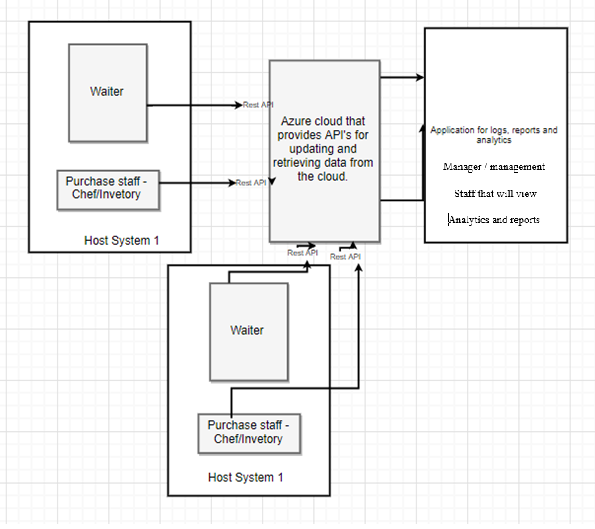


Fig 2.2 Interaction Diagram. The diagram shows the interaction between the owner/manager, the hotel staff members, and the cloud system.

## **2.4 Describe usage scenarios and how you would test that**

* The outlet staff will be provided with functionalities such as: Interface to easily view billing data, update new orders and transactions to the system.
* The purchase staff will have access to: Inventory manager which will keep track of all the goods and their respective quantities. Furthermore, they will be able to places orders to different vendors
* The management staff can do the following: View analytics of previous orders - make predictions as to what the customer currently requires.

Scenario testing can help to explore how the software will work in the hands of an end user.

**Test Scenario 1:** Check the Login Functionality

**Test Scenario 2:** Check that a New Order can be created

**Test Scenario** 3. Check without internet connectivity.

**Test Scenario 4:** Check reports and graphs

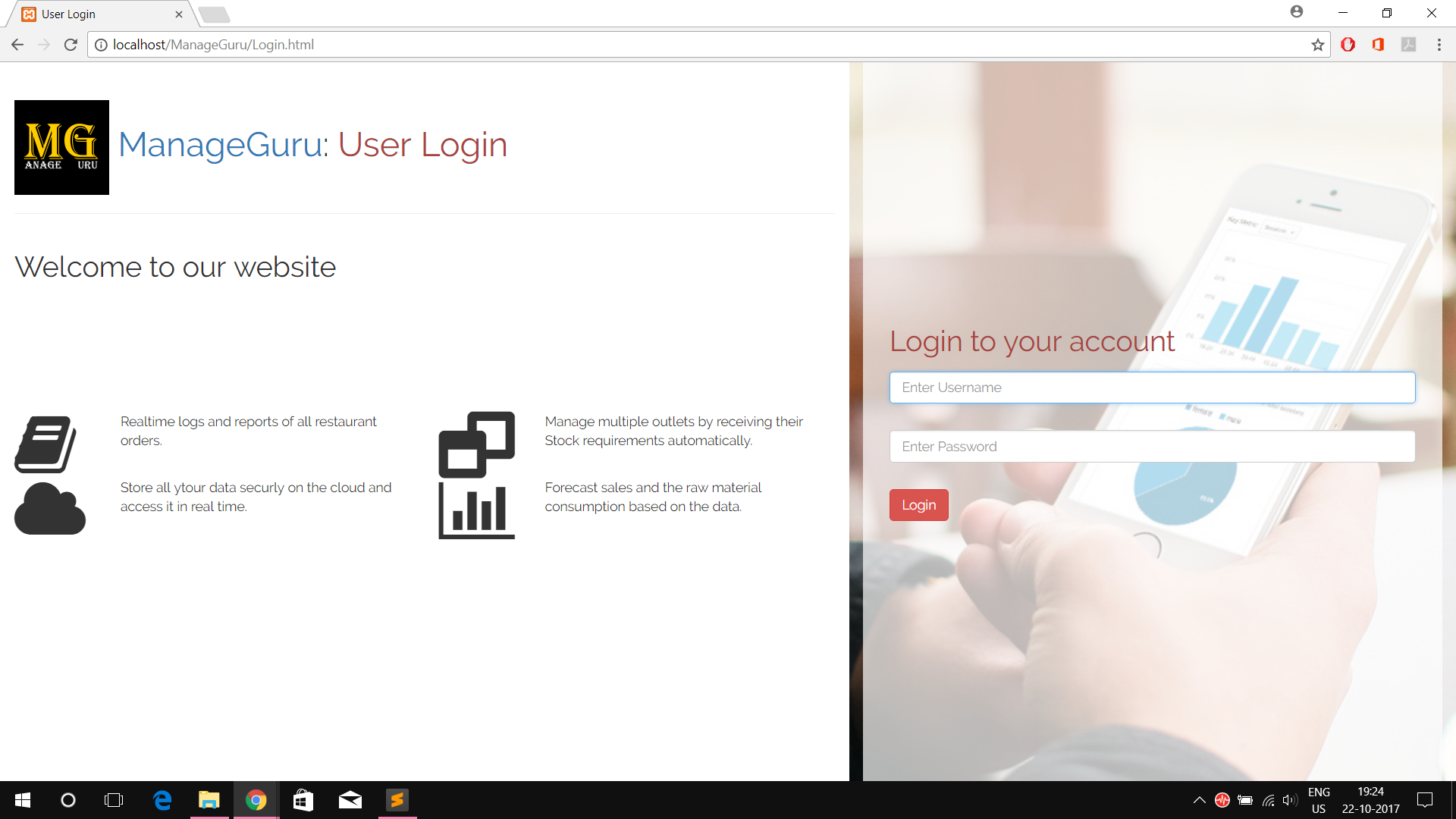
## **2.5 Architectural Styles and Patterns considered and for what reason**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N** | **Architectural Styles/ Pattern** | **Intent of this pattern** | **Rationale for choosing or not choosing** |
| 1 | Component Based SE | Each component is independent of every other | We have 3 major parts of the project:   1. Management system 2. Cloud services 3. Real Time application and Data analytics application.   Developing each of them separately simplifies our approach |
| 2. | MVC | Model: Data  View: UI  Controller: Exchange of information between the two | We could separate our project into these 3 parts. Since each of them would be independent, changing the view or the model would not affect the other. |
| 3. | State logic display | This keeps our view different from the logic and data | This type of system allows for any part of the system to be modified without  having to change the other two parts. |
| 4. | Client – Server | All the data is stored on the server and client accesses it. | We have multiple websites and applications accessing a single server |
| 5. | RESTful Services | Architecture pattern for the web that enforces uniform interface and other constraints bringing performance, scalability and modifiability. | We have multiple services and data transferred across various parts of our system which would benefit. |

