****

**Report 2** **：** **Restaurant Automation**

**实现餐厅自动化系统**

**小组成员**

夏瑞

杨帆

吴晨

张天霖

**小组分工：**

夏瑞：负责交互图、算法、网络端用户界面与设计、员工网络端实现、内部数据测试

杨帆：负责数据库E-R图、数据库设计、系统架构设计

吴晨：负责类图、算法、Android端用户界面与设计

张天霖：负责数据类型、追溯矩阵、设计模式、测试设计

目录

[交互图： 4](#_Toc449393309)

[类图和接口规范： 10](#_Toc449393310)

[类图： 10](#_Toc449393311)

[类描述： 11](#_Toc449393312)

[数据类型和操作符： 12](#_Toc449393313)

[追溯矩阵： 15](#_Toc449393314)

[设计模式： 16](#_Toc449393315)

[软件设计模式： 16](#_Toc449393316)

[OCL（对象约束语言契约）： 17](#_Toc449393317)

[系统架构和系统设计： 19](#_Toc449393318)

[架构设计： 19](#_Toc449393319)

[确定子系统： 20](#_Toc449393320)

[映射子系统到硬件： 20](#_Toc449393321)

[持久数据存储： 20](#_Toc449393322)

[全局控制流： 21](#_Toc449393323)

[硬件需求： 21](#_Toc449393324)

[算法和数据结构： 21](#_Toc449393325)

[用户界面与实现： 25](#_Toc449393326)

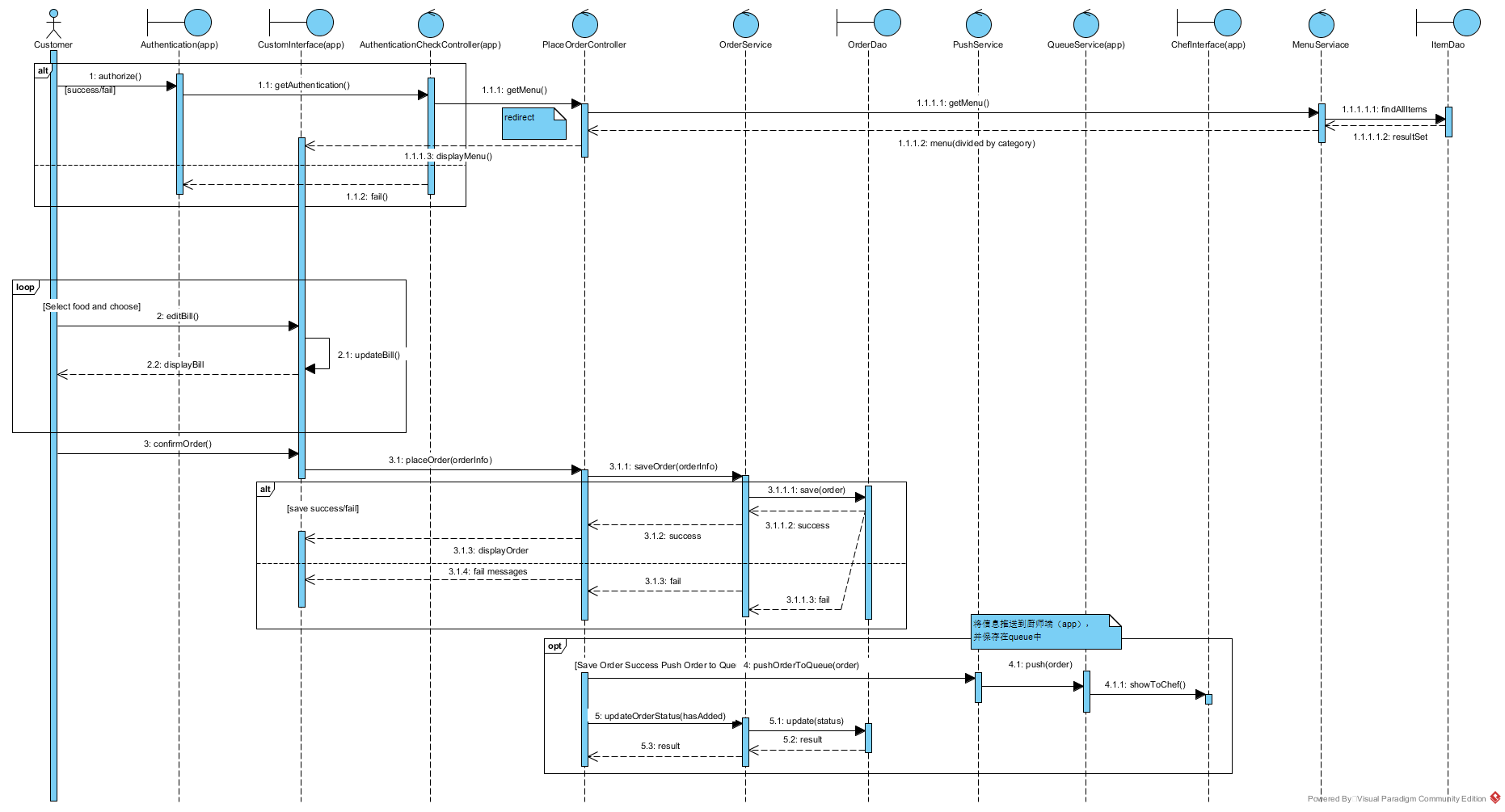
[测试设计： 32](#_Toc449393327)

[项目管理和历史工作： 35](#_Toc449393328)

[工作计划（甘特图）： 36](#_Toc449393329)

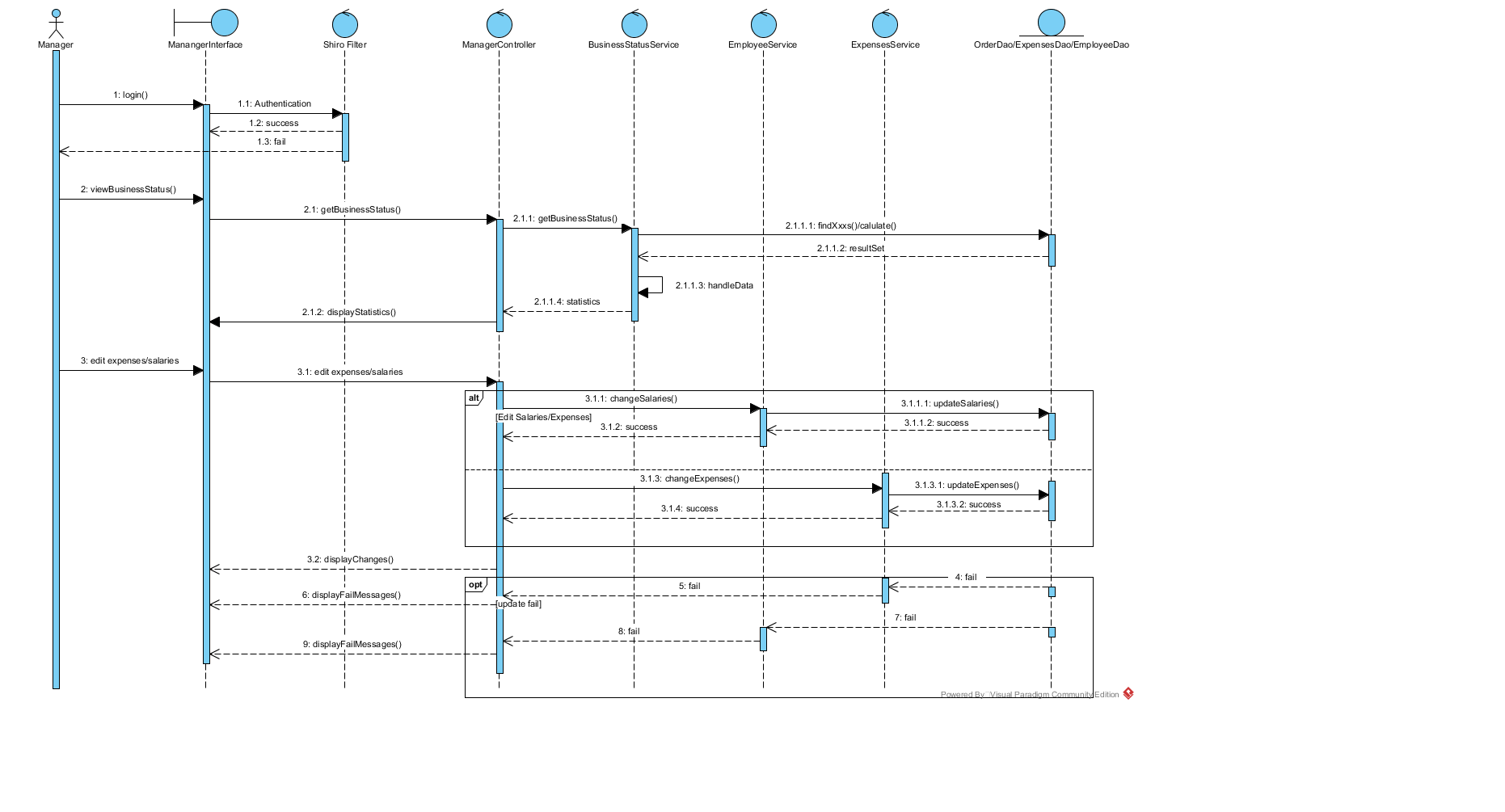
# **交互图**：

用例1 顾客订餐



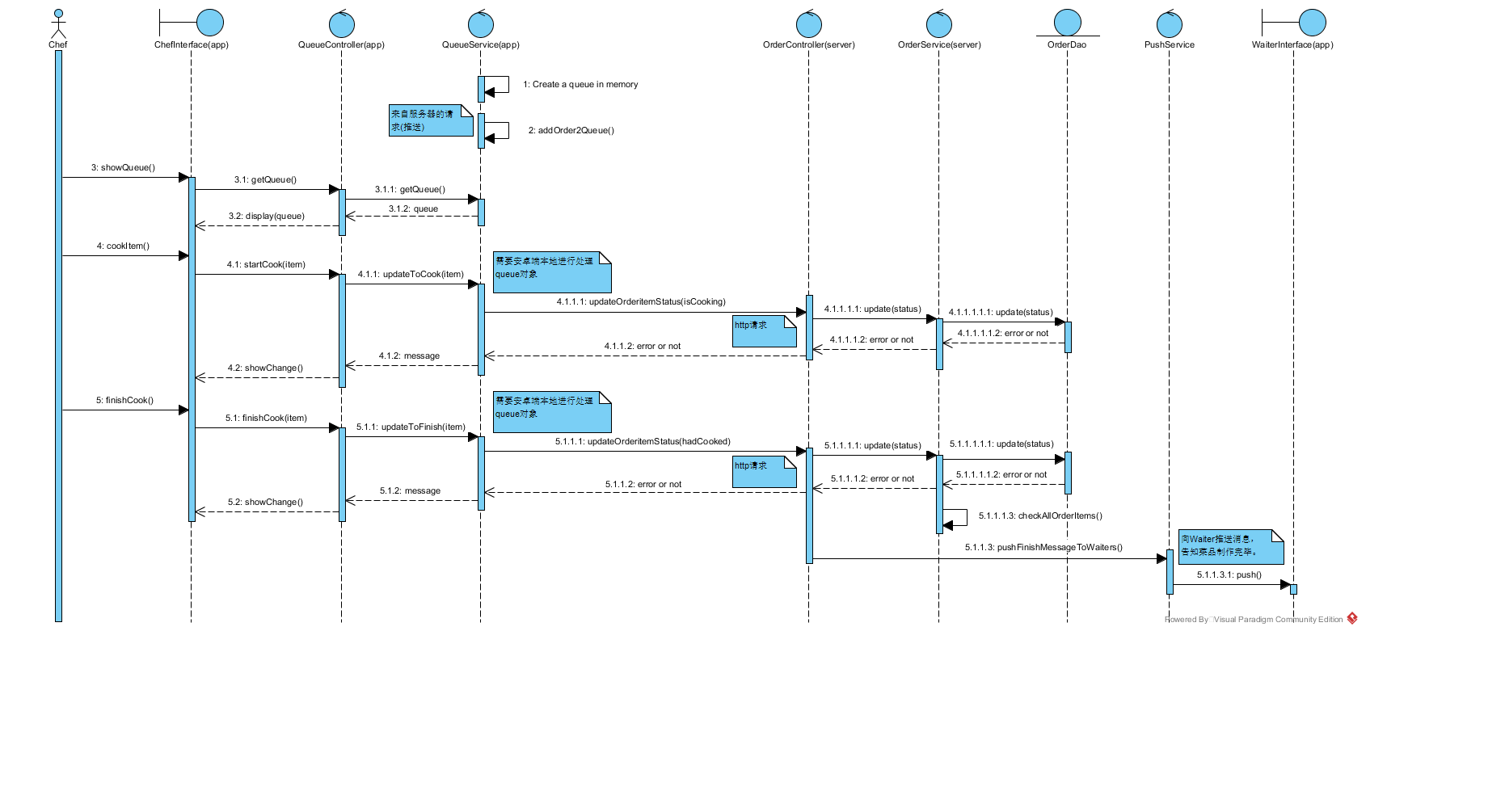
顾客可以根据顾客的用户界面进行点餐，并且显示订单。只要自己的订单没有确定，顾客就可以进行编辑。订单确定后，会将信息推送到厨师端，并保存在queue中。

