****

**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：

杨帆：

吴晨：

张天霖：

目录

[交互图： 4](#_Toc448594237)

[类图和接口规范： 11](#_Toc448594238)

[类图： 11](#_Toc448594239)

[类描述： 11](#_Toc448594240)

[数据类型和操作符： 11](#_Toc448594241)

[追溯矩阵： 11](#_Toc448594242)

[设计模式： 11](#_Toc448594243)

[OCL（对象约束语言契约）： 11](#_Toc448594244)

[系统架构和系统设计： 11](#_Toc448594245)

[架构设计： 11](#_Toc448594246)

[确定子系统： 11](#_Toc448594247)

[映射子系统到硬件： 11](#_Toc448594248)

[持久数据存储： 11](#_Toc448594249)

[全局控制流： 11](#_Toc448594251)

[硬件需求： 11](#_Toc448594252)

[算法和数据结构： 12](#_Toc448594253)

[用户界面与实现： 12](#_Toc448594254)

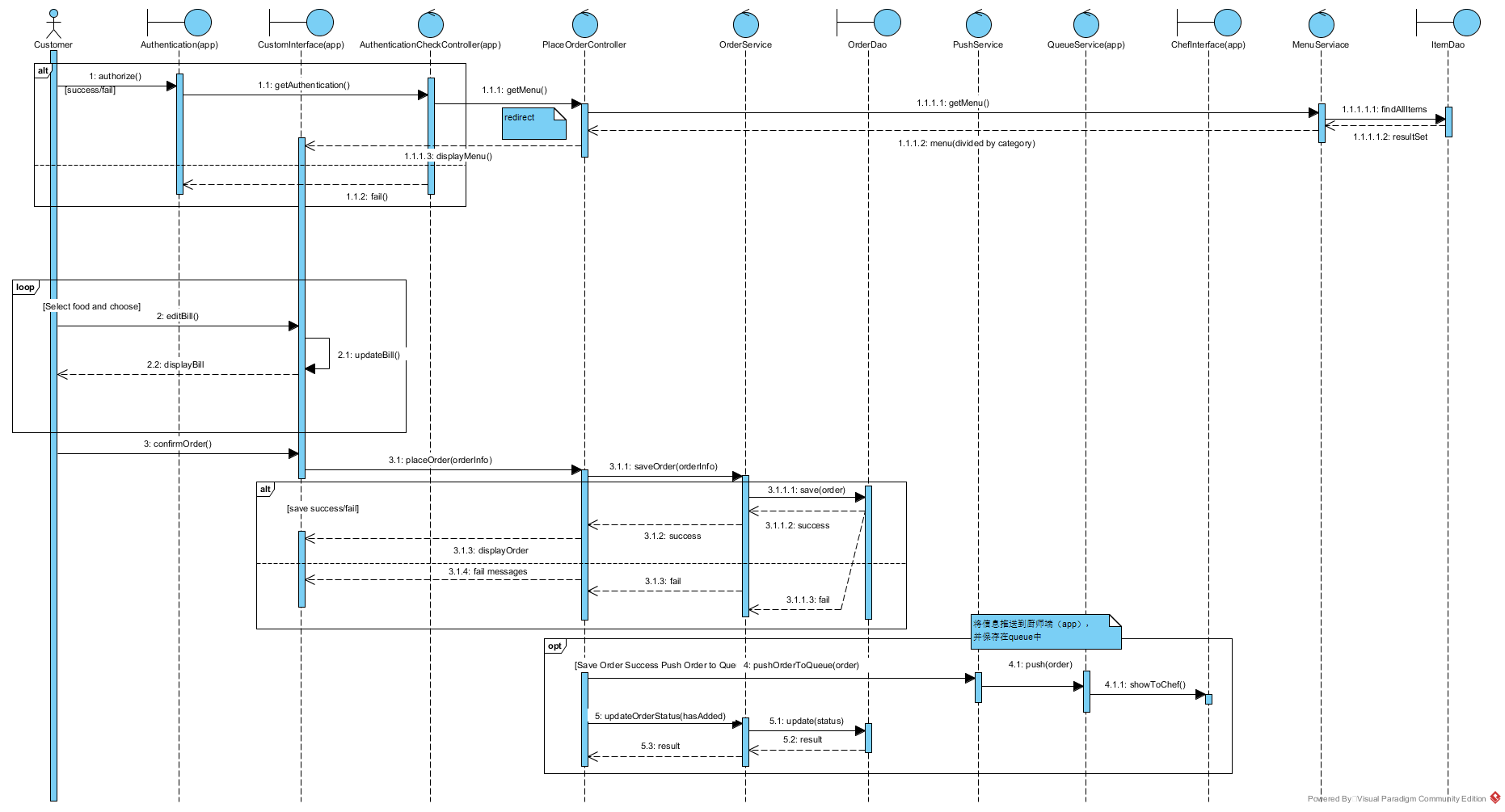
[员工网络端实现： 12](#_Toc448594255)