# 3.0 User interface design

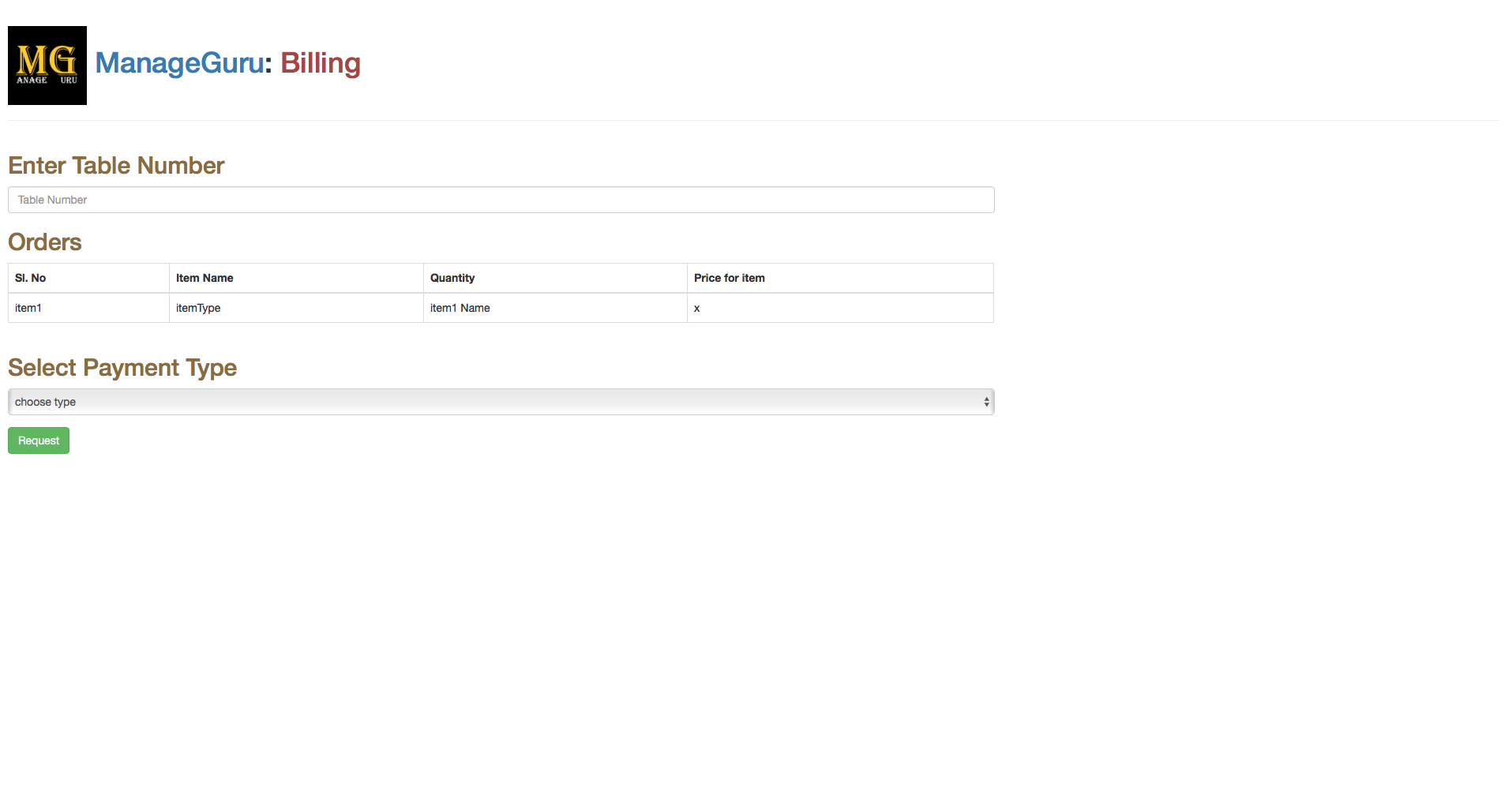
## **3.1 ManageGuru’s Management-End Login Page**

The page consists of information about the services provided by our product **ManageGuru**. A side panel to enter the login credentials. The design is screen size responsive.