用例2 查看业务状态



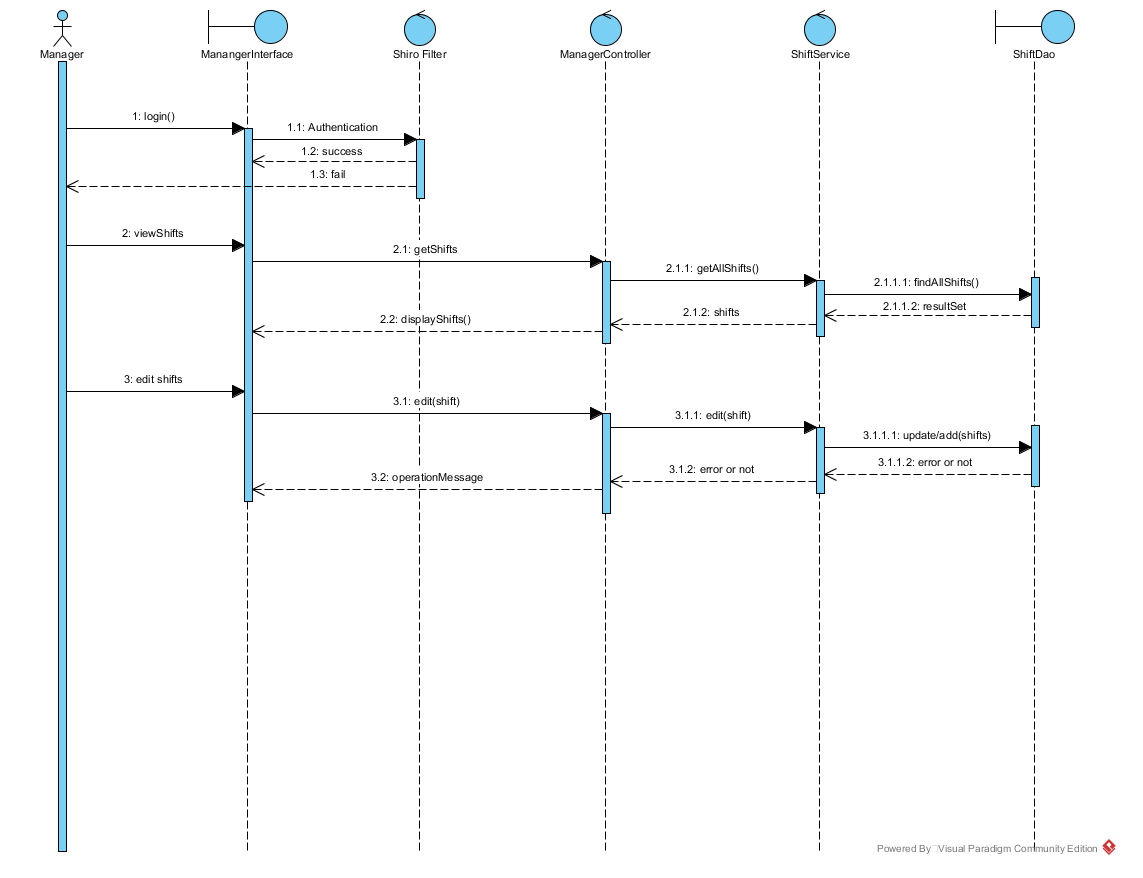
管理者可以登录管理者的用户界面查看业务状态。经理可以查看或编辑员工的工资，同时还可以查看餐厅的年（月）的利润与亏损。能有效地和数据库进行相互通信，进行存取。

用例3 管理订餐队列



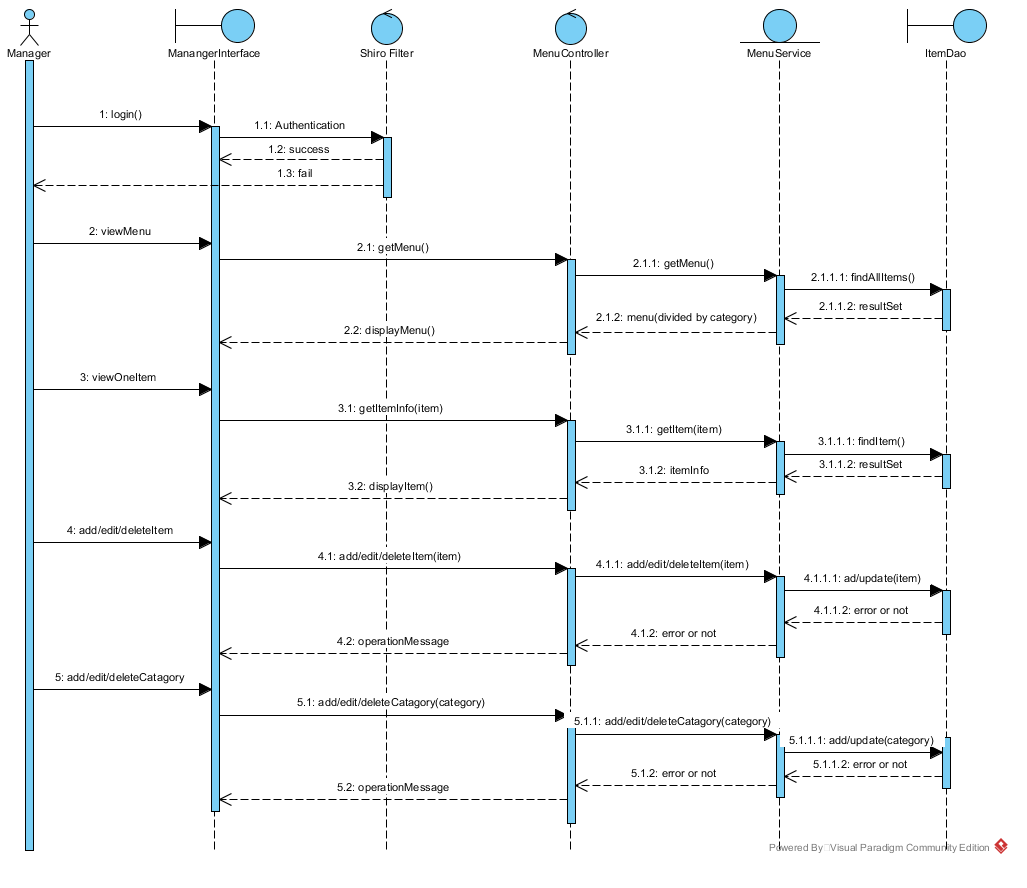
厨师可以查看由服务生或客户添加到队列的订单。厨师可以访问他的界面查看队列。厨师还可以与界面进行交互。厨师完成菜品制作后，他能进入完成的订单，从而消除订单未完成状态。这也能引发与服务员的通信，通知服务员取菜。

用例4 管理工作调度



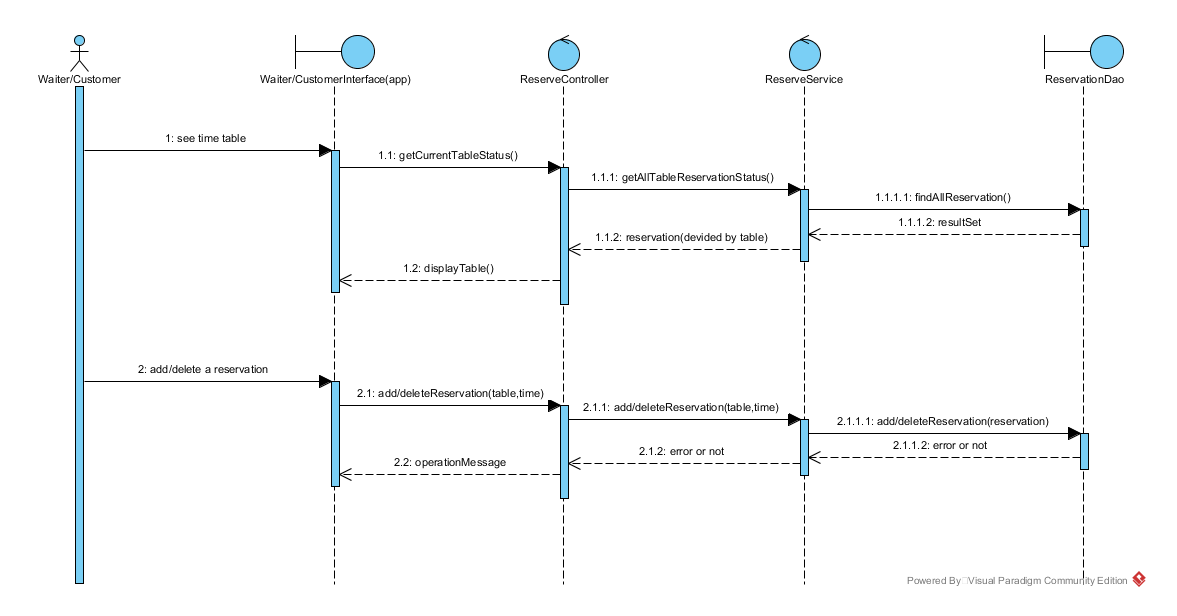
管理者可以进入管理界面对员工工作调度表进行查询和修改，对员工进行工作调度。

用例5 管理菜单



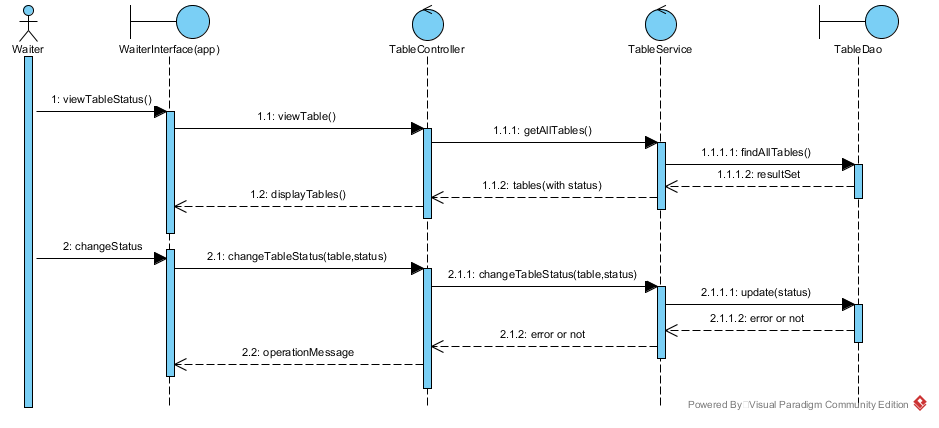
管理者可以进入管理者界面来管理菜单。管理者有能力添加或删除一个项目来更新的菜单。