[测试设计： 12](#_Toc448594256)

[项目管理和历史工作： 12](#_Toc448594257)

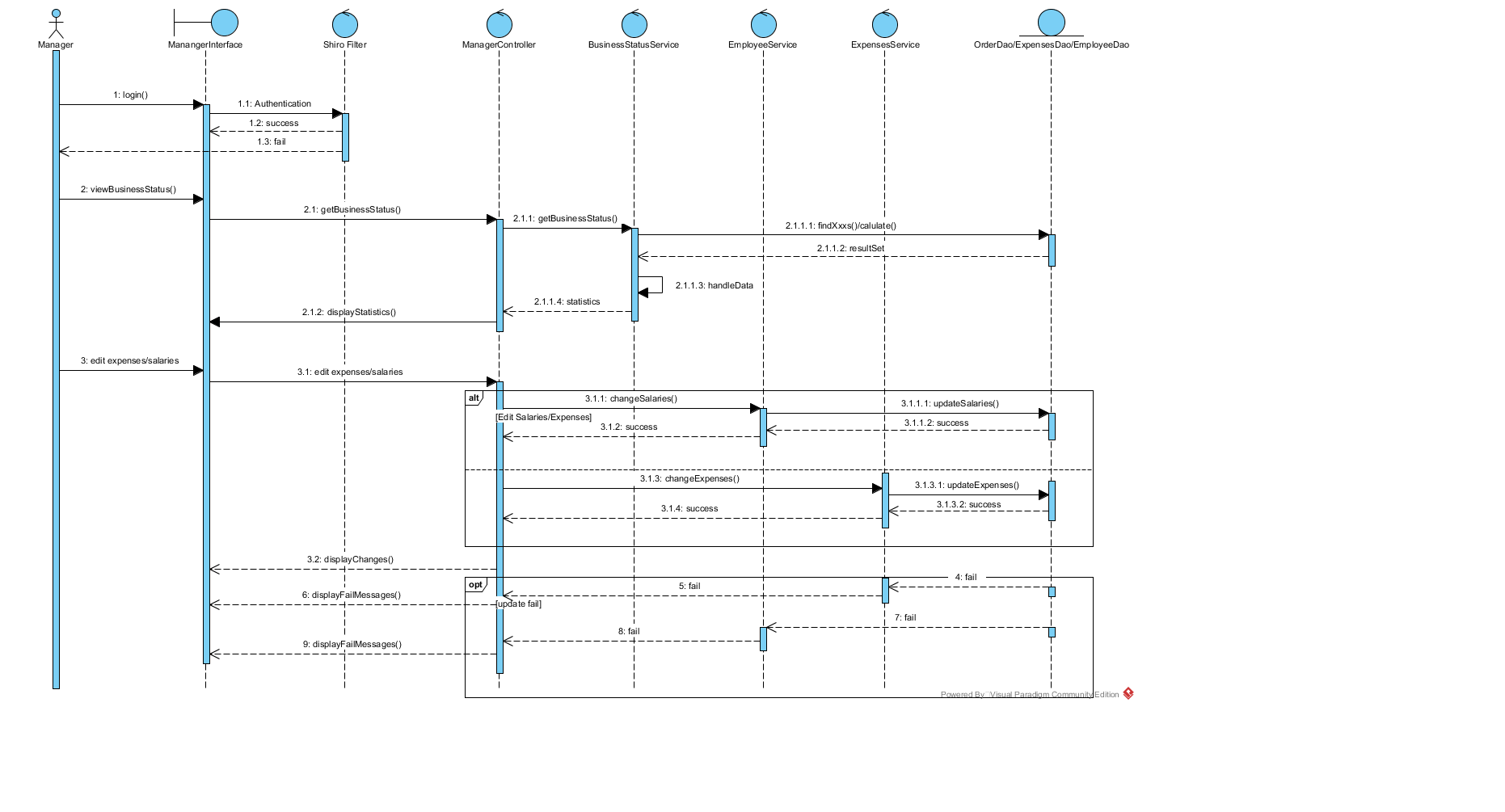
# **交互图**：

用例1 顾客订餐