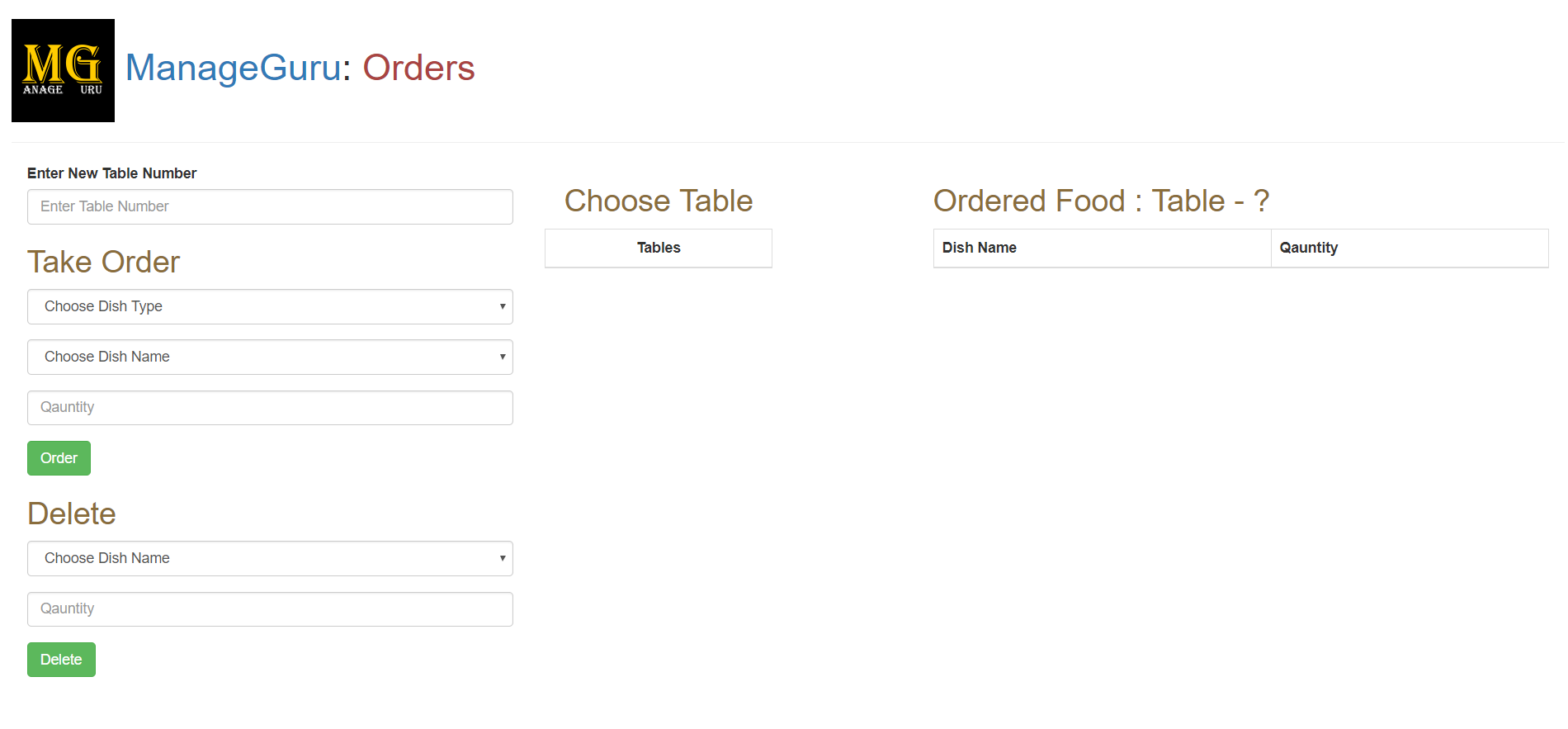
## **3.2 ManageGuru’s Billing Entry Page**

The billing page is used to take account of amount paid and the mode of payment for the respective bill number. The bill also has the orders placed by the customer. The UI here provides a view of **all the orders placed** by the customers at a table and the **total amount** and a **select option** to choose the mode of payment used by the customer (ex. cash or card).