用例6 预订座位



服务员和顾客能够进入相应的用户界面对餐厅的座位进行预订，以方便用餐时可以拥有座位。

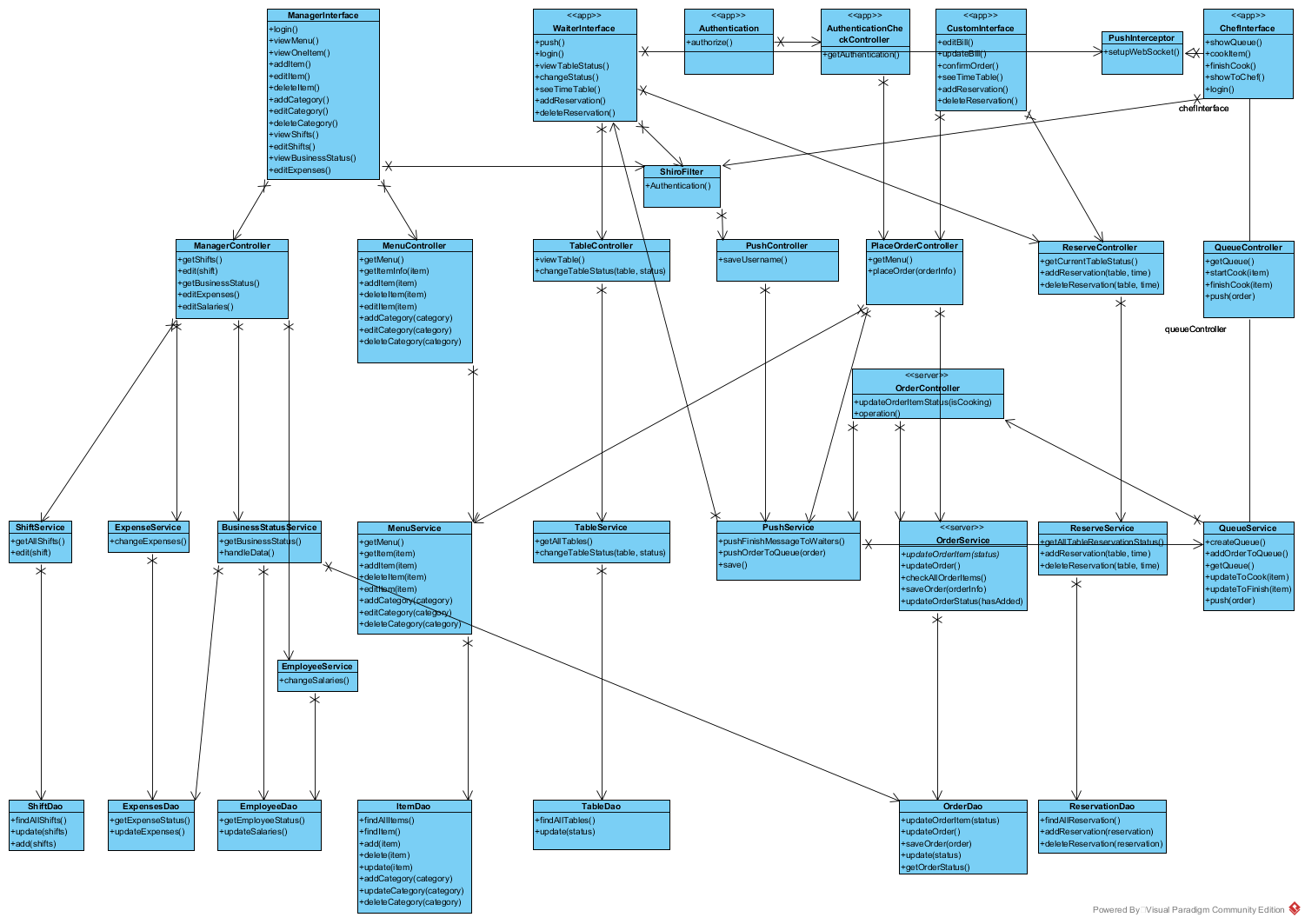
用例7 管理桌子



服务员可以进入自己的用户界面查看桌子状态并可对桌子状态进行改变，更改是否需要清洁

# 类图和接口规范：

## 类图：



## 类描述：

类图包含以下的类：

ManagerInterface-允许管理者进行登录、查看菜单、改变菜单、查看改变轮班、查看利润损失等

WaiterInterface-允许服务员进行登录、查看改变桌子状态、查看改变预订等功能

AuthenticationCheckController-权限认证控制类

CustomInterface-允许顾客改变预订、点菜、查看时间表、结账等功能

PushInterceptor-推送（设置websocket）

ChefInterface-允许厨师登录、查看序列、制作、完成制作等功能

ShiroFilter-认证授权

ManagerController-管理者控制类

MenuController-菜单控制类

TableController-餐桌控制类

PushController-推送控制类

PlaceOrderController-点餐控制类

ReserveController-预订控制类

QueueController-序列控制类

OrderController-订单控制类

ShiftService-轮换服务

ExpenseService-费用服务层

BusinessStatusService-业务状态服务层

MenuService-菜单服务层

TableService-餐桌服务层

PushService-推送服务层

OrderService-订单服务层

ReserveService-预订服务层

QueueService-序列服务层

EmployeeService-员工服务层

ShiftDao-轮换Dao层

ExpensesDao-费用Dao层

EmployeeDao-员工Dao层

ItemDao-菜单项Dao层

TableDao-餐桌Dao层

OrderDao-订单Dao层

ReservationDao-预订Dao层

# 数据类型和操作符：

ManagerInterface：

ManagerInterface使管理者能与系统进行交互进行相应的操作

方法：

|  |  |
| --- | --- |
| Login（） | 用于登录 |
| viewMenu（） | 查看菜单 |
| viewOneItem（） | 查看一个菜单项 |
| addItem（） | 增加菜单项 |
| editItem（） | 修改菜单项 |
| deleteItem（） | 删除菜单项 |
| addCategory（） | 增加菜单类别 |
| editCategory（） | 修改菜单类别 |
| deleteCategory（） | 删除菜单类别 |
| viewShifts（） | 查看轮班 |
| editShifts（） | 修改轮班 |
| viewBusinessStatus（） | 查看业务状态 |
| editExpenses（） | 修改费用 |

WaiterInterface：

WaiterInterface使服务员能与系统进行交互进行相应的操作

|  |  |
| --- | --- |
| Push（） | 进行推送 |
| login（） | 用于登录 |
| viewTableStatus（） | 查看餐桌状态 |
| changeStatus（） | 改变状态 |
| seeTimeTable（） | 查看时间表 |
| addReservation（） | 增加预订 |
| deleteReservation（） | 删除预订 |

CustomInterface：

CustomInterface使顾客能与系统进行交互进行相应的操作

|  |  |
| --- | --- |
| editBill（） | 修改账单 |
| confirmOrder（） | 进行订餐 |
| seeTimeTable（） | 查看时间表 |
| addReservation（） | 增加预订 |
| deleteReservation（） | 删除预订 |

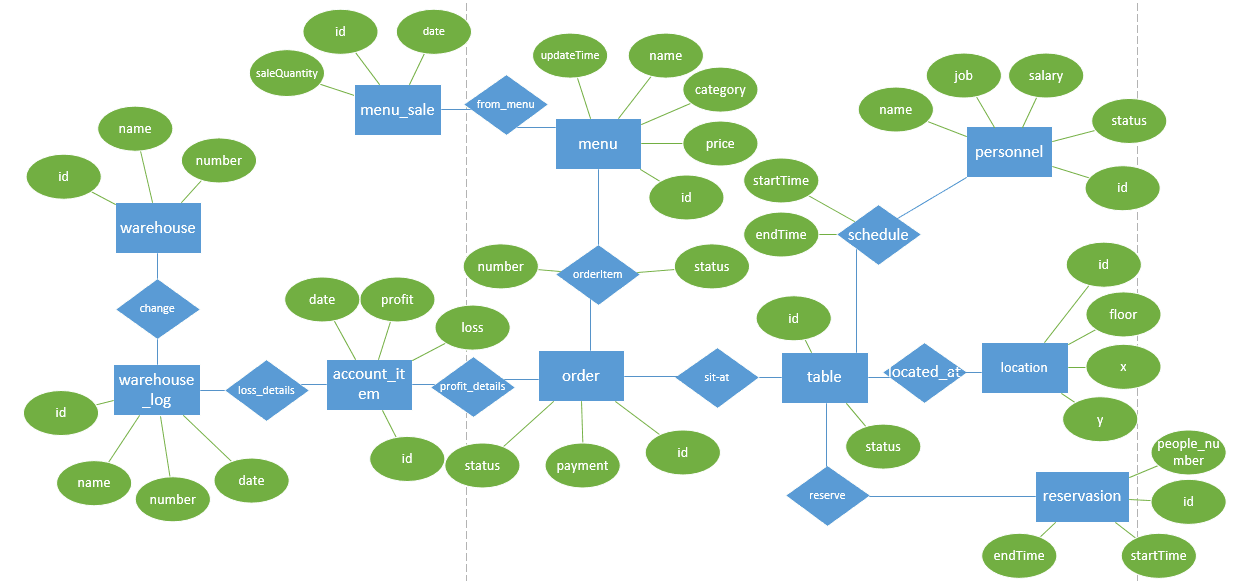
ChefInterface：

ChefInterface使厨师能与系统进行交互进行相应的操作

|  |  |
| --- | --- |
| showQueue（） | 展示订单序列 |
| cookItem（） | 制作订单项 |
| finishCook（） | 完成制作 |
| showToChef（） | 向服务员展示 |
| login（） | 用于登录 |

除interface外还有每个类对应的controller类和service、Dao层。

数据库E-R图（实体-联系图）模型



E-R图对应数据库属性：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 表 | 属性 | 类型 |
| 实体 | **personnel** | id | INTEGER |
|  |  | name | VARCHAR(10) |
|  |  | job | VARCHAR(10) |
|  |  | salary | NUMBER(7,2) |
|  |  | status | VARCHAR(10) |
|  | **table** | id | INTEGER |
|  |  | status | VARCHAR(10) |
|  |  | people\_number | INTEGER |
|  | **location** | id | INTEGER |
|  |  | floor | INTEGER |
|  |  | x | NUMBER(7,2) |
|  |  | y | NUMBER(7,2) |
|  | **reservasion** | id | INTEGER |
|  |  | people\_number | INTEGER |
|  |  | starttime | DATE |
|  |  | endtime | DATE |
|  | **menu** | id | INTEGER |
|  |  | name | VARCHAR(20) |
|  |  | category | VARCHAR(10) |
|  |  | price | NUMBER(7,2) |
|  |  | updatetime | DATE |
|  | **order** | id | INTEGER |
|  |  | payment | NUMBER(7,2) |
|  |  | status | VARCHAR(10) |
|  | **menu\_Sale** | id | INTEGER |
|  |  | date | DATE |
|  |  | saleQuantity | INTEGER |
|  | **account\_item** | id | INTEGER |
|  |  | date | DATE |
|  |  | profit | NUMBER(7,2) |
|  |  | loss | NUMBER(7,2) |
|  | **warehouse\_log** | id | INTEGER |
|  |  | name | VARCHAR(10) |
|  |  | number | INTEGER |
|  |  | date | DATE |
|  | **warehouse** | id | INTEGER |
|  |  | name | VARCHAR(10) |
|  |  | number | INTEGER |
| 关系 | **schedule** | starttime | DATE |
|  |  | endtime | DATE |
|  | **order\_item** | number | INTEGER |
|  |  | status | VARCHAR(10) |