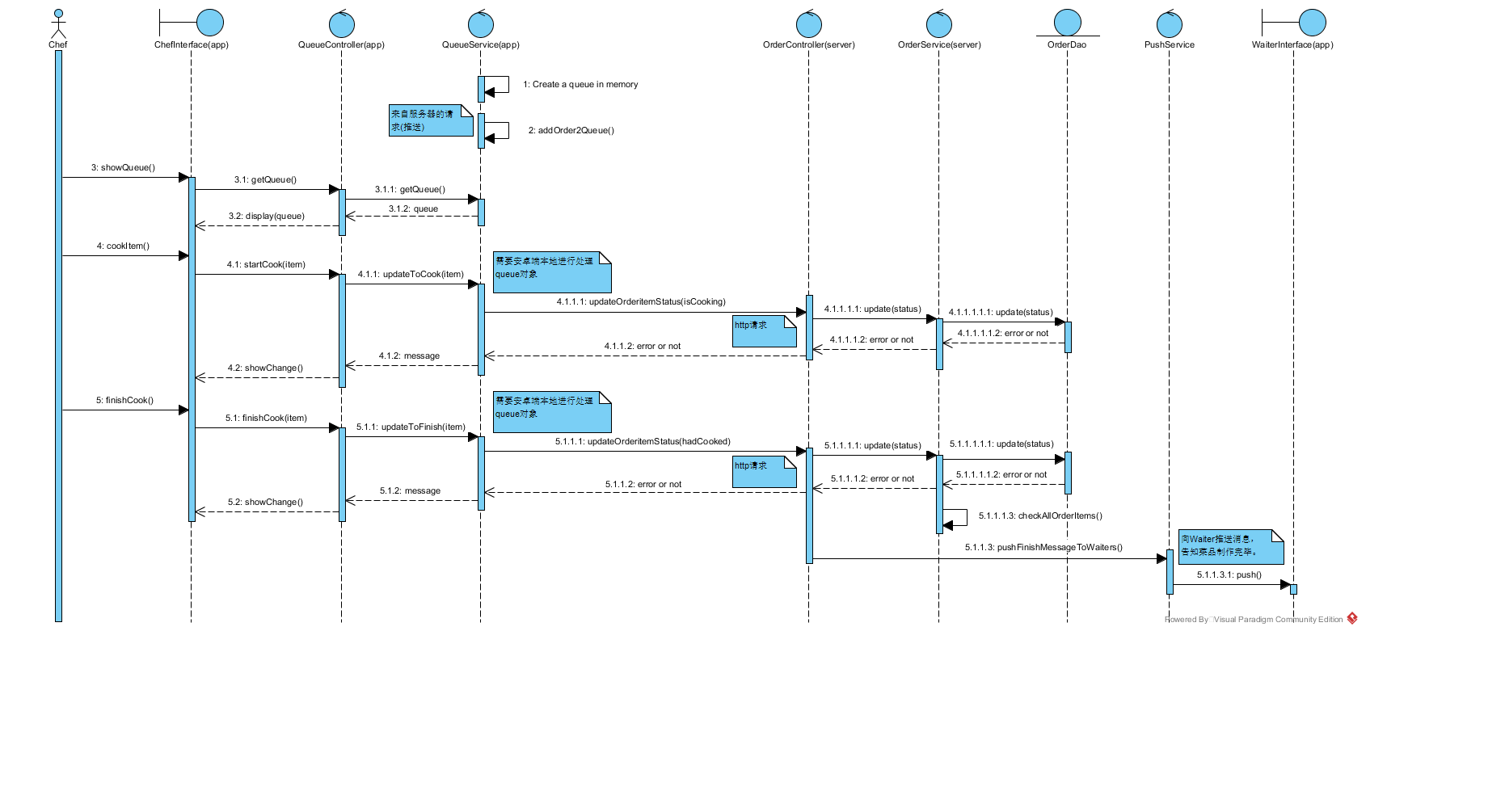
顾客可以根据顾客的用户界面进行点餐，并且显示订单。只要自己的订单没有确定，顾客就可以进行编辑。订单确定后，会将信息推送到厨师端，并保存在queue中。