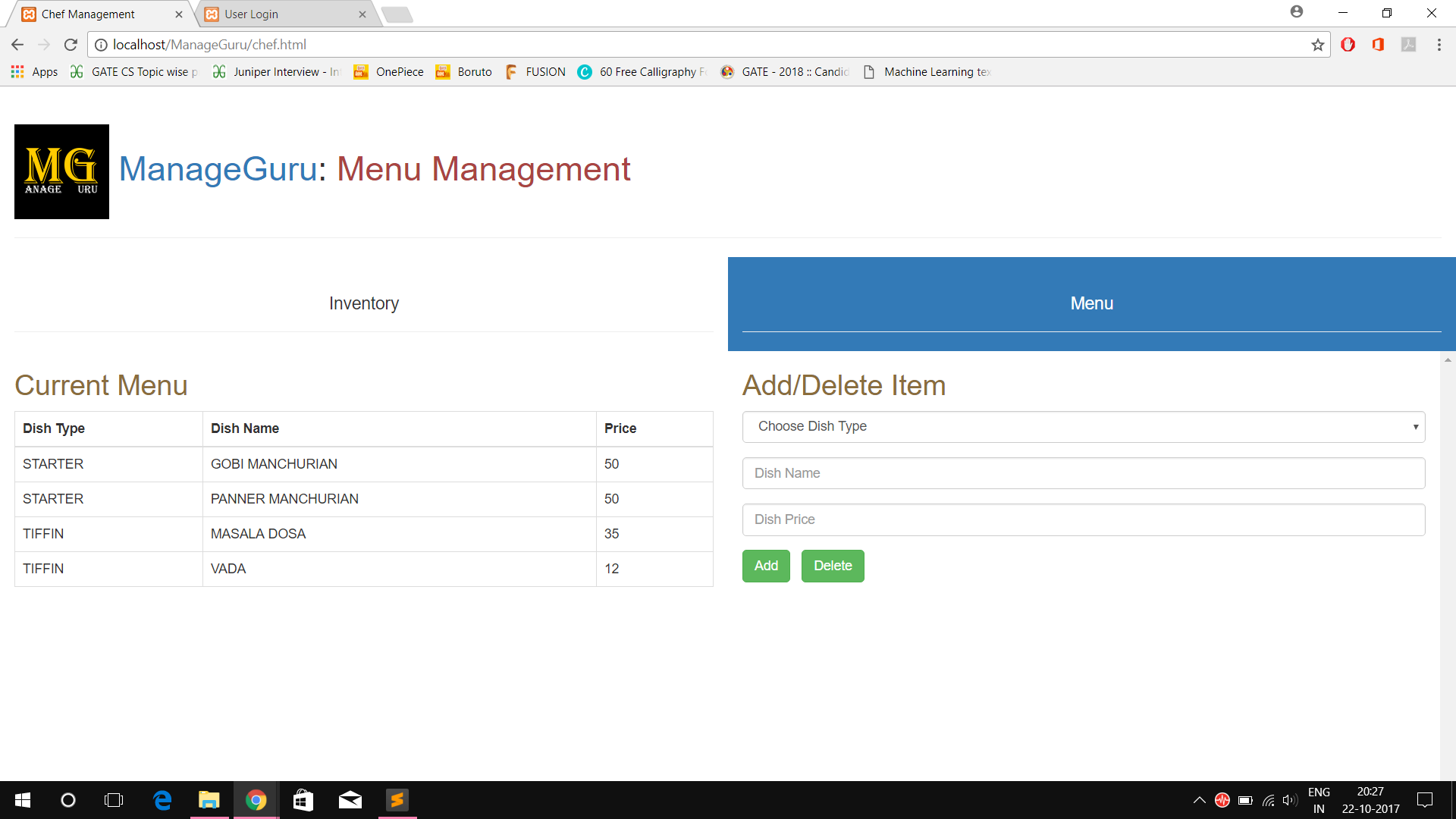
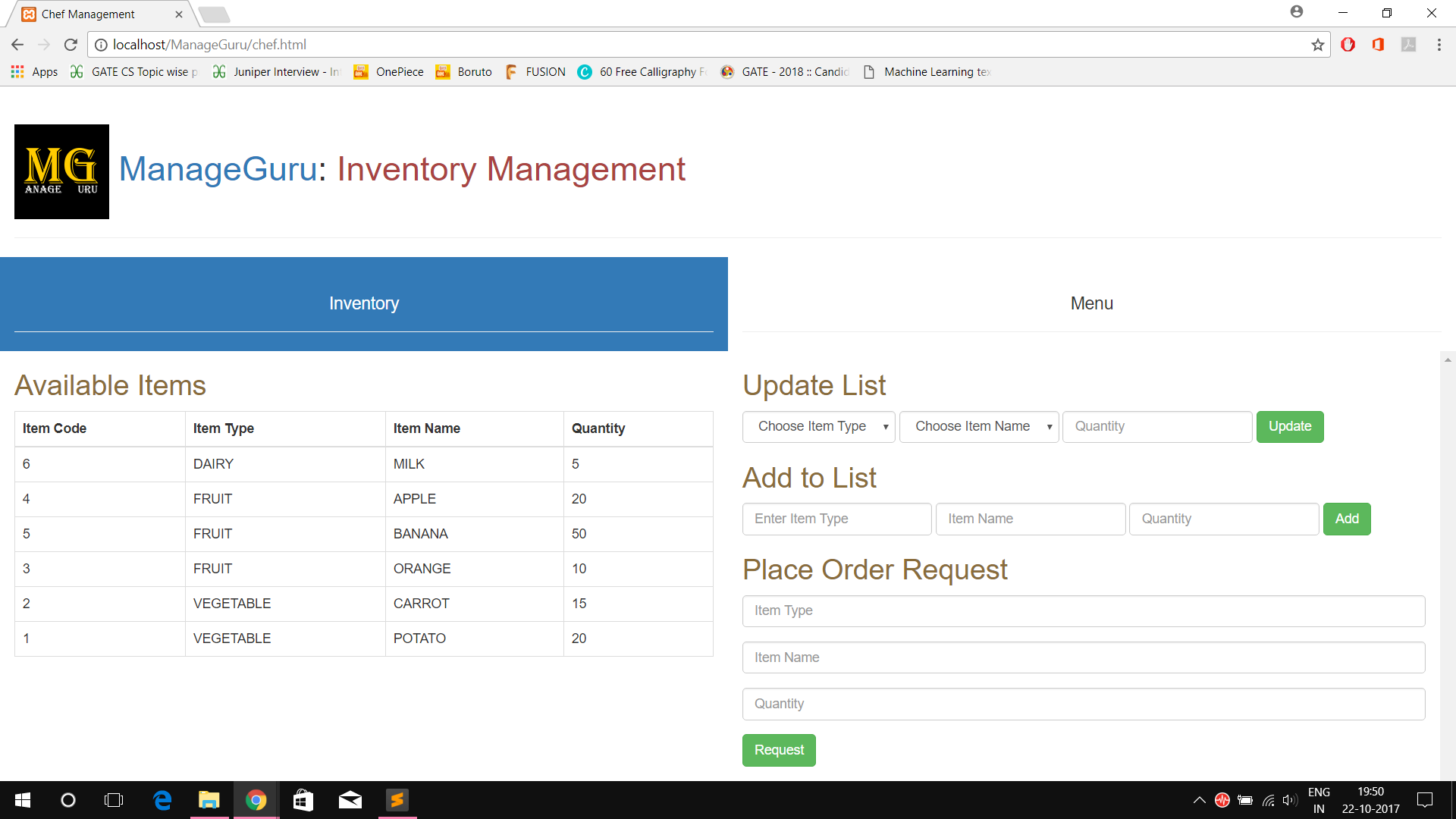
## **3.3 ManageGuru’s Waiter Order Entry**

The orders page is used by the waiters of the hotel. It provides UI to **take orders** for a **particular table** and also **shows the all ordered food** by the customer at the table. If any order is cancelled by the customer, then a **Delete** form is provided to remove the dish from the orders.