# 追溯矩阵：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Software classes | | | | | | | | | | | | |
| Domian  concepts | Managerinterface | Waiterinterface | Customeinterface | Chefinterface | Authenticationcheckcontroller | Managercontroller | Menucontroller | Tablecontrolller | Pushcontroller | Placeordercontroller | Reservecontoller | Queuecontroller | Ordercontroller |
| Manager | √ |  |  |  | √ | √ | √ |  |  |  |  |  |  |
| Waiter |  | √ |  |  | √ |  |  | √ | √ |  | √ |  |  |
| Customer |  |  |  |  |  |  |  |  |  | √ | √ |  |  |
| Chef |  |  |  | √ | √ |  |  |  |  |  |  | √ |  |
| Profit/loss | √ |  |  |  |  | √ |  |  |  |  |  |  |  |
| Orderitem | √ |  |  | √ |  |  |  |  |  |  |  |  | √ |
| Orderstatus | √ |  |  | √ |  |  |  |  |  |  |  |  | √ |
| Menumodifier | √ |  |  |  |  | √ | √ |  |  |  |  |  |  |
| Order |  | √ | √ |  |  |  |  |  |  |  |  | √ | √ |
| Check |  | √ | √ |  |  |  |  |  |  |  |  |  |  |
| Schedule | √ | √ |  | √ |  | √ |  |  |  |  |  |  |  |

这些领域概念都来源于餐厅自动化系统，每一个都与软件的一个或几个类相结合。比如， Managerinterface（管理者界面）有Manager、Profit/loss、Orderitem、Orderstatus、Menumodifier、Schedule这些领域概念，其余的类也类同。

# 设计模式：

## 软件设计模式：

Strategy（策略模式）

策略模式定义了一系列的算法，并将每一个算法封装起来，而且使它们还可以相互替换。策略模式让算法独立于使用它的客户而独立变化。本系统在处理顾客来到餐厅后合理分配桌子时，会根据情况进行算法的选择（考虑用餐人数和到来时间），这会用到策略模式。

Facade（外观模式）

外观模式为子系统中的各类（或结构与方法）提供简明一致的界面，隐藏子系统的复杂性，使子系统更加容易使用。它是为子系统中的一组接口所提供的一个一致的界面。本系统设计阶段利用Controller层、Service层、Dao层进行分层，隐藏内部处理逻辑和与数据库交互的复杂性，利用接口为用户呈现相应界面。所以用到了外观模式。

Observer（观察者模式）

观察者模式完美的将观察者和被观察的对象分离开。举个例子，用户界面可以作为一个观察者，业务数据是被观察者，用户界面观察业务数据的变化，发现数据变化后，就显示在界面上。观察者模式在模块之间划定了清晰的界限，提高了应用程序的可维护性和重用性。观察者设计模式定义了对象间的一种一对多的依赖关系，以便一个对象的状态发生变化时，所有依赖于它的对象都得到通知并自动刷新。本系统设计在厨师完成订单推送到服务员端功能上用到了观察者模式，将完成订单实时进行推送，以实现尽快上菜的需求。

## OCL（对象约束语言契约）：

managerInterface::additem(string itemName, double Price): bool

//The manager is the one updating the menu

inv: self.manager -> True

//Item is not available on Menu

pre: MenuItem == NULL //Update item in menu

post: additem(string itemName, double Price) == true

managerInterface::deleteitem(itemInfo\* i): bool

//The manager is the one updating the menu

inv: self.manager -> True //Item is available on Menu

pre: MenuItem != NULL

//Removing Item succesful from menu

post: deleteitem(itemInfo\* i) == true

managerInterface::fire(eInfo\* e): bool

//The manager is the one firing the employee

inv: self.manager -> True

//employee is in database

pre: employeeinfo != NULL

//Removing employee succesful from database

post: fire(eInfo\* e) == true

managerInterface::hire(eInfo\* e): bool

//The manager is the one hiring the employee

inv: self.manager -> True

//employee is not in database

pre: employeeinfo == NULL

//Adding employee succesful from database

post: hire(eInfo\* e) == true

waiterInterface:placeOrder() : bool

//The Waiter is the one placing the order

Invariants: self.Waiter -> True

//The table has customers on it

Pre-Conditions: table.occupied -> True

//Order Succesful and pushed into Queue

Post-Condtions: placeOrder == true

waiterInterface:cancelOrder(orderinfo\* o) : bool

//The Waiter is the one placing the order

Invariants: self.Waiter -> True

//The table has customers on it

Pre-Conditions: table.occupied -> True

//Order Deleted and removed from Queue

Post-Condtions: cancelOrder(orderinfo\* o) == true

waiterInterface: changeStatus (tablestatus\* s) : bool

// The Waiter is the one changing table Status

Invariants: self.Waiter -> True

// The table has customers on it

Pre-Conditions: table.occupied -> True

// The table has not customers

Post-Condtions: table.occupied -> false

chefInterface: finishitem(item\*i) : bool

// The order has been in the queue

Invariants: self.chef -> True

// The item has not finished

Pre-Conditions:item.finished -> false

// The item has finished

Post-Condtions: item.finished -> true

customInterface: addreserve (tablereserved\* s) : bool

// The Waiter is the one add reservation

Invariants: self.customer -> True

// The table has not been resreved

Pre-Conditions: table. reserved -> false

// The table has been resreved

Post-Condtions: table. reserved -> true

customInterface:placeOrder() : bool

//The custom is the one placing the order

Invariants: self. custom -> True

//The table has customers on it

Pre-Conditions: table.occupied -> True

//Order Succesful and pushed into Queue

Post-Condtions: placeOrder == true

custom Interface:cancelOrder(orderinfo\* o) : bool

//The custom is the one placing the order

Invariants: self. custom -> True

//The table has customers on it

Pre-Conditions: table.occupied -> True

//Order Deleted and removed from Queue

Post-Condtions: cancelOrder(orderinfo\* o) == true

## 系统架构和系统设计：

### 架构设计：

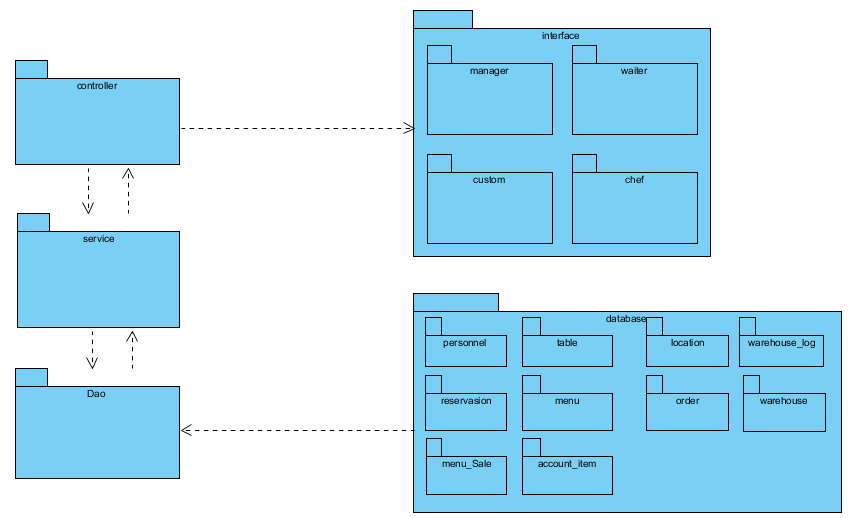
我们设计的架构是基于C/S（客户端-服务器）模型的，客户端直接连接集成服务器。这种系统的工作原理是建立统一的基础设施，所有通信都通过服务器与客户端进行连接与请求。

服务器搭建有oracle数据库，以及满足请求服务的程序。从数据库的角度来看，它的作用是储存员工信息，菜单信息，订单信息，餐桌状态等其他相关信息并进行管理。根据不同登录用户的权限进行访问级别的改变。除了为员工提供所需数据库外，还保留本地数据库，

该数据库发送和接收与程序相关的信息。有了这个系统，服务器可以与客户端进行相应的信息传送。

在实现餐厅自动化上，该系统采用了几种不同类型的用户（客户端）与其相应的功能；用户首先进行登录通过服务器进行验证，然后会授予客户端不同的权限。比如，管理者登录到系统可以进行菜单的修改和员工轮班的改变，而服务员登录到系统可以对订单进行处理并进行餐桌的预订、桌子状态的改变等。

### 确定子系统：



包图由系统中的组成元素组成。首先，界面包中包含了各种类型的员工（管理者、服务员、厨师）和顾客，为不同用户提供不同的用户界面和功能选择。数据库保存所有相关信息，如菜单项、订单、桌子状态、库存状态等。控制器负责具体的业务模块流程的控制，在此层里面要调用Serice层的接口来控制业务流程。Service层主要负责业务模块的逻辑应用设计。Dao层主要是做数据持久层的工作，负责与数据库进行联络与通信。

### 映射子系统到硬件：

由于客户端-服务器模型体系结构的性质，该系统的硬件层次结构是比较简单的。服务器上运行计算机上的数据库和数据。员工及顾客通过他们设备访问餐厅自动化应用程序，其中包含通信接口和控制器。

### 持久数据存储：

该系统存储的信息需要进行持久保存，以方便进行数据管理和数据的交互。在oracle中，我们用数据表来保存这些数据。备份确保数据的有效性，维护确保数据的可靠性。

### 全局控制流：

线性执行：

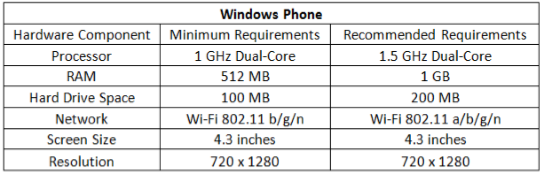
这是一个过程驱动的软件。几乎都以线性的方式来执行。顾客进入餐厅会利用程序选择位置。然后顾客会通过平板设备或服务员通过平板设备进行点菜。订单点菜完毕后会将订单传递给厨师，当厨师准备好时，将通知服务员将菜品直接到客户。当顾客吃完他会通过平板付账，然后服务员会标志桌子为脏状态。这清楚地表明，所有的行动都是一种线性的方式，取决于先前的操作。

并发：

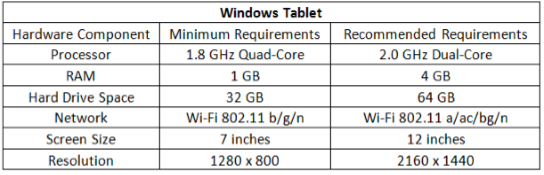
系统将包含多个线程，这将涉及多个程序同时独立运行。多个客户将在同一时间下订单，这就是为什么我们需要并发。该解决方案是将多个线程运行到队列中。另一个原因是管理者编辑库存或费用时，一个线程将处理数据库中的更新。不需要同步，因为每一个线程都是独立的。

### 硬件需求：

采用手机的便携性，员工和客户现在可以利用所有功能。这些功能需要以下：



利用平板电脑，员工和客户可以使用所有功能。这些功能需要以下：



### 算法和数据结构：

获取分析数据

1.产品的收益

Sellmap[productId,sells]

for(All orderitems in the db)

{

//pick up the productId,

//find the entry in sellmap

sells++

}

Show(sellmap)

2.菜品的受欢迎程度

SortMapBysells(sellMap);

//put top10 and bottom 10 items

3.就餐时长

时长：（1）30分钟以下 （2）30分钟-60分钟 （3）60分钟-90分钟 （4）90分钟-120分钟 （5）120分钟以上 对应period[5]

for (All orders in period)

{

time1=place\_order\_timestamp;

time2=pay\_bill\_timestamp;

duration=time-time1;

Switch(duration){

//put this order into its group

}

}

ShowAsChart(period);

4.等待订单的时长

for(All orders need to be calculate)

{

waitTime=0;

lastDish=null//Because we have not provide food to customer before their first dish comes.Maybe the restaurant offer some food for

customer on free,in order to decrease the waiting time.In that case,the lastDish can be initialized by a not-null object.