用例2 查看业务状态



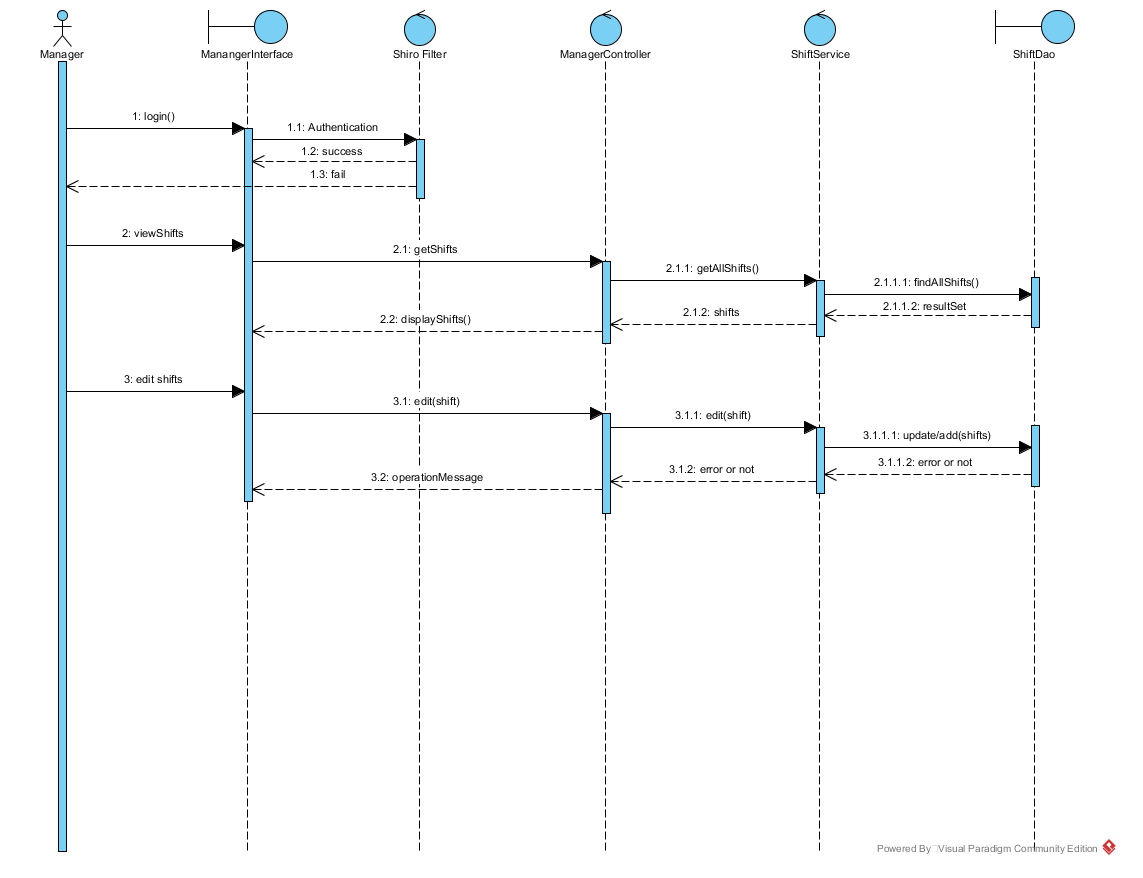
管理者可以登录管理者的用户界面查看业务状态。经理可以查看或编辑员工的工资，同时还可以查看餐厅的年（月）的利润与亏损。能有效地和数据库进行相互通信，进行存取。