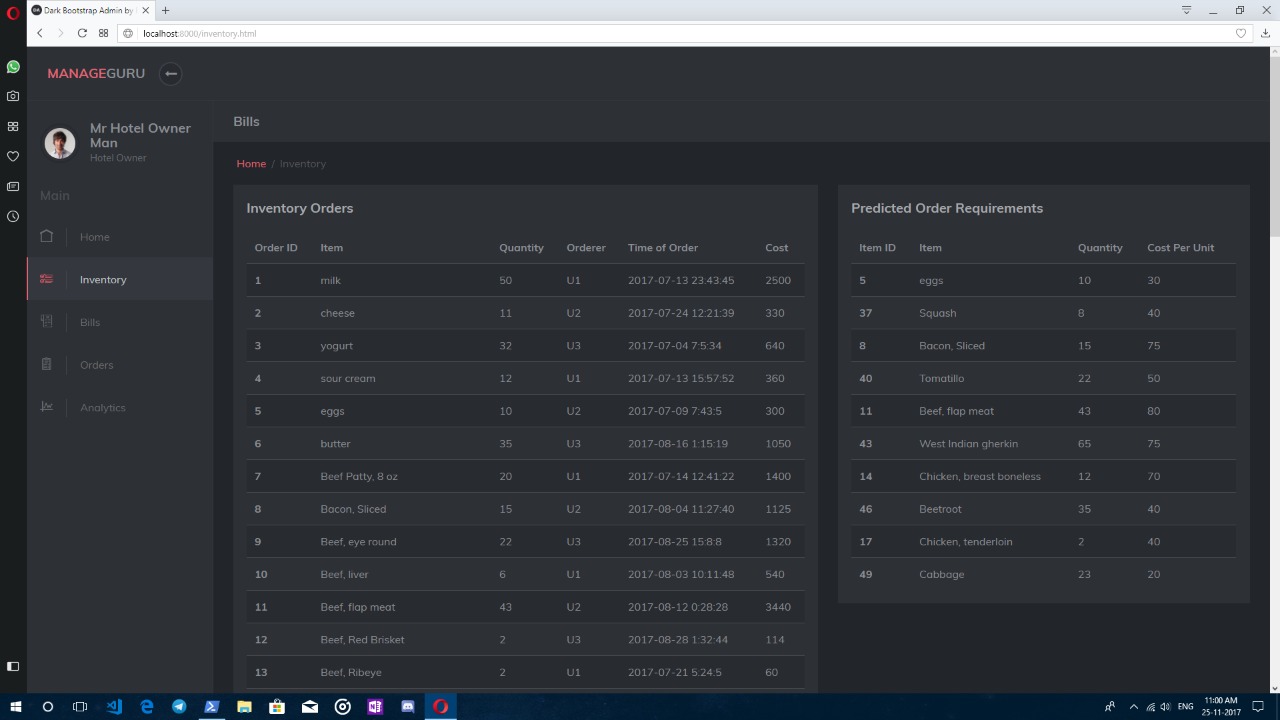
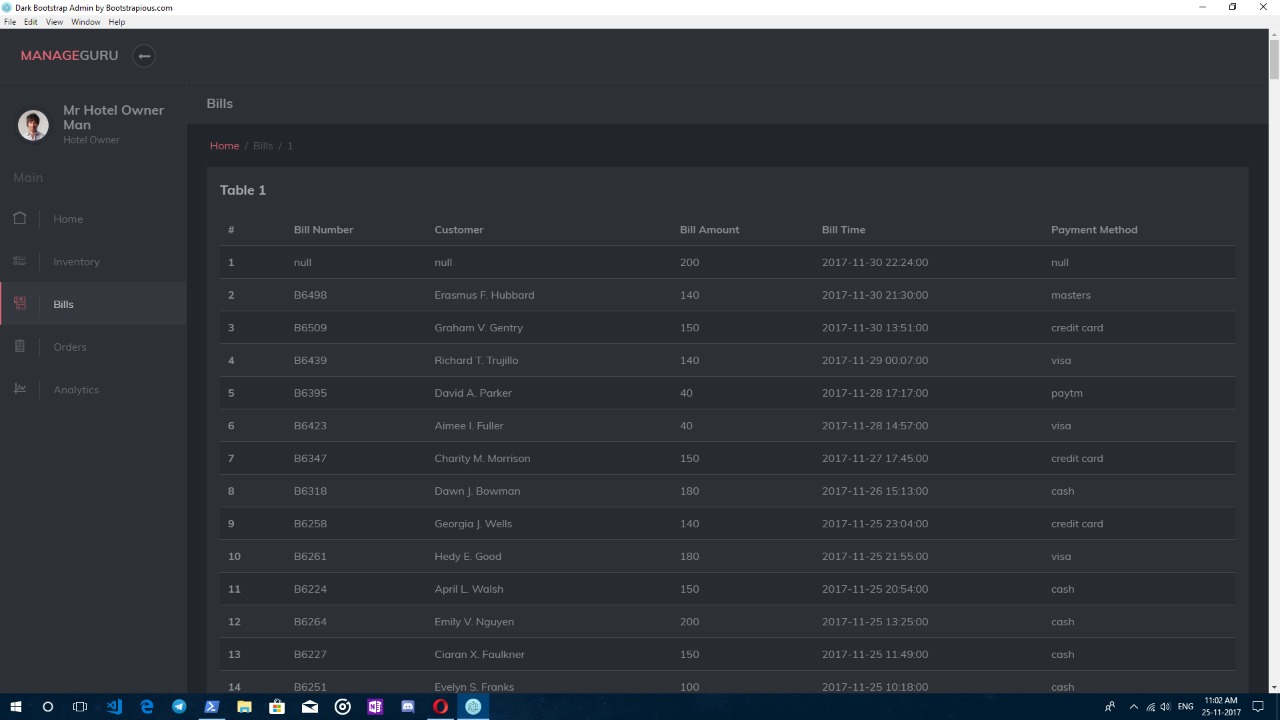
## **3.4 ManageGuru’s Chef Management-End**