Current =0 //The relative time to the order-start timestamp

For(every orderItem in the order Order By serveTime Asc)

{

currentDish = orderItem.getDish;

if(lastDish.eatTime<orderItem.serveTime){

//last dish may have been eaten up.while the next dish has not been served.the customer may wait.

waitTime=orderItem.serveTime-current-lastDish.eatTime

}//End if

Else{

//the customer may eating the food while the next dish is coming.No wait time.

}

lastDish = currentDish;

current = orderItem.serveTime

}//End for

}//End for

//calculate the average waitTime and show the waitTime distribute Chart

动态加载时间表算法：

private LinearLayout getTimeTableView(List<TimeTableModel> model) {

LinearLayout mTimeTableView = new LinearLayout(getContext());

mTimeTableView.setOrientation(VERTICAL);

int modesize = model.size();

if (modesize <= 0) {

mTimeTableView.addView(addStartView(MAXNUM));

} else

for (int i = 0; i < modesize; i++) {

if (i == 0) {

mTimeTableView.addView(addStartView(model.get(0)

.getStartnum()));

mTimeTableView.addView(getMode(model.get(0)));

} else if (model.get(i).getStartnum()

- model.get(i - 1).getStartnum() > 0) {

mTimeTableView.addView(addStartView(model.get(i)

.getStartnum() - model.get(i - 1).getStartnum()-(model.get(i-1).getEndnum()-model.get(i-1).getStartnum())));//减去持续的时间 modifed by WuChen 3.24

mTimeTableView.addView(getMode(model.get(i)));

}

if (i + 1 == modesize) {

mTimeTableView.addView(addStartView(MAXNUM

- model.get(i).getEndnum()));

}

}

return mTimeTableView;

}

该系统将使用几种不同的数据结构来实现，其中包括队列、优先级队列和数组。利用不同的数据结构来实现程序中不同的功能。

当顾客进入餐厅，会记录用餐人数、到达时间的信息，然后会添加到服务员的优先级队列结构，通过优先级队列对顾客点餐顺序进行选择。

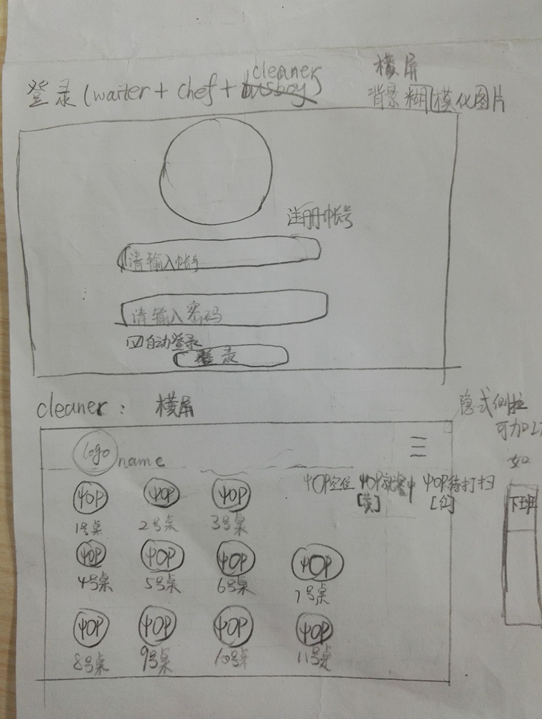
另一方面，厨师使用队列来管理所有的订单。根据FCFS（先来先服务）的

算法进行菜品的制作。

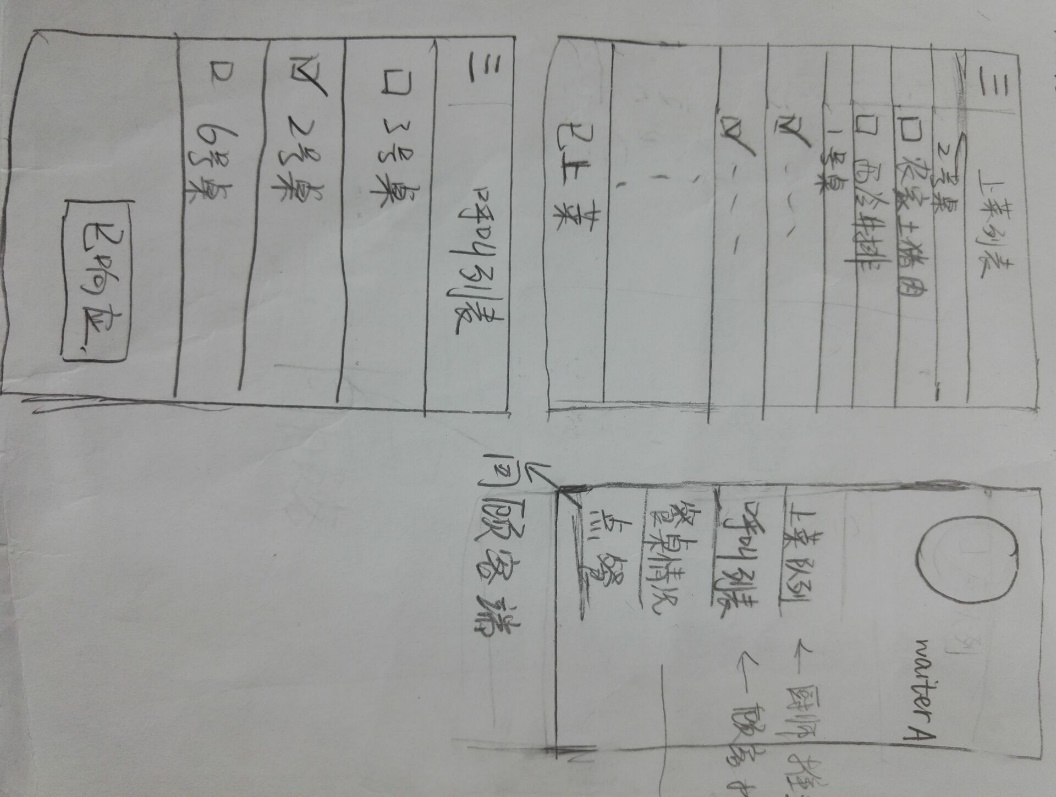
菜单项将被存储在数组中。从性能的角度，利用数组能够更好地进行统计分析并方便修改存储，被应用在菜单项上非常适合。