用例3 管理订餐队列



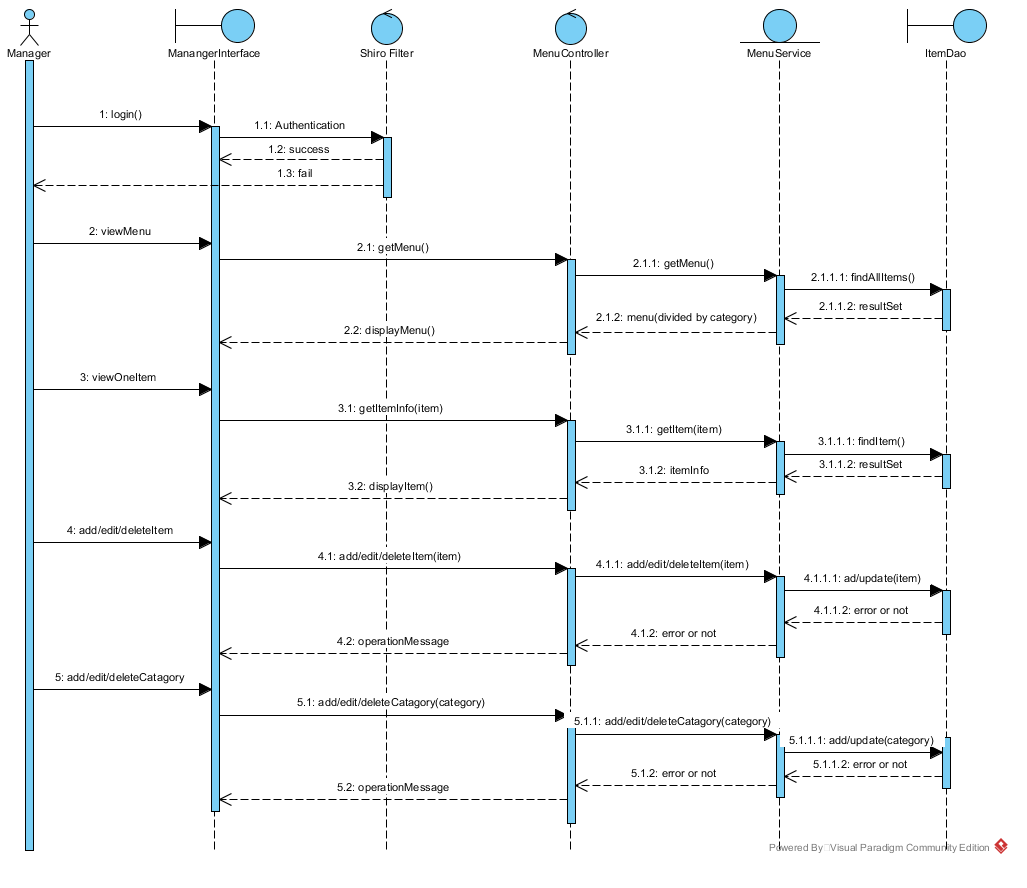
厨师可以查看由服务生或客户添加到队列的订单。厨师可以访问他的界面查看队列。厨师还可以与界面进行交互。厨师完成菜品制作后，他能进入完成的订单，从而消除订单未完成状态。这也能引发与服务员的通信，通知服务员取菜。

用例4 管理工作调度



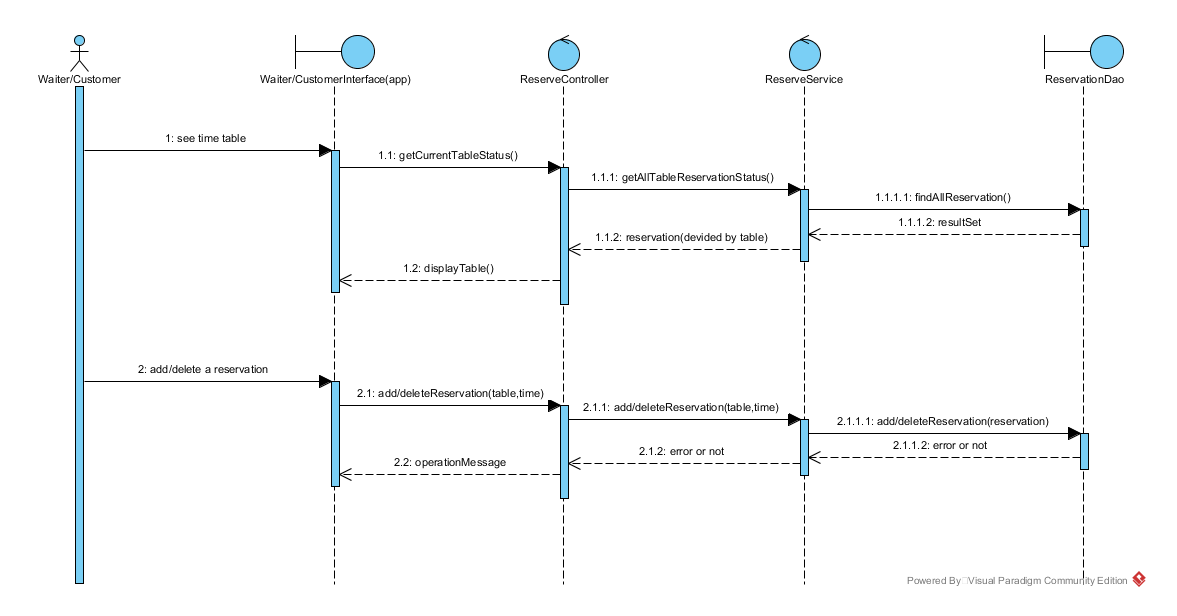
管理者可以进入管理界面对员工工作调度表进行查询和修改，对员工进行工作调度。

用例5 管理菜单



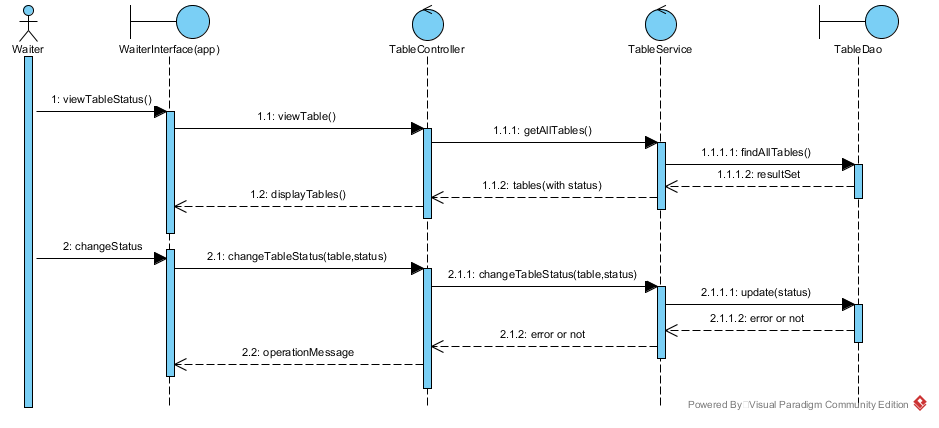
管理者可以进入管理者界面来管理菜单。管理者有能力添加或删除一个项目来更新的菜单。