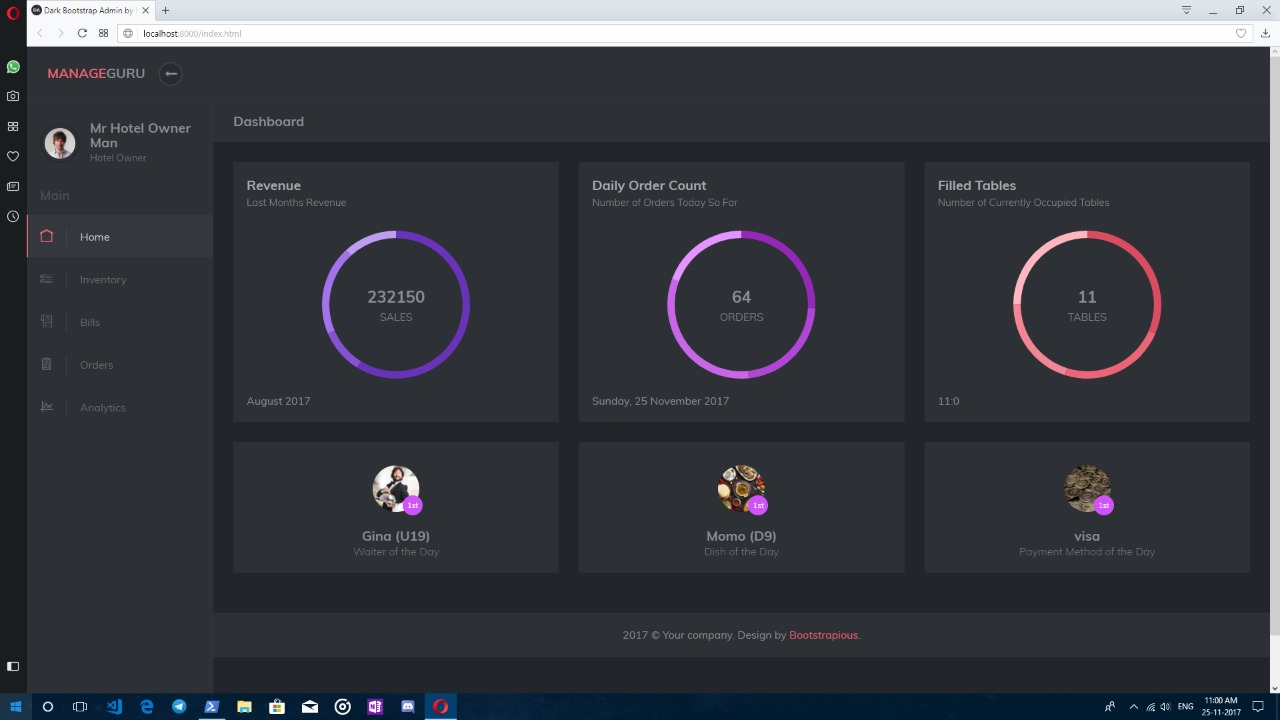
The Chef Management end has two parts, 1) Inventory Management and 2) Menu Management. In Inventory Management tab the chef can **see all the current items** present in the inventory, **update the list** if the items are used, **add new items** to the inventory list and also **place a buy order request** which is approved by the owner/manager. In the Menu Management tab, the chef can **add/delete the dishes** in the menu.