### 用户界面与实现：

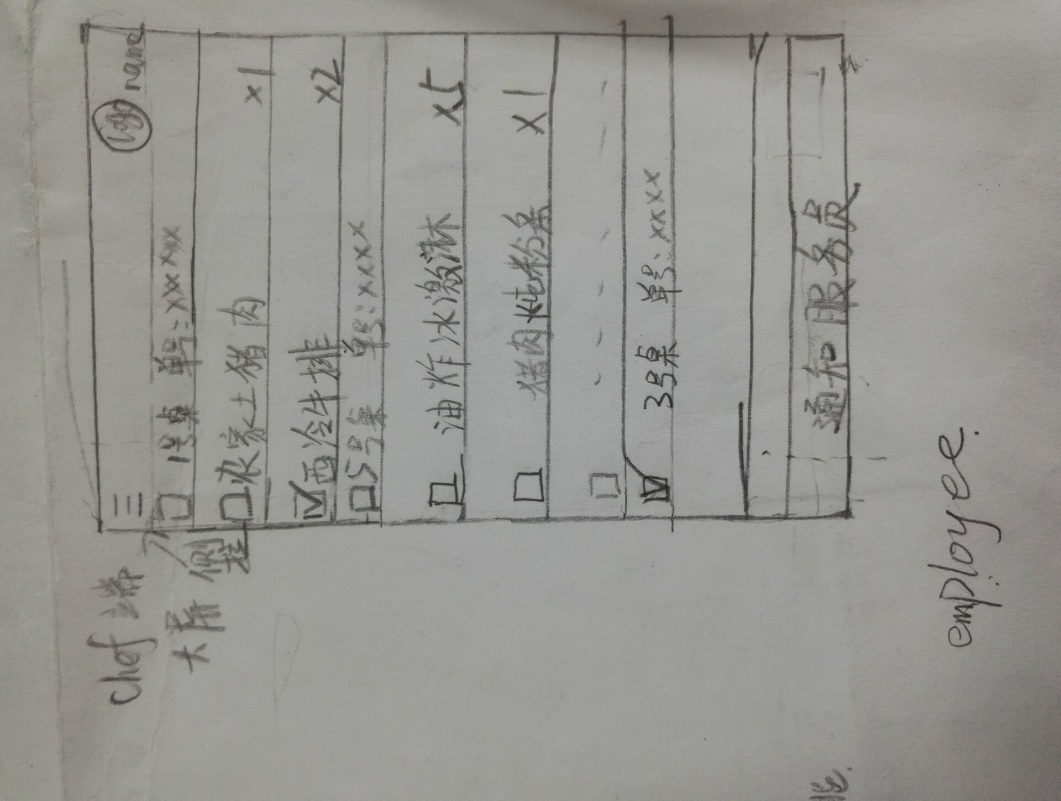
Android端界面草图：



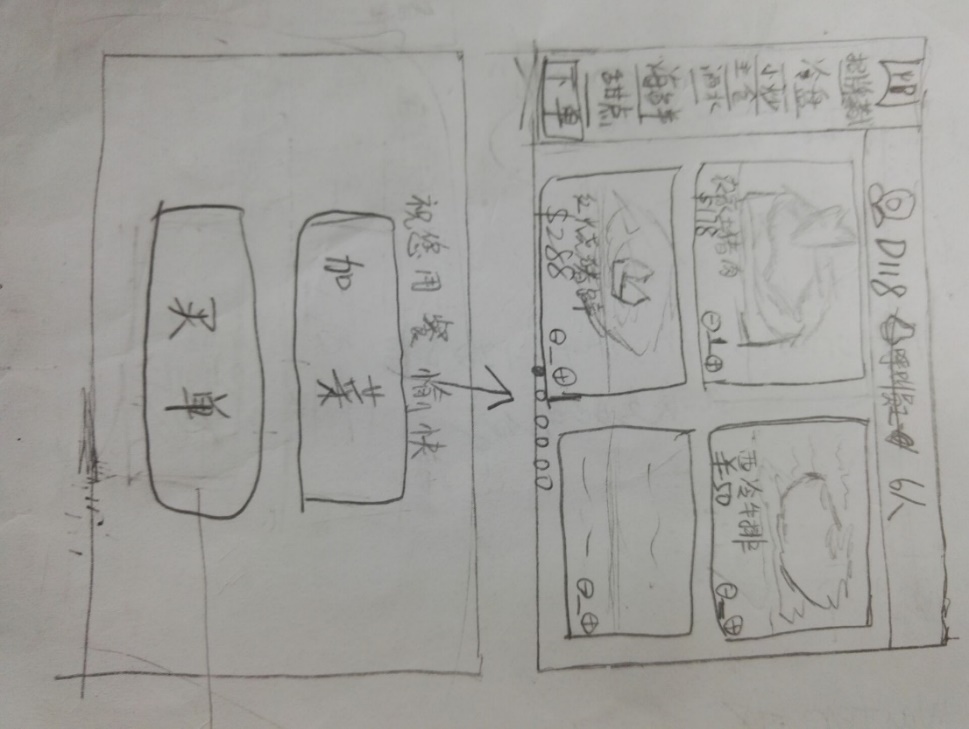
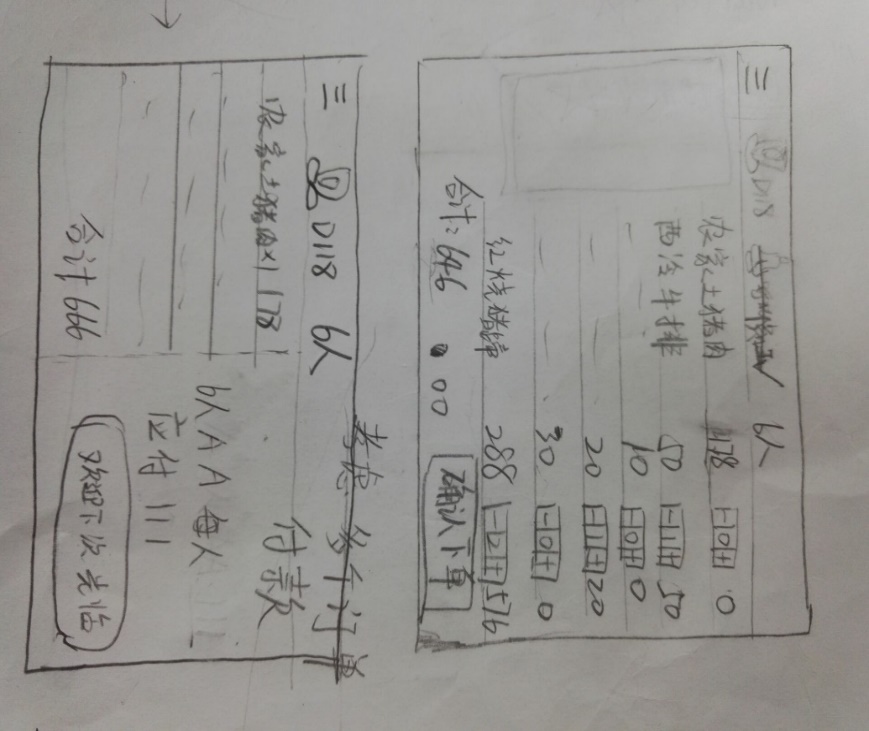
员工登录界面和清洁工查看桌子状态界面