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。

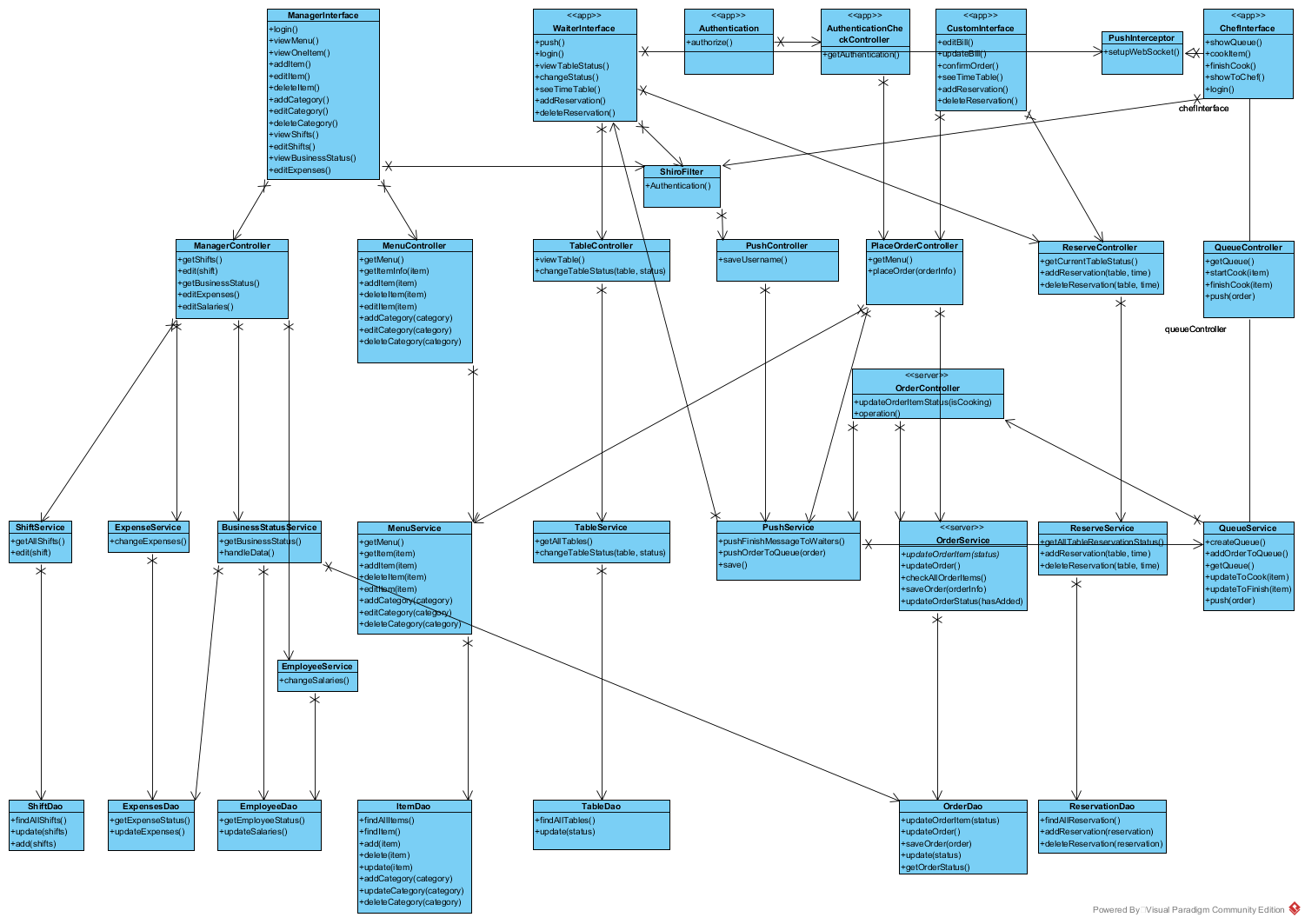
用例7 管理桌子



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 |

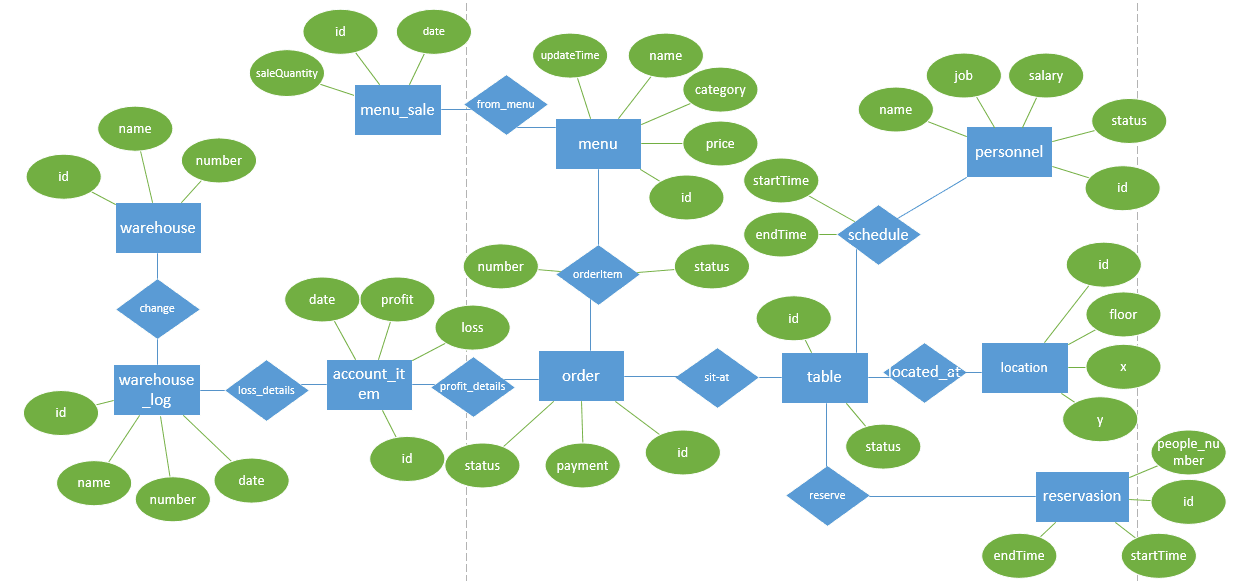
ChefInterface：

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：

## 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：

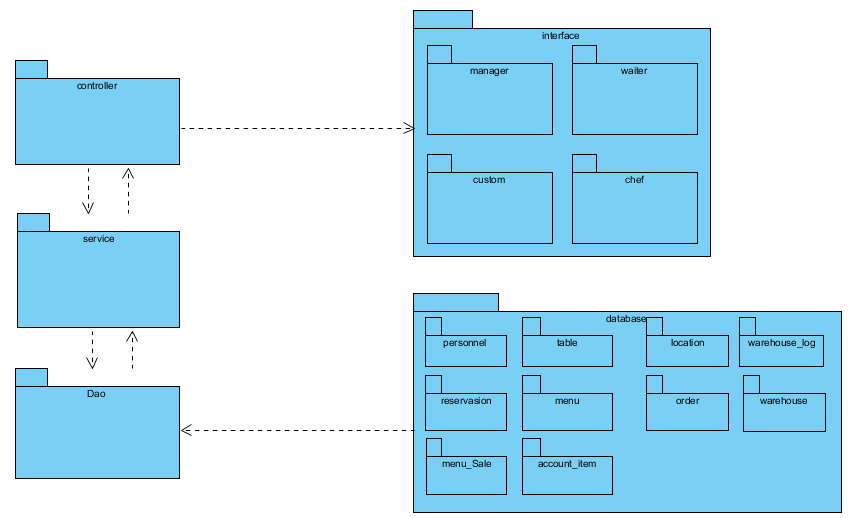
我们设计的架构是基于C/S（客户端-服务器）模型的，客户端直接连接集成服务器。这种系统的工作原理是建立统一的基础设施，所有通信都通过服务器与客户端进行连接与请求。

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：

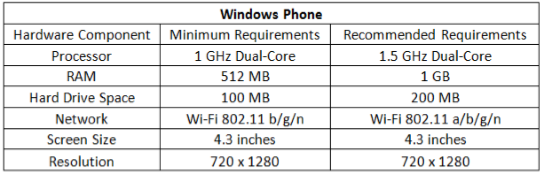
这是一个过程驱动的软件。几乎都以线性的方式来执行。顾客进入餐厅会利用程序选择位置。然后顾客会通过平板设备或服务员通过平板设备进行点菜。订单点菜完毕后会将订单传递给厨师，当厨师准备好时，将通知服务员将菜品直接到客户。当顾客吃完他会通过平板付账，然后服务员会标志桌子为脏状态。这清楚地表明，所有的行动都是一种线性的方式，取决于先前的操作。

并发：