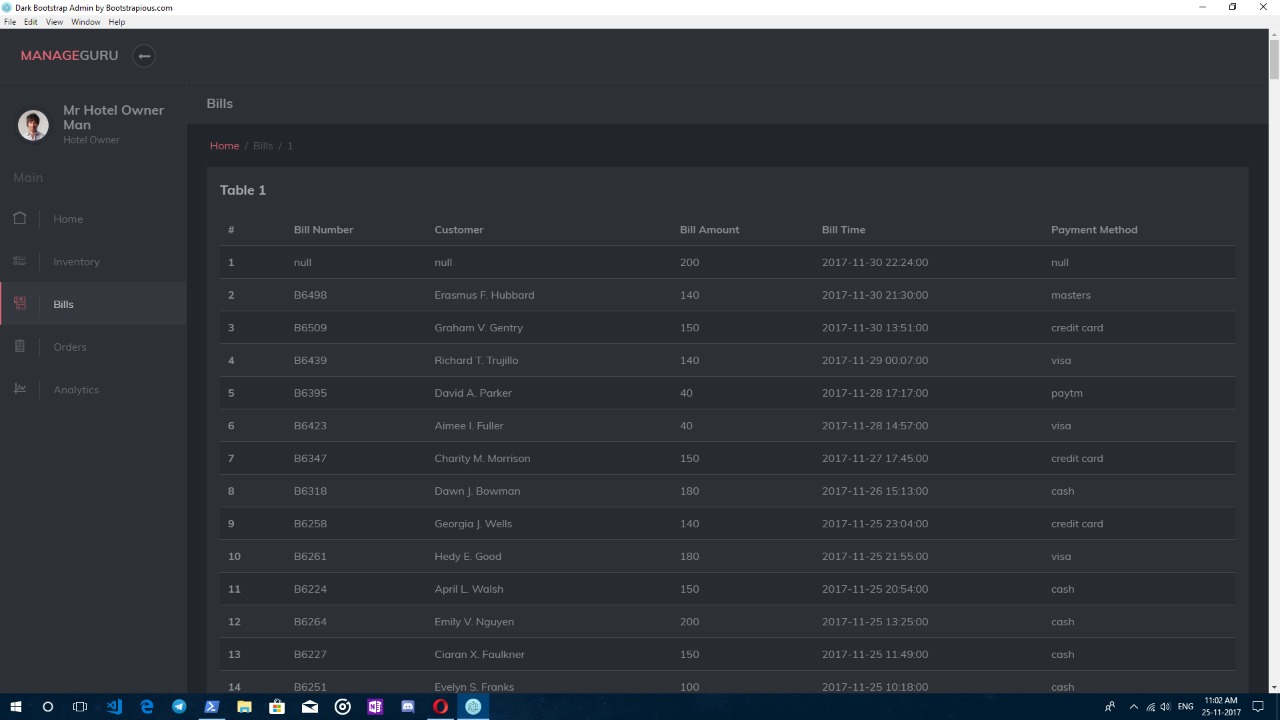
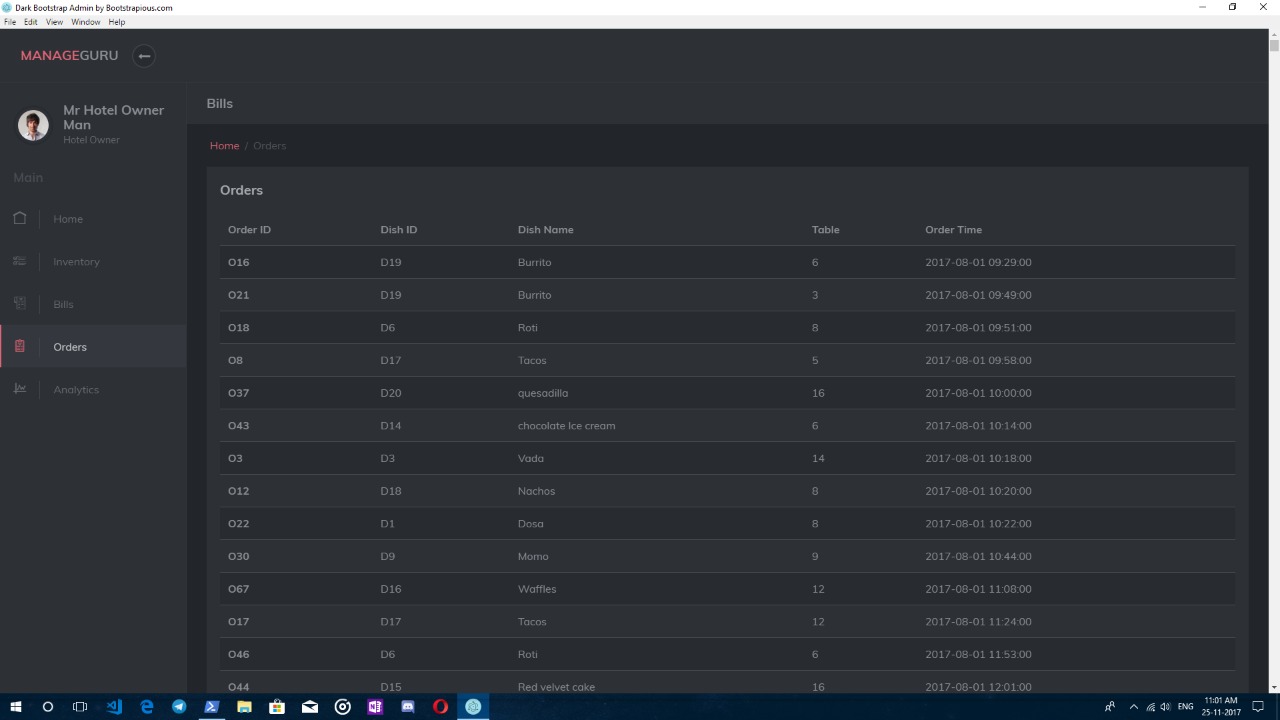
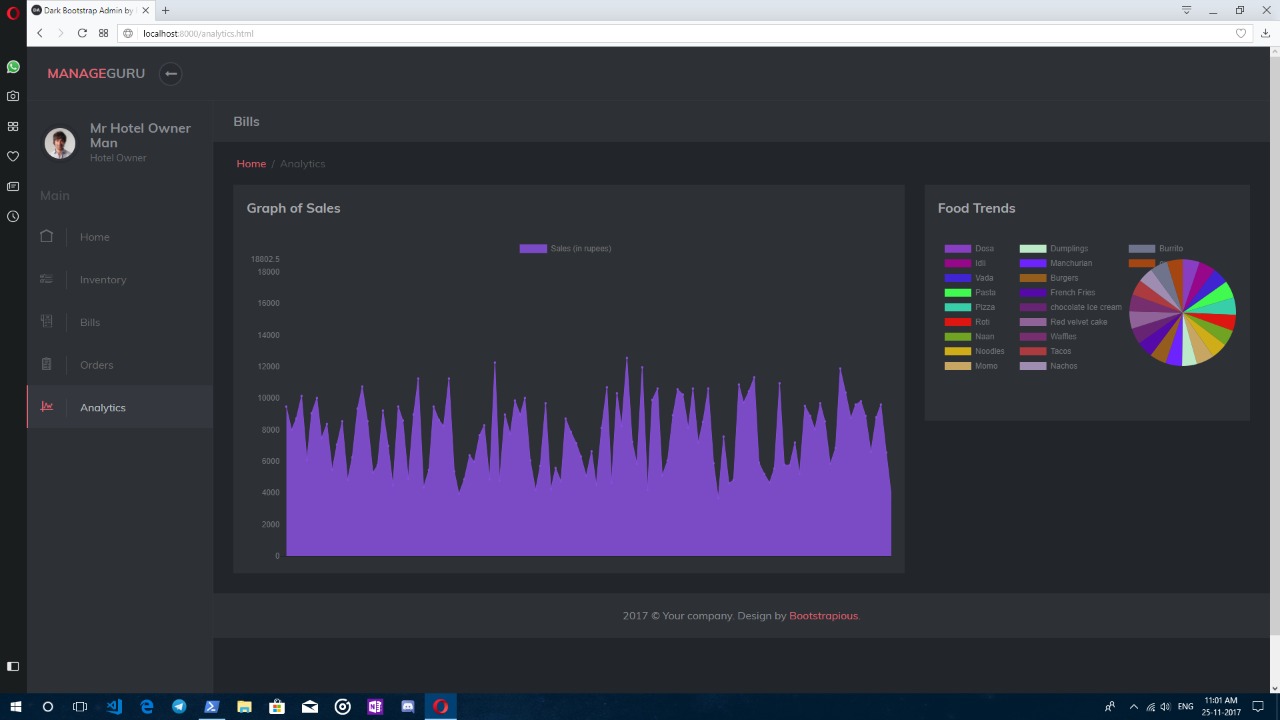
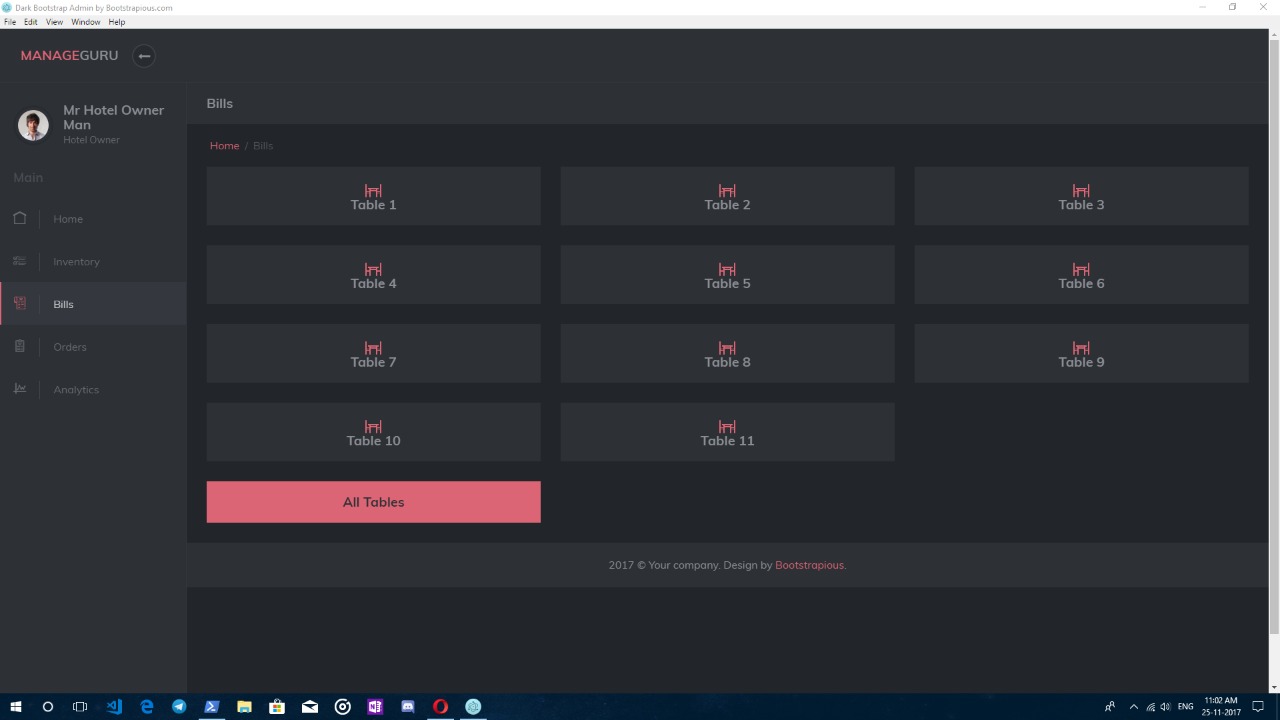