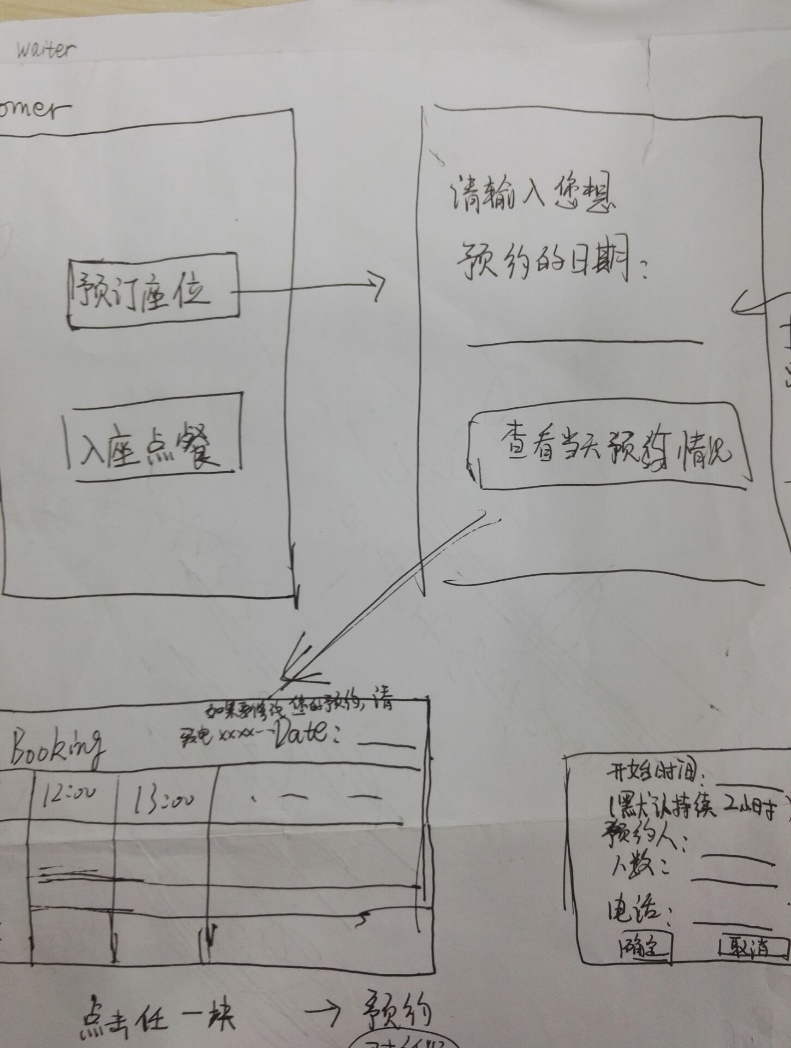
服务员工作界面（查看上菜列表、呼叫列表、餐桌情况和点餐功能）



厨师界面（查看订单并修改状态）

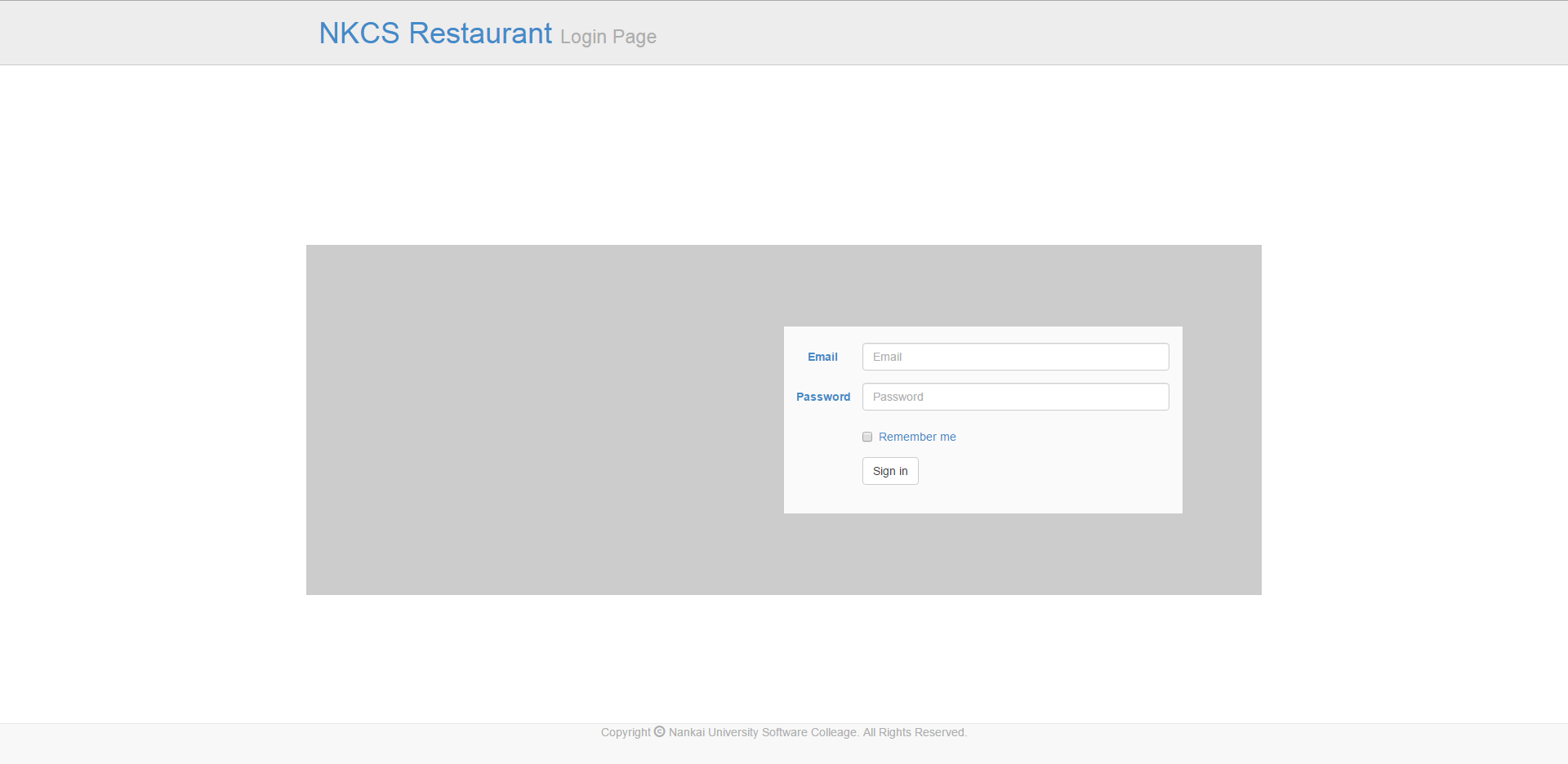


顾客点餐界面和付款界面

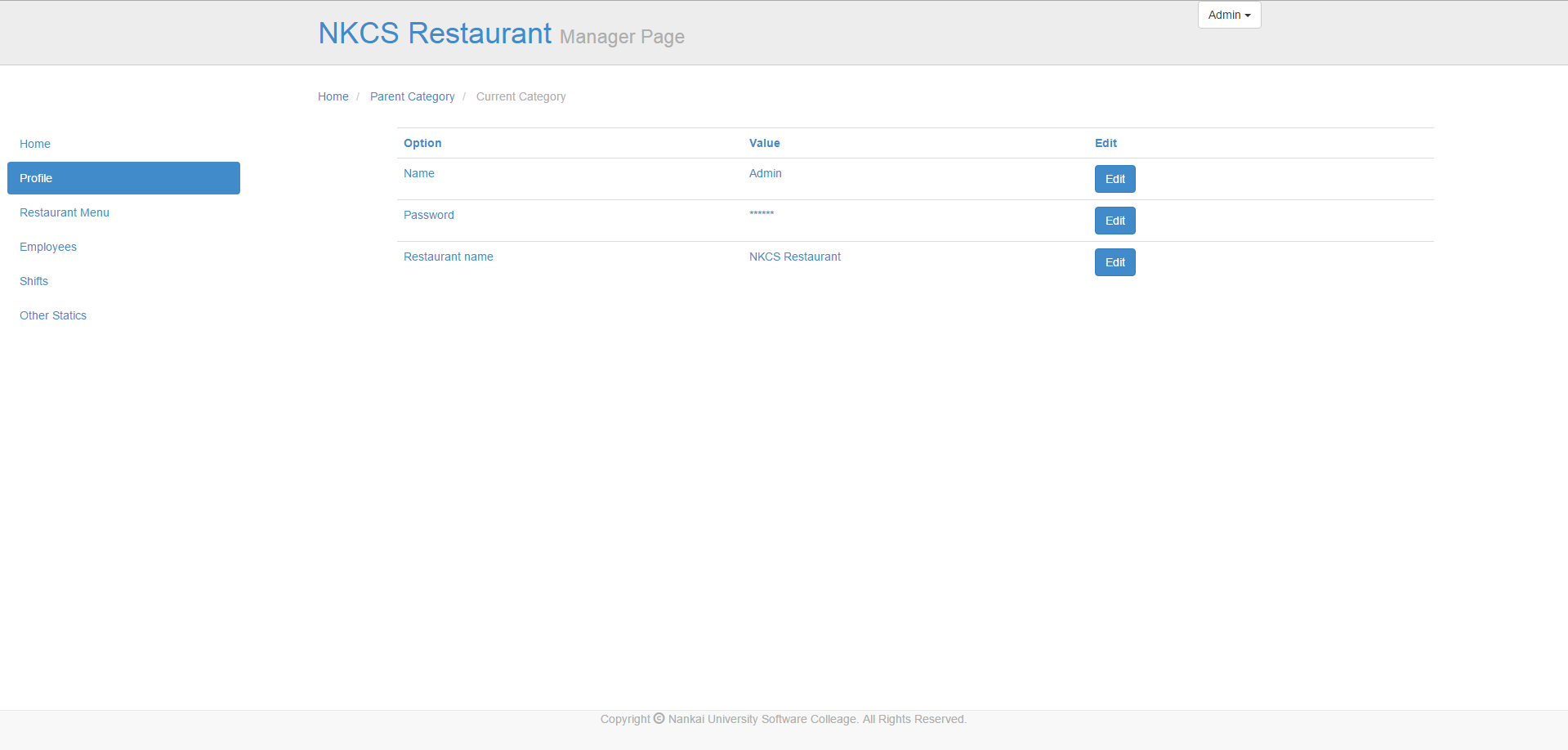


顾客或服务员预定座位界面

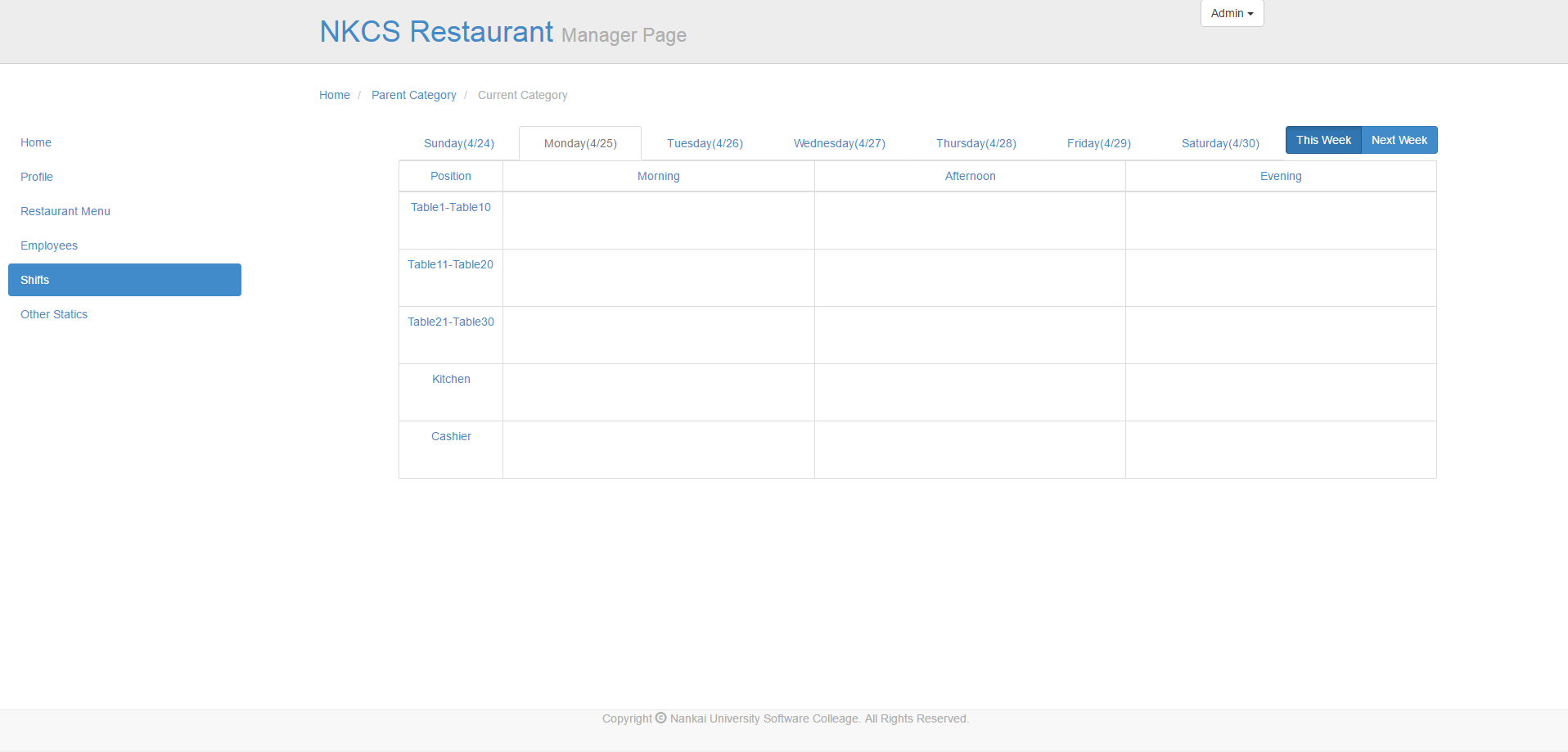
网络端界面：



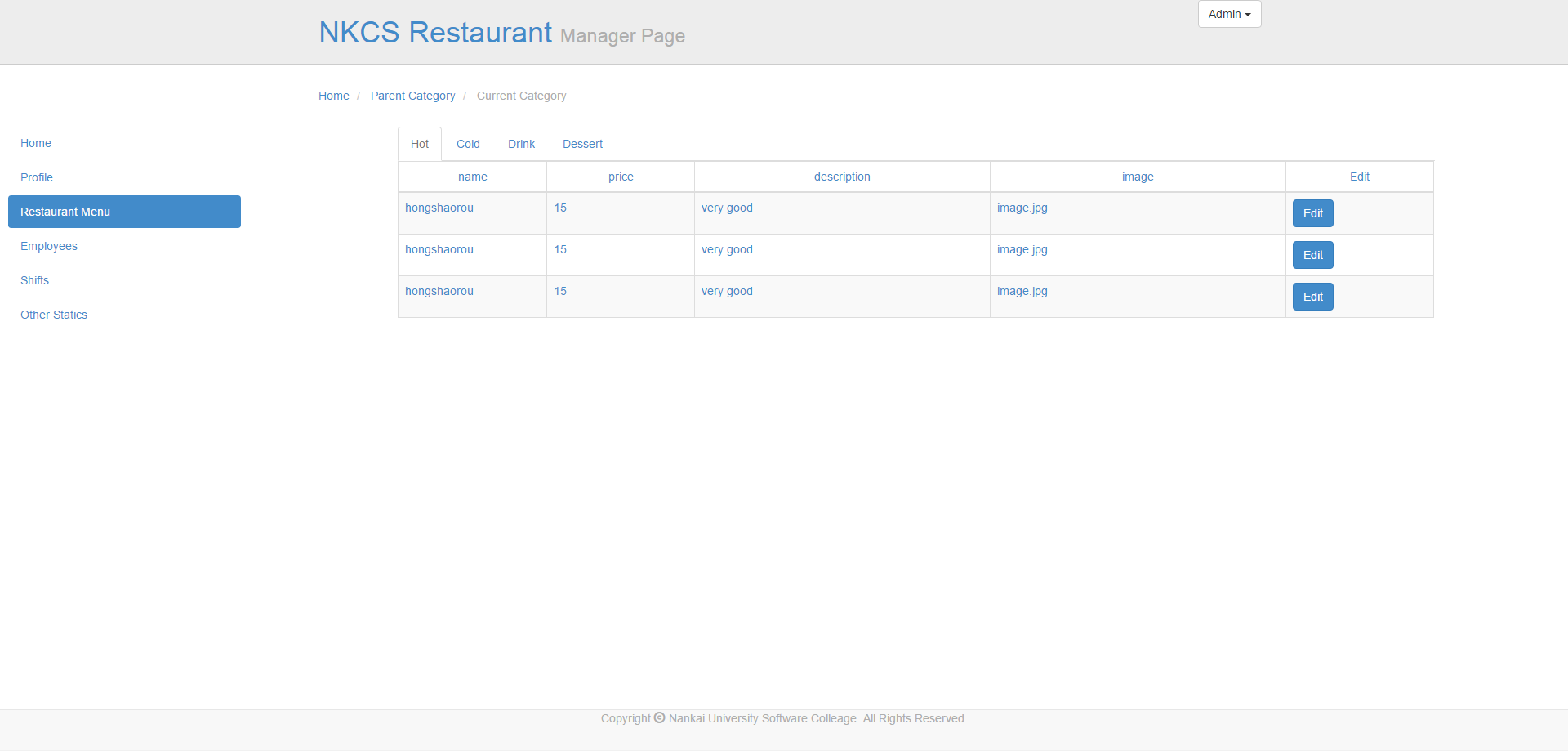
用户登录界面



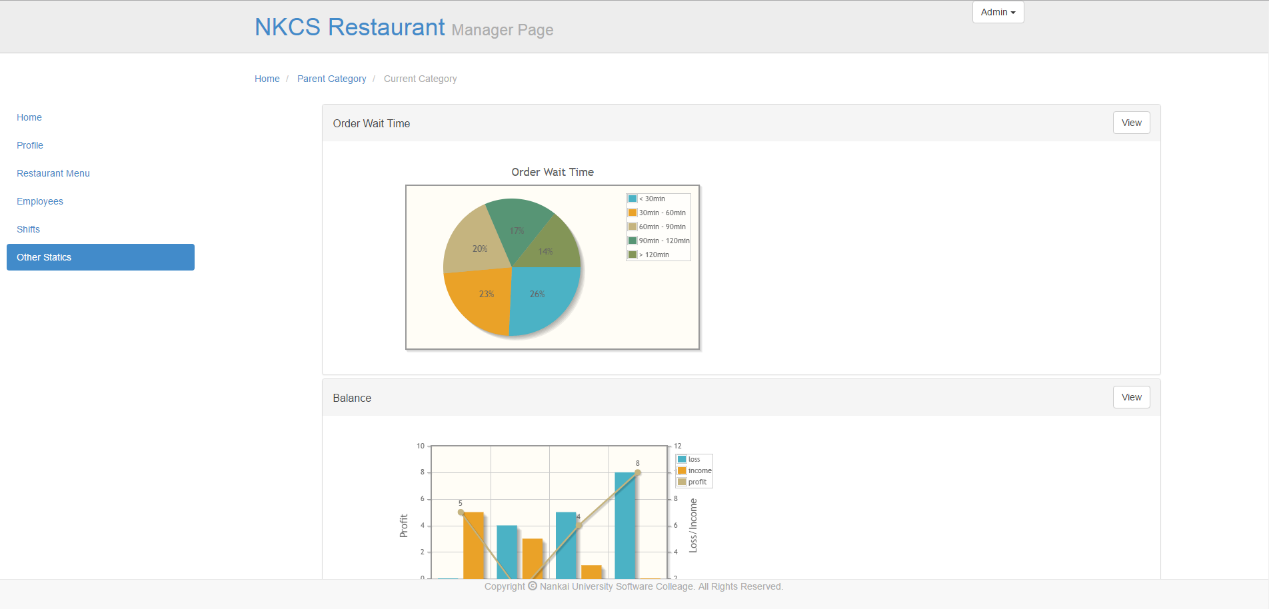
个人信息修改



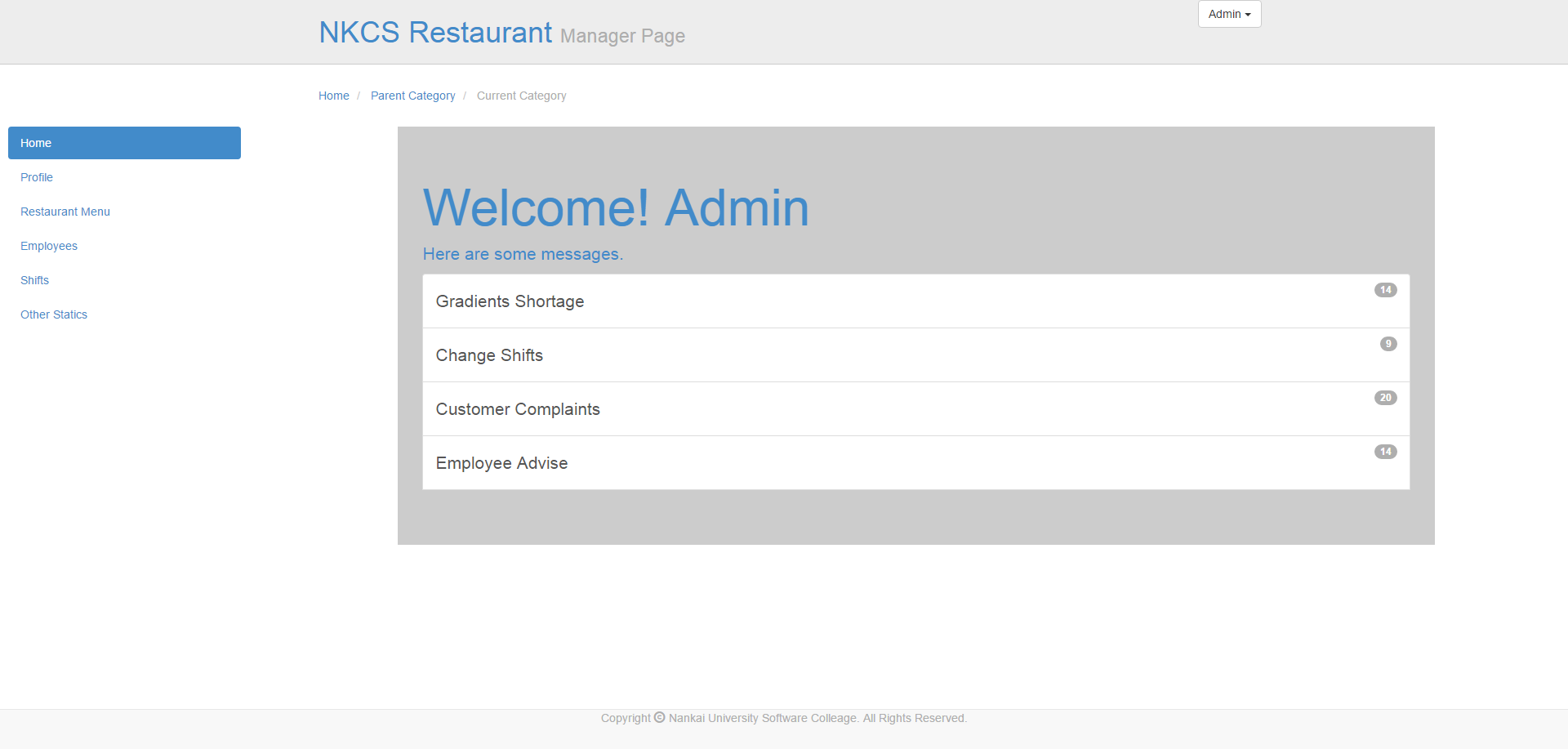
工作日程安排



菜品修改



其他统计数据



消息浏览

Manager的界面制作的的主要理念是:尽可能少的进行页面跳转。

所以在设计的过程中，使用了比较多的JavaScript和jQuery内容。项目中使用了Bootstrap框架，在进行页面布局上，采用Bootstrap的网格布局，并努力实现了一部分功能的响应式设计。项目中应用的Bootstrap控件主要有：Dropdown, Button, Nav, NavBar, Breadcromb等，这些控件是的项目的主题更加一致鲜明。同时使用了：modal, collapse等JavaScript动态效果，提高了项目的用户体验。

与此同时，该项目中为了实现一些与数据库进行交互的任务，使用了Ajax进行异步的数据传输。通过Ajax的数据传输，用户页面在不进行页面跳转的情况下，也可以进行数据的刷新。

最后，为了在展示统计数据时，能够达到更佳的表现效果，页面的最后一个功能项中的图标使用的是jqPlot，通过色彩鲜明、错落有致的数据表，为用户提供更直观的数据分析。

# 测试设计：

厨师：

|  |  |
| --- | --- |
| 测试用例标识符：TC-01  功能测试：ShowQueue（）  Pass/Fail标准:当订单列表正确显示则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 订单列表正确显示 |
| Call Function (Fail) | 订单列表未正确显示 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-02  功能测试：finishCook（）  Pass/Fail标准:当订单列表某项状态未完成则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 订单列表厨师选中的订单显示完成 |
| Call Function (Fail) | 厨师选中后订单未显示完成 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-03  功能测试：Showtochef（）  Pass/Fail标准:当完成订单信息会传送到服务员处则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 服务员成功接收到订单完成消息 |
| Call Function (Fail) | 服务员未成功接收到订单完成消息 |

服务员：

|  |  |
| --- | --- |
| 测试用例标识符：TC-04  功能测试：viewTableStatus（）  Pass/Fail标准:成功显示桌子状态则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 成功显示桌子状态信息 |
| Call Function (Fail) | 未成功显示桌子状态信息 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-05  功能测试：changeStatus（）  Pass/Fail标准:点击桌子图标桌子状态会改变则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 点击图标后桌子状态改变 |
| Call Function (Fail) | 点击图标后桌子状态未改变 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-06  功能测试：seeTimeTable（）  Pass/Fail标准:点击查看时间表能显示预订时间表则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 预订时间表成功显示 |
| Call Function (Fail) | 预订时间表未成功显示 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-07  功能测试：addReservation（）  Pass/Fail标准:增加预订信息成功则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 增加预订信息成功 |
| Call Function (Fail) | 增加预订信息未成功 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-08  功能测试：deleteReservation（）  Pass/Fail标准: 删除预订信息成功则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 删除预订信息成功 |
| Call Function (Fail) | 删除预订信息未成功 |

清洁工：

|  |  |
| --- | --- |
| 测试用例标识符：TC-09  功能测试：viewTableStatus（）  Pass/Fail标准:成功显示桌子状态则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 成功显示桌子状态信息 |
| Call Function (Fail) | 未成功显示桌子状态信息 |

员工：

|  |  |
| --- | --- |
| 测试用例标识符：TC-10  功能测试：Login（）  Pass/Fail标准:员工能成功登陆则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 成功登录到界面 |
| Call Function (Fail) | 未成功登录到界面 |

管理者：

|  |  |
| --- | --- |
| 测试用例标识符：TC-11  功能测试：viewMenu（）  Pass/Fail标准:能成功显示菜单则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 成功显示菜单 |
| Call Function (Fail) | 未成功显示菜单 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-12  功能测试：addItem（）editItem（） deleteItem（）  Pass/Fail标准:管理者能成功增加、修改、删除菜单项并显示最终结果则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 管理者能成功增加、修改、删除菜单项并显示最终结果 |