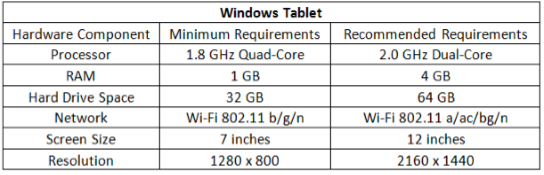
系统将包含多个线程，这将涉及多个程序同时独立运行。多个客户将在同一时间下订单，这就是为什么我们需要并发。该解决方案是将多个线程运行到队列中。另一个原因是管理者编辑库存或费用时，一个线程将处理数据库中的更新。不需要同步，因为每一个线程都是独立的。

### 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，其中包括优先级队列、数组和哈希表。这些将使用不同的员工工作和利用的应用程序。当服务生在参加客户的工作时，特别是这是指如何存储表。方进来，让坐在一张桌子旁，而表添加到服务员的队列结构使FCFS（先来先服务）在餐厅点菜。

另一方面，厨师使用优先级队列来管理所有的订单。这可以归因于的想法，虽然表正在办理中FCFS，食物可能会略有不同，以熟。考虑到某些菜比别人花的时间更长，优先队列允许厨师在另一个菜是在稍后的时间里被一个宴会点的一个宴会来开始烹饪一道菜。

菜单项将被存储在数组中。从性能的角度看各种结构，阵列清楚地为我们的目的。该菜单是非常重要的性能，由于其在几个不同的情况下使用。这是一个快速的例子，是客户浏览菜单和管理者，使用它进行统计分析，并修改菜单的基础上。这些操作都是从高性能集合中受益，即数组。

在实际使用中，哈希表通常是首选的快速数据查找，由于它们的映射性质。其结果是，在系统中，他们将被实现在员工门户网站，以地图的每个员工的细节（不限于只支付和调度信息）。在这样的情况下，哈希表中的每一个键都将代表一个员工，而值对应于通过管理者门户访问的相关信息。

### 用户界面与实现：

### 员工网络端实现：

### 测试设计：

### 项目管理和历史工作：