## 

## **Analytics pages on Electron:**







## **3.5 Other User Interfaces**

1. **Admin Page at the Management End**, used by the owner/manager present in the hotel to add new users to the management system, see all currently active users (attendance) and the order requests placed.
2. **Reports Page at the Monitor End**, used by the owner to monitor the working of the hotel remotely. The report page provides all the analytics report and real time statistics.

# 4.0 Detailed Design Approach

## **4.1 Design patterns considered and for what reason**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Design Pattern** | **Intent of this pattern** | **Rationale for choosing or not choosing** |
| 1 | Inheritance | To make a concrete class layout. | To have a clear idea of a class hierarchy |
| 2 | Abstraction | Abstraction as is. | To provide an abstract way of looking at data and behaviour |
| 3 | Enumeration | Ordered listing of all the items in a collection. | To provide a few fixed objects of a datatype with a few objects |
| 4 | Decorator | Easily extend functionality using callbacks. | Common and easy javascript pattern to elegantly extend code. |
| 5 | Adaptor | Change the interface of a function allowing its functionality. | Again, easy to implement pattern on javascript suiting our web platforms to change function/class interface. |
| 6 | Strategy | An interchangeable set of algorithms that encapsulate each other. | Provide a similar service abstraction with many services under one hood. |
| 7 | Async Event Based Programming | The program isn’t block from handling events while an event is being handled. | To not have our web service become unusable due to each active client. |
| 8 | Lock | Lock a resource to a particular user. | To ensure synchronised access to a modified resource, nearly always important in concurrent applications. |
| 9 | Thread Pool | Maintains multiplethreads waiting fortasks to be allocated forconcurrent execution by the supervising program. | To easily pool and manage a large number of threads for a large number of tasks. Useful in scaling web servers and runtimes that operate as such. |