| Call Function (Fail) | 增加、修改、删除菜单项失败 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-13  功能测试：addCategory（）editItem（）deleteItem（）  Pass/Fail标准: 管理者能成功增加、修改、删除菜品种类并显示最终结果则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 管理者能成功增加、修改、删除菜品种类并显示最终结果 |
| Call Function (Fail) | 增加、修改、删除菜品种类失败 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-14  功能测试：viewShifts（）editShifts（）  Pass/Fail标准:能成功显示员工调度表并能修改则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 能成功显示员工调度表并能修改则通过 |
| Call Function (Fail) | 员工调度表不能成功显示或不能修改调度 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-15  功能测试：viewBusinessStatus（）  Pass/Fail标准:餐厅财务状态图表能成功显示则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 餐厅财务状态图表能成功显示 |
| Call Function (Fail) | 餐厅财务状态图表未能成功显示 |

顾客：

|  |  |
| --- | --- |
| 测试用例标识符：TC-16  功能测试：confirmOrder（）  Pass/Fail标准: 顾客能根据需求成功点餐并进入自己的订单队列则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 顾客能根据需求成功点餐并进入自己的订单队列 |
| Call Function (Fail) | 顾客不能成功点餐 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-17  功能测试：seeTimeTable（）  Pass/Fail标准:顾客查看预定表时预定时间表能成功显示则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 预定时间表能成功显示 |
| Call Function (Fail) | 预定时间表未能成功显示 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-18  功能测试：editBill（）  Pass/Fail标准:顾客能成功进行账单付款则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 顾客能成功进行账单付款 |
| Call Function (Fail) | 顾客不能进行账单付款 |

|  |  |
| --- | --- |
| 测试用例标识符：TC-19  功能测试：addReservation（）deleteReservation（）  Pass/Fail标准:顾客通过操作能增加预订和删除预订并显示到预定表则通过 | |
| 测试过程 | 预期结果 |
| Call Function (Pass) | 顾客通过操作能增加预订和删除预订并显示到预定表 |
| Call Function (Fail) | 顾客不能增加预订和删除预订 |

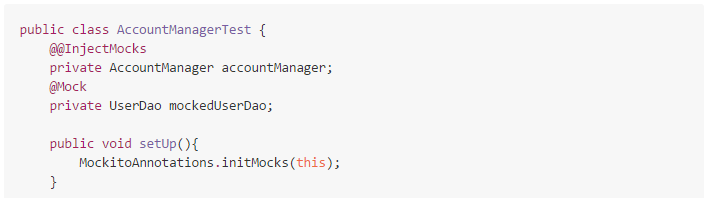
网络端单元测试

1. ORM映射与数据库表结构映射以及Dao接口测试：

@ContextConfiguration(locations = { "/applicationContext-test.xml" })将applicationContext导入到当前的测试类中。

继承SpringTransactionalTestCase类，对事务进行处理。

1. Service类测试：创建Mock对象



使用mock生成dao对象，并将dao对象注入到service对象中。最后使用Mockito的verify()函数进行运行时检测。

1. 使用Selenium2对Web页面的跳转逻辑和Ajax进行测试。

# 项目管理和历史工作：

项目管理：

对于小组成员来说，管理在开发过程中起着至关重要的作用，只有优秀的管理和正确的团队领导，开发才能顺利进行。由于每个人工作时间的不同，需要对其进行协调和管理。

该组成员的代码和文档都是通过GitHub进行统一管理的，有助于版本控制以及开发意见的收集。能够保证每个成员的代码文档与其他成员保持一致，方便软件开发。

历史工作：

# 工作计划（甘特图）：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 任务名 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Report2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 交互图 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 类图 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 追溯矩阵 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ER图 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OCL |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 架构设计 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 子系统 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 算法 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 用户界面设计 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 网络端实现 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 测试设计 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 项目管理 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 工作计划 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Demo1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 实现 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 演示 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |