10/28/2016

V-Menu Restaurant Management System

Building the Restaurant of Tomorrow Today

School: University of North Florida

Course: Software Engineering (CEN 4010)

Instructor: Dr. Douglas Leas

Team #2

Lead Developer: Matthew Boyette

Lead Designer: David Hughes

Developer: William Mejia

Developer: Demetrius Myers

Developer: Slaven Popadic

Table of Contents

Section 1: Overview

V-Menu is a system designed to turn the day-to-day logistics of managing a restaurant into a problem solvable using distributed computing. By turning the individual human elements of the restaurant into computable problems, we hope to push the boundaries and take the next step toward a completely automated restaurant. Although a fully automated restaurant is still relatively infeasible with today's level of off-the-shelf consumer technology, it won't be long before advances in robotics make such an occurrence not only practical, but also commonplace. There are three primary factors of functionality that must be addressed before robotic employees become ready for mainstream adoption.

- Precision of movement. Tasks like cooking and waiting tables require precise movements, and the objects being moved are designed to be manipulated by a human hand.
- Collision avoidance. A restaurant is usually a bustling place with a great deal of movement (not only by employees, but also customers). Being able to avoid colliding with obstacles is critical.
- Communication interface. Computerized speech recognition still has some ways to go before a randomly chosen customer can reliably communicate their orders verbally to a robotic employee.

Many restaurants currently have similar systems in place already, but the feature-set and overall design is inconsistent. Further, many of these systems focus solely on patrons who want food delivered to an exterior location, or patrons who wish to pick up their food at the restaurant and then take it with them to dine elsewhere (usually a home or office).

Our goal is two-fold: to serve the needs of those dining inside the restaurant, and to merge the functionality of existing systems for a consistent design and interface which can be easily customized or extended to suit each individual client.

Section 2: Project Team

- Matthew is the team leader and lead developer. The parts of this deliverable that he contributed are the title page, table of contents, and the overview (this page). He also created most of the diagrams and images presented in this document.
- David was the lead designer. Unfortunately, David had to withdraw from the class for personal reasons; however, we have included him here in recognition of the work that he did prior to his withdrawal. The parts of this deliverable that he contributed are some of the GUI mockups and assisted with creating some of our design diagrams.
- William is a developer. The parts of this deliverable that he contributed are assisting with creating diagrams and integration of the OpenMenu database schema with our Drupal database. He also wrote most of the text used throughout the rest of this document.
- Demetrius is a developer. The parts of this deliverable that he contributed are presenting our design to the rest of the class and the PowerPoint slides for that presentation.
- Slaven was a developer, but has recently taken on the mantle of lead designer due to the loss of David. The parts of this deliverable that he contributed are assisting Demetrius with the design presentation, and contributing GUI mockups.

Additionally, every member of the team participated in generating ideas and discussing implementation related topics during both our regular team meetings and off-hours using our Slack channel.

Section 3: Design Rationale

There are several critical issues and tradeoffs that we must address due to some of our design decisions. They are listed and described below based on with which facet of our design each is associated.

Section 3A: Client-Server - Critical Issues and Tradeoffs

- User devices will need wireless internet access in some form to reach the server; either Wi-Fi or a data plan. Many low-end consumer tablets do not have SIM card slots and are therefore Wi-Fi only. The restaurant owner cannot assume that all customers will have their own private data plans through a mobile carrier. Thus, they may wish to provide free Wi-Fi access to their customers to mitigate this risk. This is a responsibility our project does not address; providing this service is left to the restaurant owner.
- Performance can be a serious issue if the server is under heavy load. Likewise, there is a single-point of failure; if the server goes down, our application is useless. Restaurant owners using our product may wish to invest in a cloud-computing platform (such as Amazon's AWS or Microsoft's Azure, just as examples) to spin up multiple server instances for load balancing and redundancy to mitigate this risk.
- All non-trivial operations must be performed server-side; performing operations on the client-side represents a security risk. Fortunately, in our case the clients are nothing more than internet browsers; their only responsibility is to render content. So, this risk is mitigated by our project through other facets of our overall design, namely that we are building a web application (see below for more details).
- The restaurant owner may wish to invest in at least one dedicated IT professional to perform regular basic maintenance duties on the server. Many of the underlying systems and services powering our application receive regular security updates, from the operating system (Linux) all the way to the content management system (Drupal). This is a responsibility our project does not address; adhering to best practices concerning server maintenance and security is left to the restaurant owner.

Section 3B: Web Application - Critical Issues and Tradeoffs

- The biggest drawback of this facet of our design is web browser compatibility. In theory, each web browser should render our content the same way. Unfortunately, in practice this is rarely the case. We have selected a subset of the most popular browsers, and will be testing our application with their latest stable versions. You can get more details below in the *Assumptions and Risks* section of this document.
- Different countries have different laws governing the practices of web applications. For example, in the United States, the *Americans with Disabilities Act* (ADA) mandates accessibility requirements to accommodate people with disabilities. Additionally, the European Union mandates not only access requirements, but also data privacy requirements. It is impossible for our team to completely meet these requirements in the time we have available. Instead, we have considered usability and accessibility extensively throughout our design process, and complying with any remaining specific local laws or regulations is left to the restaurant owner; we make no guarantees that we comply with every local law for any individual restaurant.

Section 4: High-Level Design

Section 4A: High-Level Design - The Short(er) Version

The high-level design of the V-Menu system is straightforward. We use a standard client-server configuration. The server runs the software that enables the application to function properly, and the clients interact with that server. Clients send requests to the server, and the server responds to those requests. There is nothing particularly special about the underlying hardware used by either the client or the server. Any standard off-the-shelf computing hardware will suffice for both roles. In practice, the server hardware should be tailored for the storage of large data sets and fast access to those data sets. However, going into detail about best practices concerning server hardware is beyond the scope of this document.

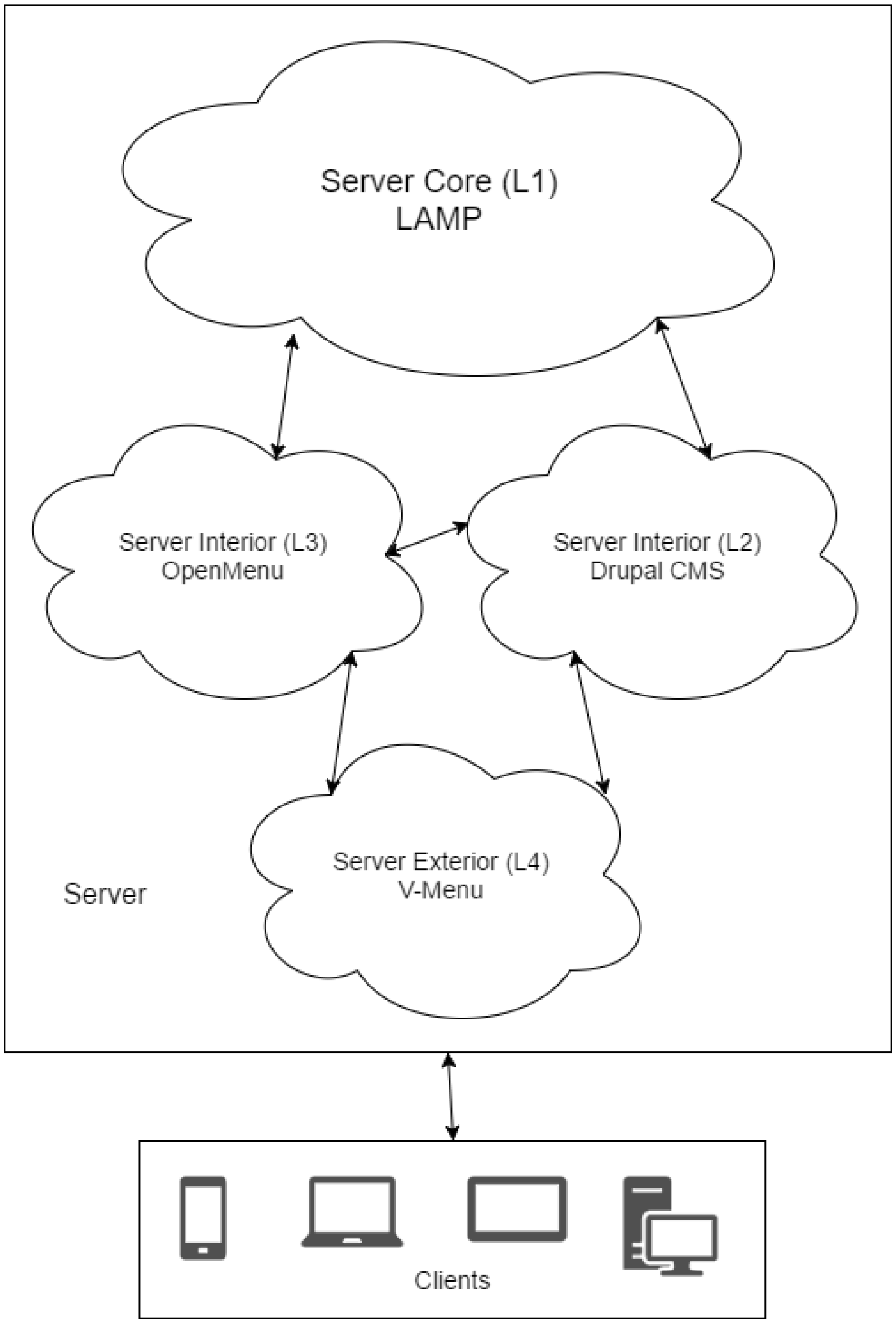
We aim to provide an application that can service any web-capable computing device, but our primary form-factor targets are mobile devices. Specifically, we are most interested making sure that using the application on smartphones and tablets is simple and efficient. We envision a future where instead of using pens and pads of paper, wait staff carry around relatively inexpensive thin tablets. These devices do not need high performance hardware specifications; instead they should each be optimized for accessing web content and lasting the entire work day on a single full charge of the battery.

Each order is a collection of menu items. Whenever an order is placed, that order gets stored in our database. This process is elaborated on in much more detail further in the document. The clear majority of the application revolves around displaying the menu to the customer, allowing him or her to choose the desired menu items, and then storing the order in our database. Aside from kitchen staff flagging an order as completed (ready for delivery to the customer) and wait staff flagging an order as delivered, the orders are not manipulated in any way once they are placed and confirmed. This constraint will be enforced in our implementation; at no point in time will any user be directly manipulating the database. Our application will do that to maintain the integrity of the data.

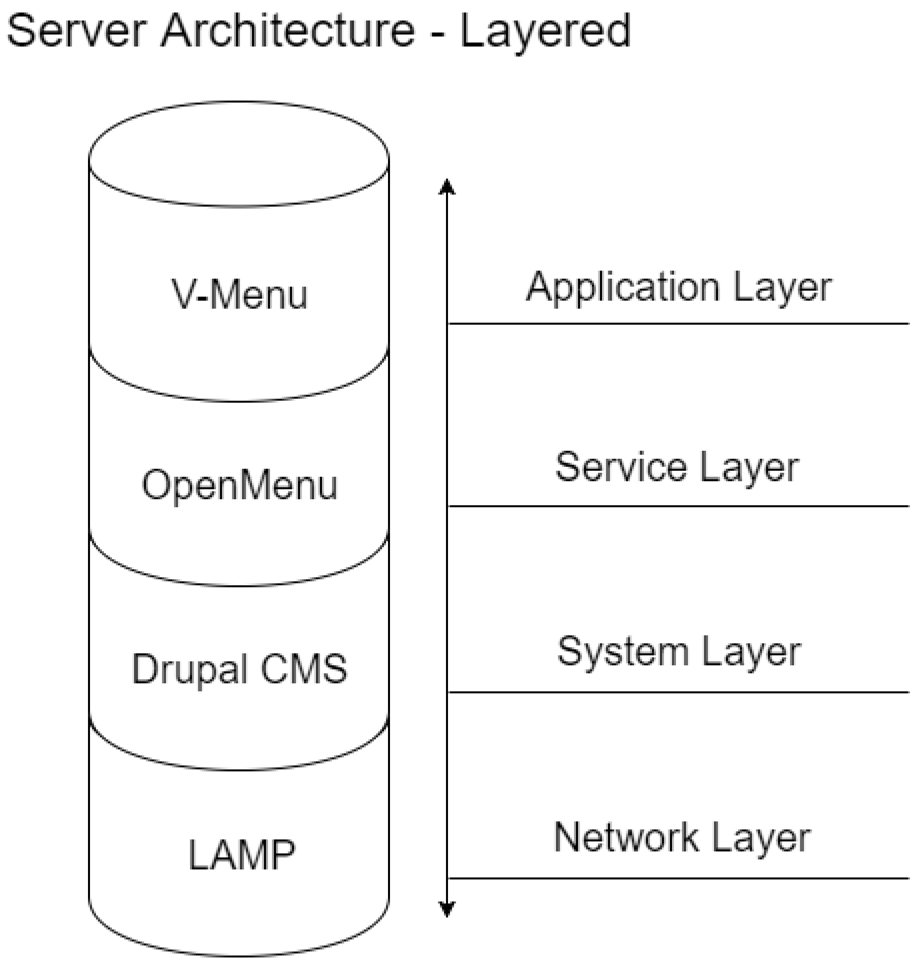
This project uses a variety of architectural styles and design philosophies. Instead of adhering to a single rigid and inflexible approach, we have strived in each component of the overall system to apply the best technique to solve the applicable problem. The rest of this document will go into significant detail behind each paradigm we have used, and provide explanations for why those choices were made.

Section 4B: Architectural Style Breakdown

Our application uses a web-based client-server architecture using a centralized standard relational SQL database. For the web-based aspect of the project we are using PHP as our primary server-side programming language, which renders standard HTML documents using CSS for styling. We use a modified version of AJAX that combines JavaScript with JSON and YAML (Yet Another Markup Language). Our content management system, Drupal, provides an API framework that abstracts database operations and simplifies writing code in a consistent manner. We are also employing the OpenMenu specification for our database, so that restaurant owners will be able to create, share, and distribute menus in an open format. This improves interoperability, and allows an owner who owns multiple restaurants to use our system in all his or her establishments, even if they serve different types of food.

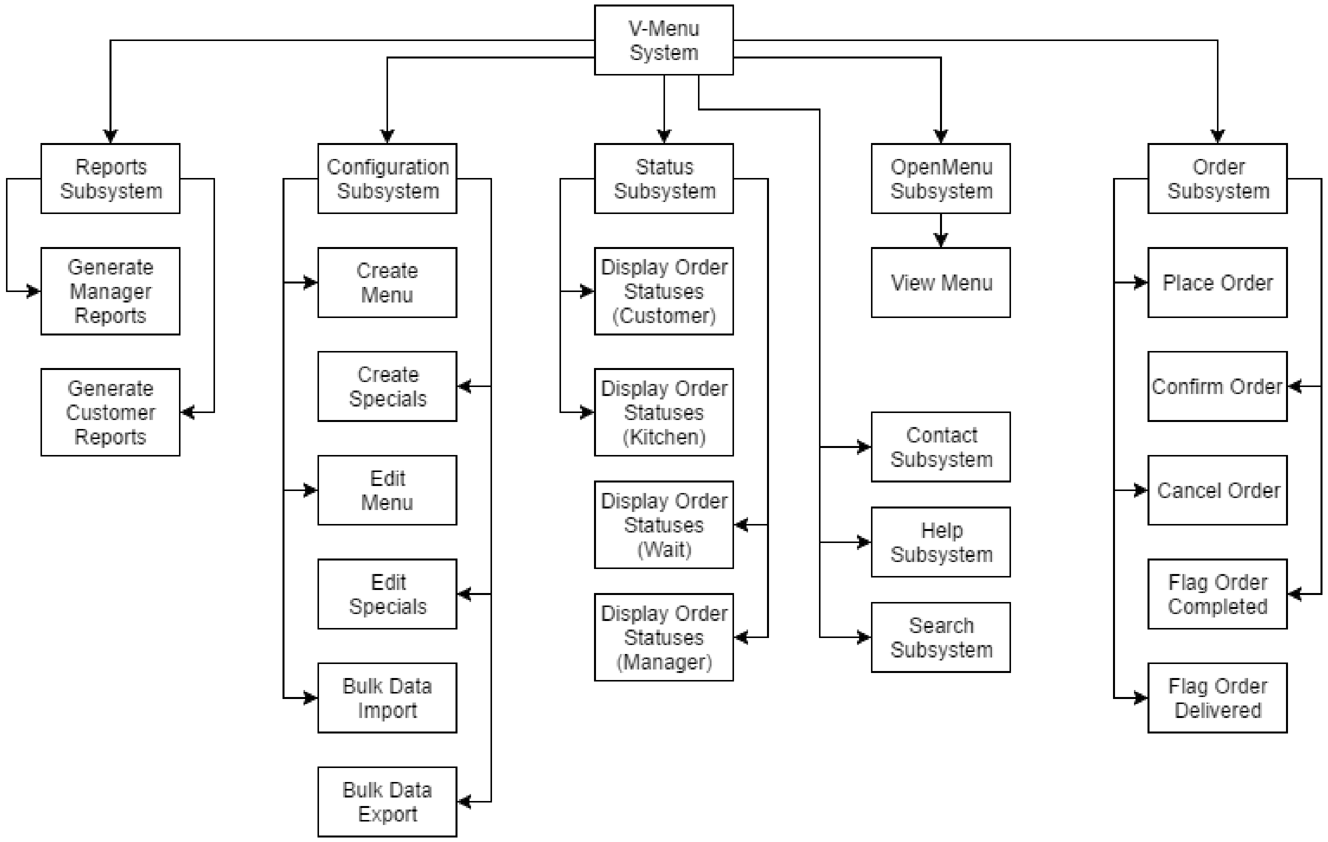


Our server architecture takes a layered stack approach that puts the application layer, that is the layer that directly faces the clients and consists of our application code, at the top of the software stack. Below the application layer is the service layer. This is the layer that represents our implementation of the OpenMenu specification and our data storage schema. Below the service layer is the system layer. This is the layer that represents the content management system we are using (Drupal). Below the system layer is the network layer. This is the layer that represents the LAMP (Linux, Apache, MySQL, and PHP) stack. The network layer handles standard web protocols (HTTP/HTTPS), contains the PHP interpreter, contains the relational database engine, and handles any necessary interaction with the physical hardware of the computing device that is being used as the server. Data can be passed up or down the chain from layer to layer. There are three main reasons that each of these technologies were chosen: they are open source (or have open source equivalents, such as MariaDB for MySQL), well documented, and being actively maintained by their respective development teams.



Section 4C: Decomposition Style Breakdown

Our team chose to perform functional decomposition for our application. We chose this decomposition style because it most accurately mirrored our high-level use case diagram from the requirements deliverable, and we are committed to remaining as close to our original requirements specification as possible. This is to prevent feature creep by keeping development focused on our core features which are most important. The functional decomposition diagram is shown on the next page.



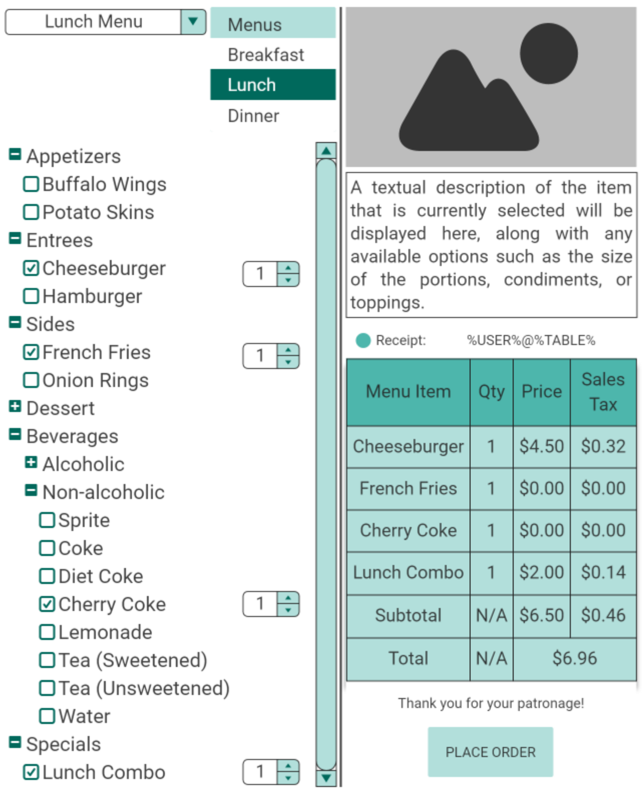
The V-Menu system consists of eight self-contained subsystem modules. These modules are described in detail in the following list:

1. Reports
    - This subsystem is responsible for generating and displaying reports for customers and managers. Customers will be able to see reports about which menu items are most popular. Managers will see financial reports, employee performance reports, and usage reports for the V-Menu service itself (how many users ordered using V-Menu over a given period).
2. Configuration
    - This subsystem is responsible for allowing the owner or a trusted manager to configure the V-Menu service. This includes creating menus, editing menus, creating special, editing specials, bulk data import, and bulk data export. Data import/export will use open source and well documented data formats.
3. Status
    - This subsystem is responsible for displaying the statuses of orders to users based on role. Anonymous customers must provide the unique order identification number to look up their order (since they are anonymous, their orders will not be associated with their user account, because they don't have one). Customers will be able to see the status of all their orders both past and present. Kitchen staff will be able to see all orders that have not yet been completed. Wait staff will be able to see all orders that have been completed, but not delivered for their respective tables. Managers will see all orders that have not been delivered, and will be able to look up delivered orders by their unique order identification number.

4. OpenMenu
    - This subsystem is responsible for pulling the menu information from the database and displaying it in a nice accessible format for the customer (or a member of the wait staff) to select individual menu items. A prototype of the GUI element responsible for this function is provided in the next section. The database elements supporting this function are covered in more detail in the *Detailed Design* section.
5. Order
    - This subsystem is responsible for all operations concerning orders. This includes placing orders, confirming orders, canceling orders, flagging orders as completed, and flagging orders as delivered. A state transition diagram is provided later in the document (in the *Detailed Design* section) which describes in more extensive detail the design of this subsystem.
6. Contact
    - This simple subsystem has only one function: allow the customer to directly contact the owner of the restaurant. There will be a form option, as well as displaying the restaurant's phone number and mailing address. The form will allow the customer to contact the owner by email. This can be done anonymously, or in a uniquely identifiable manner depending on the customer's preference. If the customer wants a reply, they must be uniquely identifiable. If the customer wishes to remain anonymous, this will be accomplished by having the V-Menu system send the email with its support email address in the ***From*** field of the email.
7. Help
    - This simple subsystem has only one function: allow the customer to request help from the wait staff. A state transition diagram is provided later in the document (in the *Detailed Design* section) which describes in more extensive detail the design of this subsystem.
8. Search
    - This simple subsystem has only one function: search for other restaurants (based on ZIP code) in that use the V-Menu system.

Section 4D: Initial GUI Prototypes

Section 4D-1: View Menu Prototype



- Appetizers
  - ☐ Buffalo Wings
  - ☐ Potato Skins
- Entrees
  - ☑ Cheeseburger 1
  - ☐ Hamburger
- Sides
  - ☑ French Fries 1
  - ☐ Onion Rings
- Dessert
- Beverages
  - Alcoholic
  - Non-alcoholic
    - ☐ Sprite
    - ☐ Coke
    - ☐ Diet Coke
    - ☑ Cherry Coke 1
    - ☐ Lemonade
    - ☐ Tea (Sweetened)
    - ☐ Tea (Unsweetened)
    - ☐ Water
- Specials
  - ☑ Lunch Combo 1

A textual description of the item that is currently selected will be displayed here, along with any available options such as the size of the portions, condiments, or toppings.

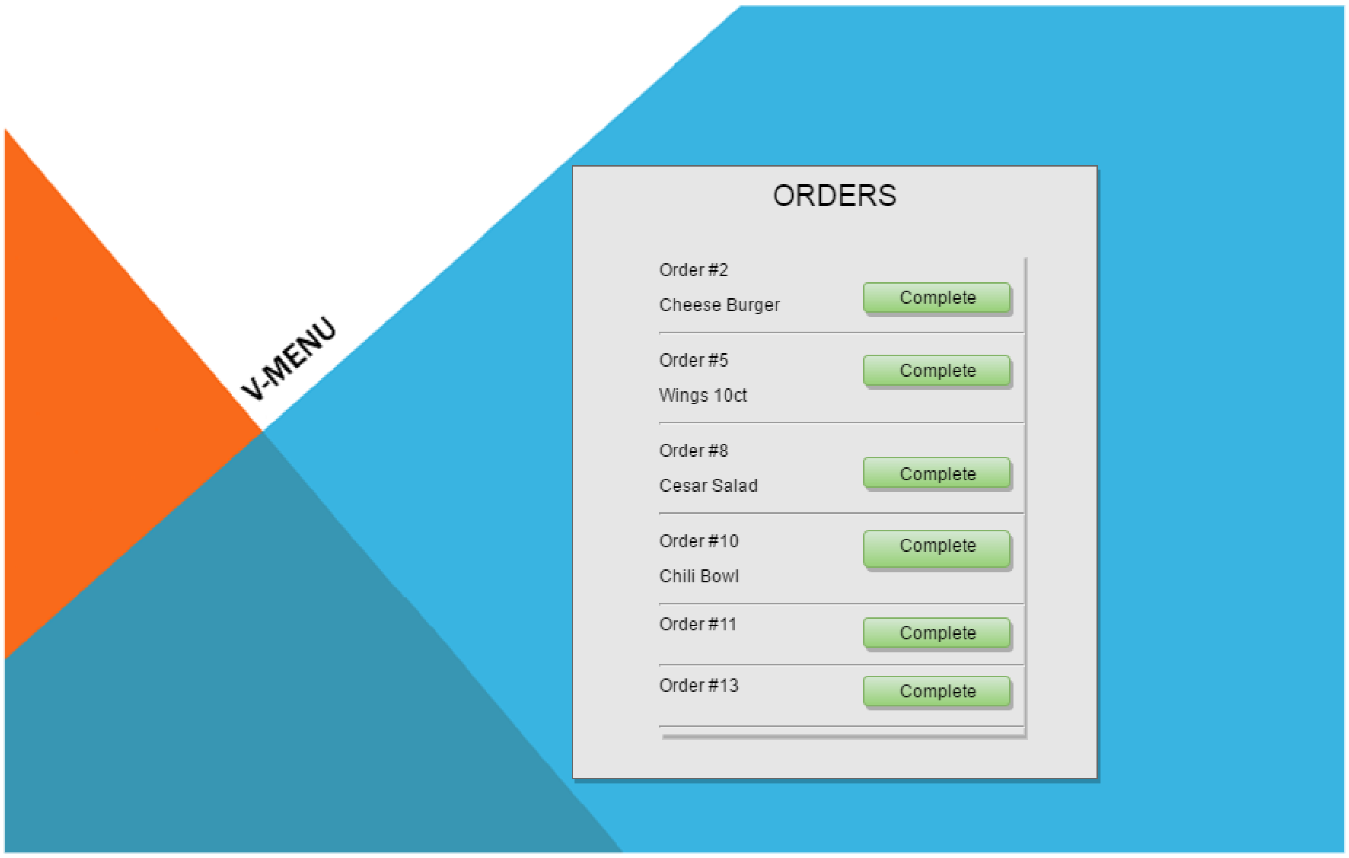
Receipt: %USER%@%TABLE%

Menu Item	Qty	Price	Sales Tax
Cheeseburger	1	$4.50	$0.32
French Fries	1	$0.00	$0.00
Cherry Coke	1	$0.00	$0.00
Lunch Combo	1	$2.00	$0.14
Subtotal	N/A	$6.50	$0.46
Total	N/A	$6.96	

Thank you for your patronage!

This prototype shows a potential design of the GUI for viewing the menu. Users can click or tap on menu items without necessarily checking their respective boxes. Doing so will cause a stock photo of the item and a description to load. Checking the box for a menu item will cause a quantity field to appear beside it. The screen should also include a preview of the customer's receipt.

Section 4D-2: Kitchen Staff View Prototype



Order #2
Cheese Burger

Order #5
Wings 10ct

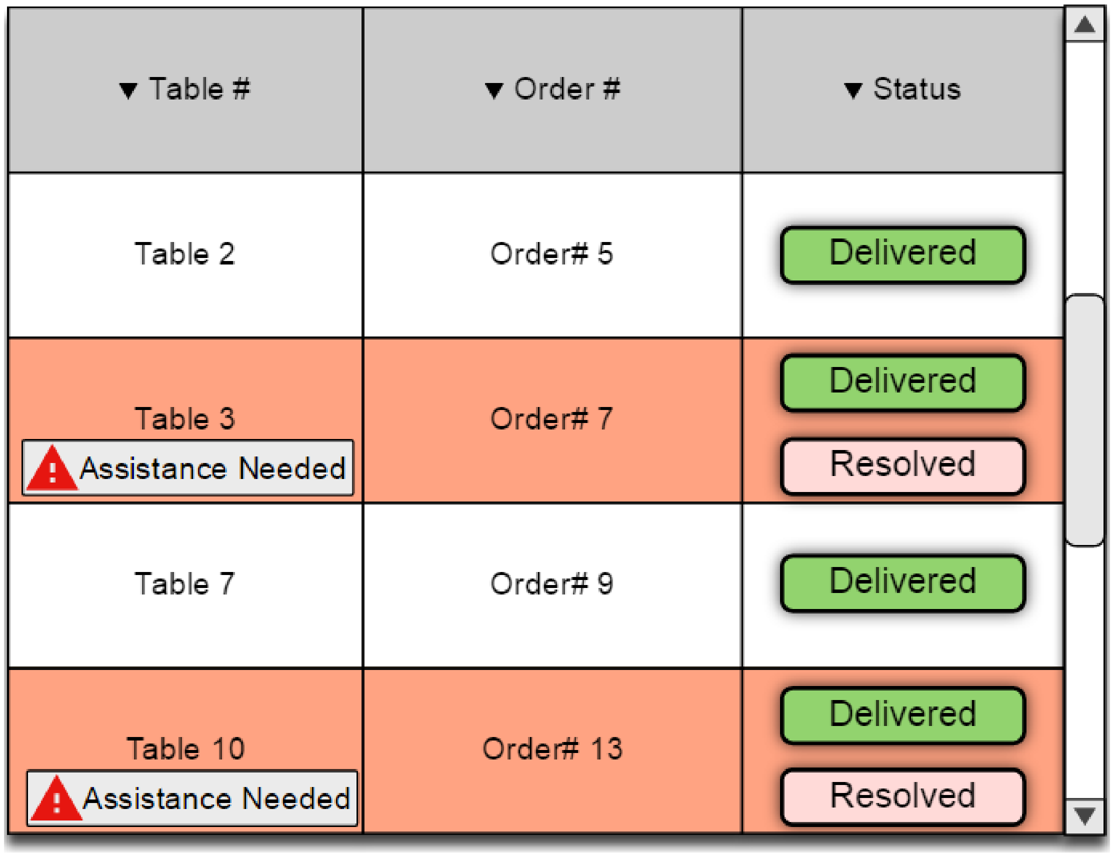
Order #8
Cesar Salad

Order #10
Chili Bowl

Order #11

Order #13

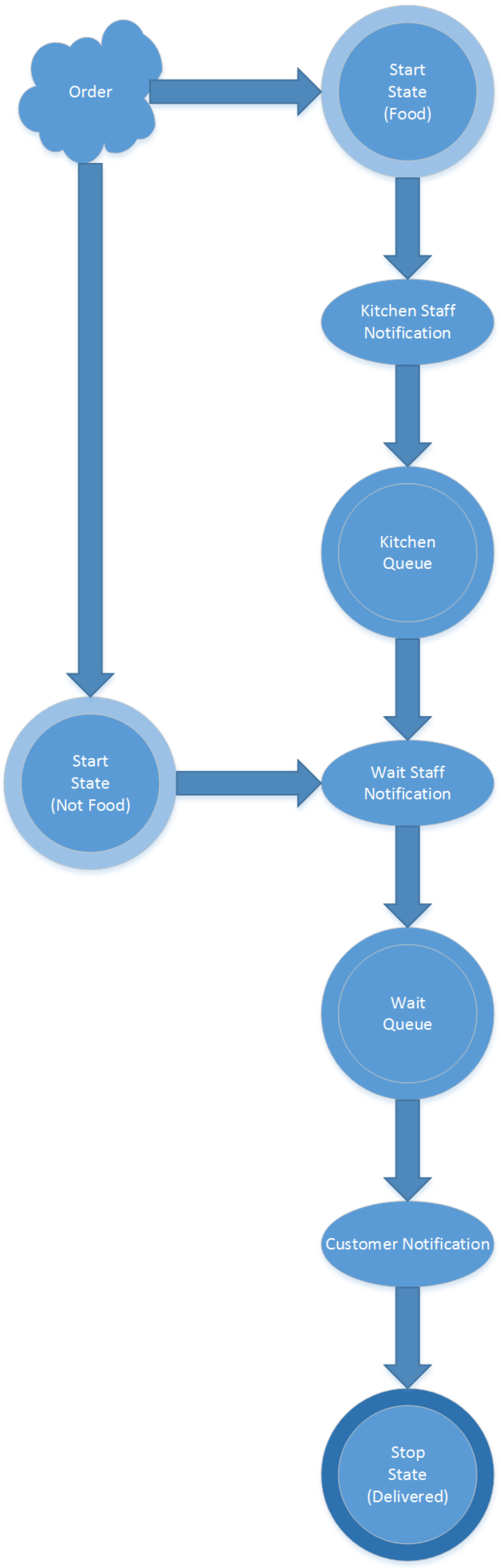
Section 4D-3: Wait Staff View Prototype



▼ Table #	▼ Order #	▼ Status
Table 2	Order# 5	Delivered
Table 3 ⚠ Assistance Needed	Order# 7	Delivered Resolved
Table 7	Order# 9	Delivered
Table 10 ⚠ Assistance Needed	Order# 13	Delivered Resolved

Section 5: Detailed Design

Section 5A: Ordering Process



Ordering Process Unidirectional

An order is a collection of menu items and metadata which is created by either a Customer or a member of the Wait Staff.

Menu items are categorized into food and non-food. Non-food items include replacement dishware, utensils, napkins, beverages, etcetera. Each menu item has an individual cost associated, which is represented by a non-negative floating point number.

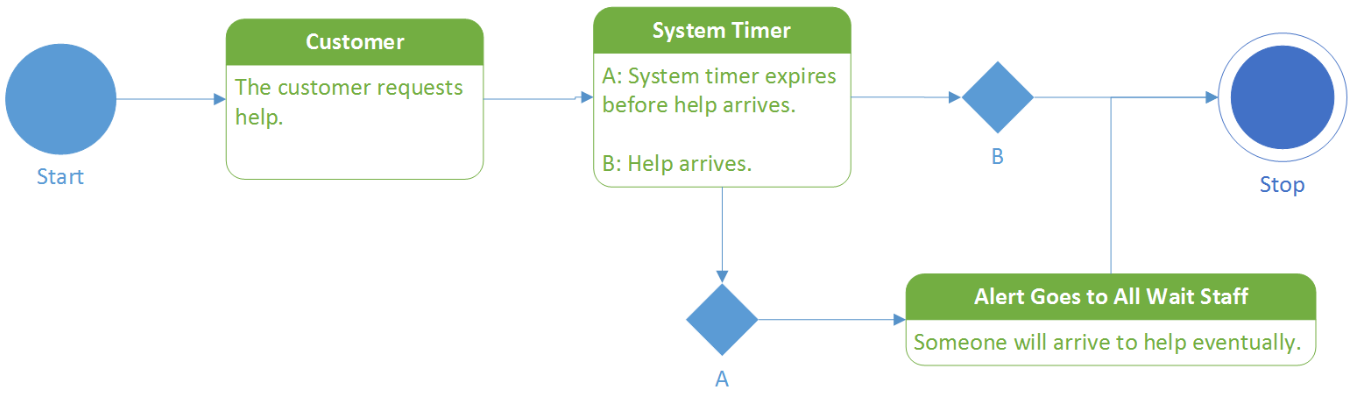
Relevant metadata includes which table is the order's destination, which member of the Wait Staff is assigned to that table, sub-total, sales tax, total, and an order or transaction identifier of some kind.

When an order gets confirmed and logged in the system, then the Kitchen Staff is notified and it enters the Kitchen Queue.

When an order emerges from the Kitchen Queue, it is fully prepared and ready for delivery to the Customer. The appropriate member of the Wait Staff is notified, and then the order enters the Wait Queue.

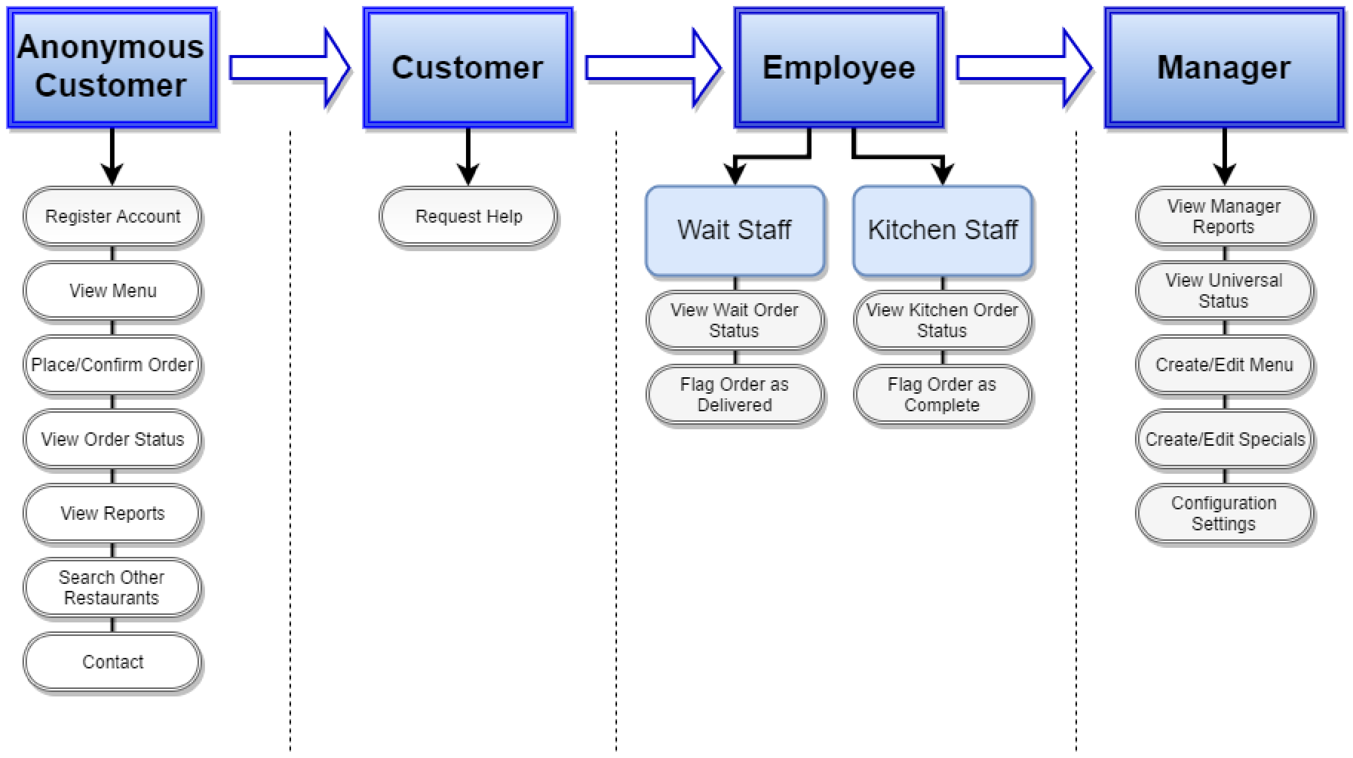
When an order emerges from the Wait Queue, it is on its way to the Customer. The final step of the process is to notify them of their order's imminent arrival.

Section 5B: Request Help


Start
Customer
The customer requests help.
System Timer
A: System timer expires before help arrives.
B: Help arrives.
B
Stop
A
Alert Goes to All Wait Staff
Someone will arrive to help eventually.

The above state transition diagram shows how the Request Help feature is intended to function. The system timer value will be set by the manager user-role using the Configuration subsystem.

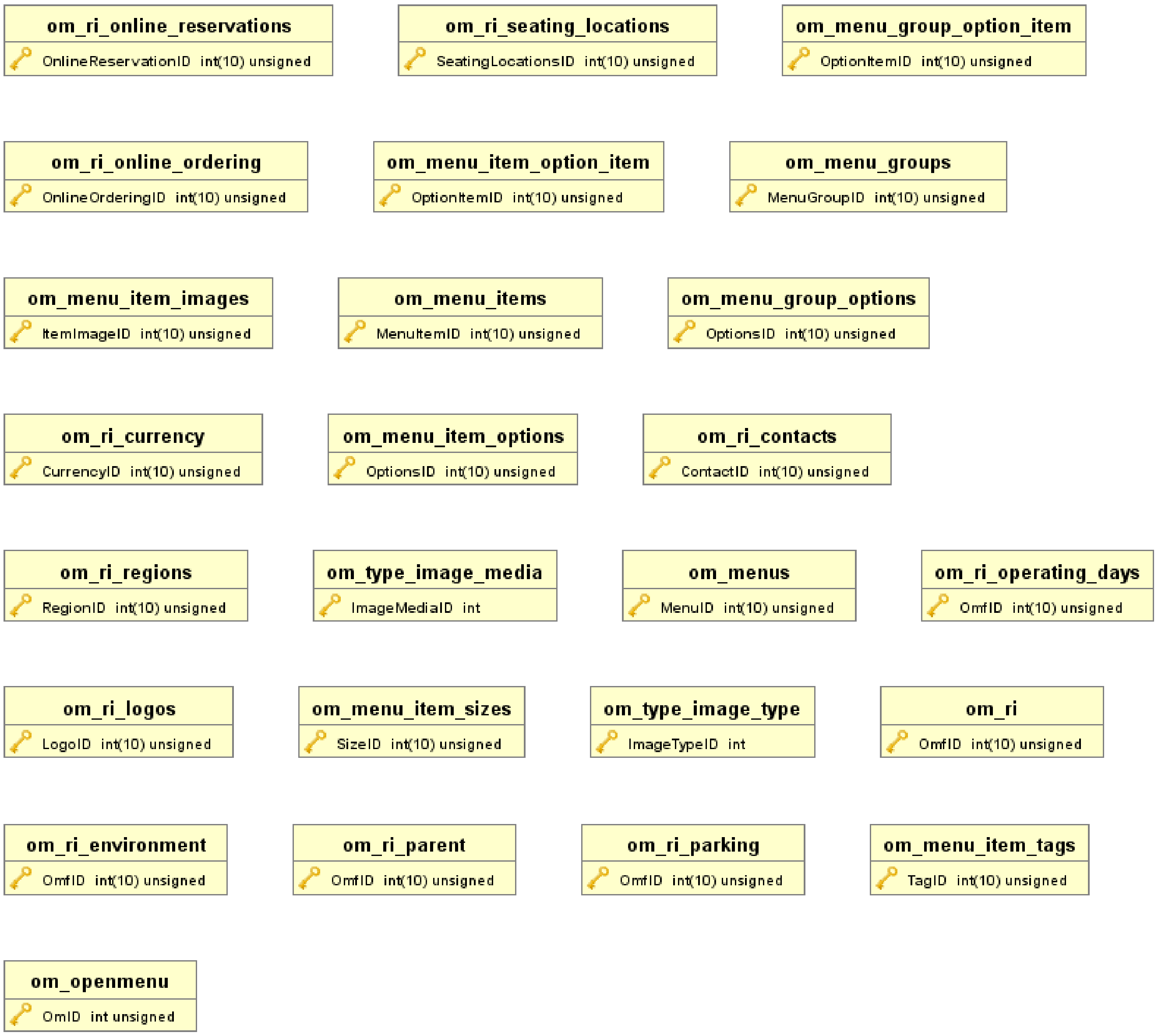
Section 5C: User Privileges



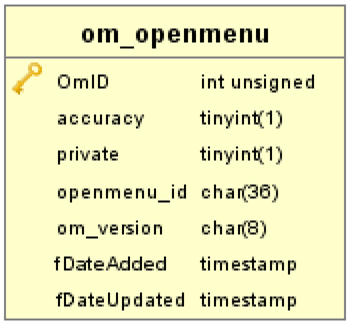
The permissions for each user-role are organized in a layered fashion. The above diagram is read from left to right. The user-role on the far left has the least privileges, and the user-role on the far right has the most privileges. Additionally, each role has all the privileges granted to the roles that preceded it. This was the logical solution, as employees will not always be on the clock: they could be customers too!

Section 5D: Database Schema

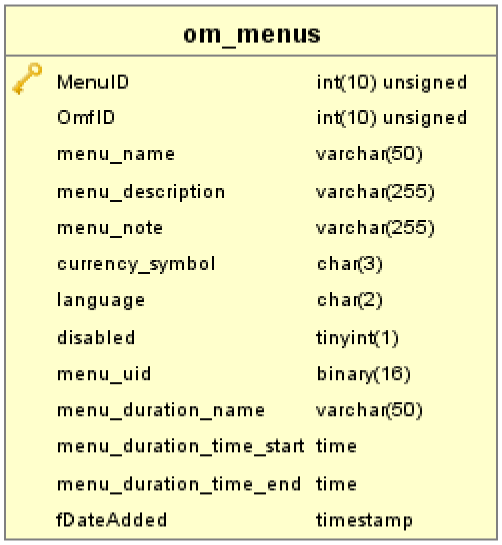
Note that this section deals primarily with the schema for the OpenMenu database. Drupal maintains user information in its own database. Below are all the tables present in the OpenMenu database along with their primary keys. This document won't go into complete detail for each table since it would make this document enormous; however, I will discuss the more important ones.



om_ri_online_reservations	OnlineReservationID int(10) unsigned
om_ri_seating_locations	SeatingLocationsID int(10) unsigned
om_menu_group_option_item	OptionItemID int(10) unsigned
om_ri_online_ordering	OnlineOrderingID int(10) unsigned
om_menu_item_option_item	OptionItemID int(10) unsigned
om_menu_groups	MenuGroupID int(10) unsigned
om_menu_item_images	ItemImageID int(10) unsigned
om_menu_items	MenuItemID int(10) unsigned
om_menu_group_options	OptionsID int(10) unsigned
om_ri_currency	CurrencyID int(10) unsigned
om_menu_item_options	OptionsID int(10) unsigned
om_ri_contacts	ContactID int(10) unsigned
om_ri_regions	RegionID int(10) unsigned
om_type_image_media	ImageMediaID int
om_menus	MenuID int(10) unsigned
om_ri_operating_days	OmfID int(10) unsigned
om_ri_logos	LogoID int(10) unsigned
om_menu_item_sizes	SizeID int(10) unsigned
om_type_image_type	ImageTypeID int
om_ri	OmfID int(10) unsigned
om_ri_environment	OmfID int(10) unsigned
om_ri_parent	OmfID int(10) unsigned
om_ri_parking	OmfID int(10) unsigned
om_menu_item_tags	TagID int(10) unsigned
om_openmenu	OmID int unsigned

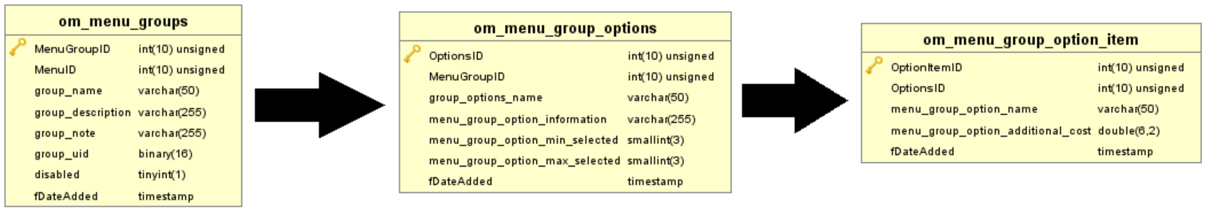


om_openmenu	
OmID	int unsigned
accuracy	tinyint(1)
private	tinyint(1)
openmenu_id	char(36)
om_version	char(8)
fDateAdded	timestamp
fDateUpdated	timestamp



om_menus	
MenuID	int(10) unsigned
OmfID	int(10) unsigned
menu_name	varchar(50)
menu_description	varchar(255)
menu_note	varchar(255)
currency_symbol	char(3)
language	char(2)
disabled	tinyint(1)
menu_uid	binary(16)
menu_duration_name	varchar(50)
menu_duration_time_start	time
menu_duration_time_end	time
fDateAdded	timestamp

Shown above are the core tables of the OpenMenu database. The **om_openmenu** table is purely for metadata regarding the currently implemented version of OpenMenu. The **om_menus** table contains all the top-level menu categories (Breakfast, Lunch, and Dinner for example). This table has a one-to-many relationship with the **om_menu_groups** table shown below. In other words, each instance of **om_menu** contains one or more instances of **om_menu_groups**.

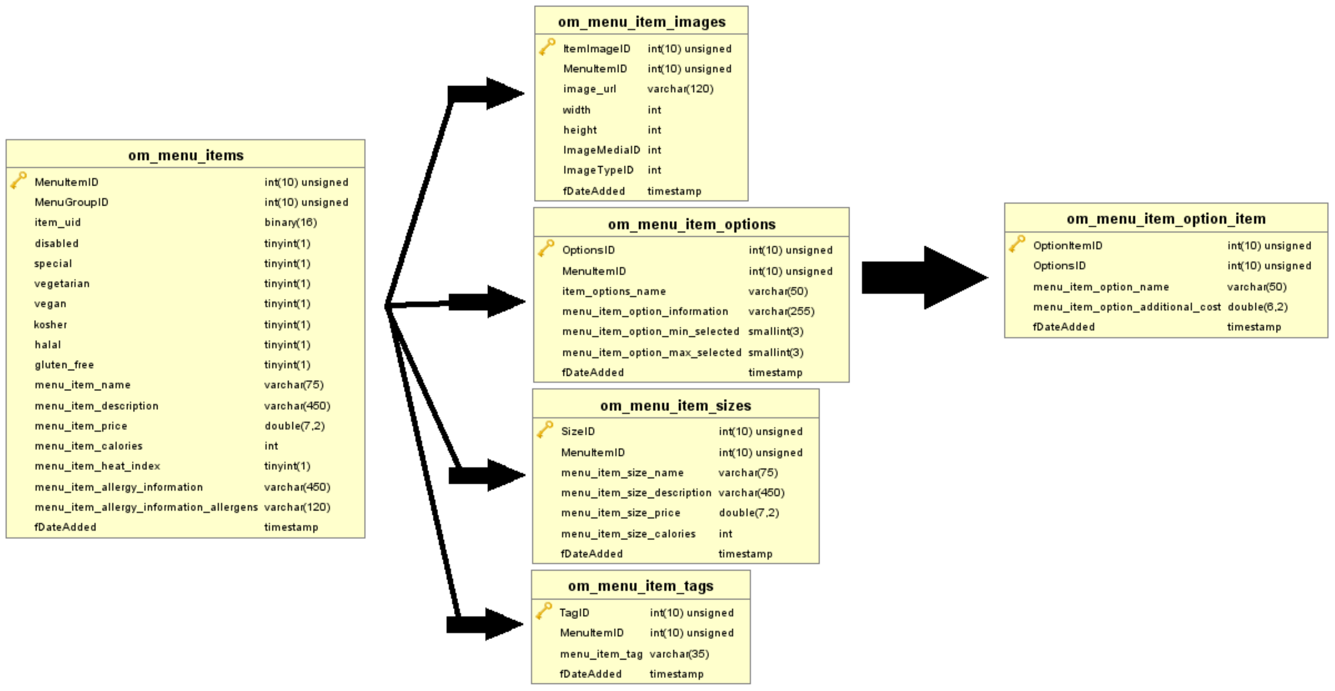


om_menu_groups	
MenuGroupID	int(10) unsigned
MenuID	int(10) unsigned
group_name	varchar(50)
group_description	varchar(255)
group_note	varchar(255)
group_uid	binary(16)
disabled	tinyint(1)
fDateAdded	timestamp

om_menu_group_options	
OptionsID	int(10) unsigned
MenuGroupID	int(10) unsigned
group_options_name	varchar(50)
menu_group_option_information	varchar(255)
menu_group_option_min_selected	smallint(3)
menu_group_option_max_selected	smallint(3)
fDateAdded	timestamp

om_menu_group_option_item	
OptionItemID	int(10) unsigned
OptionsID	int(10) unsigned
menu_group_option_name	varchar(50)
menu_group_option_additional_cost	double(6,2)
fDateAdded	timestamp

As you can see, each menu group can have one or more options, and a minimum or maximum can be set for the number of options selected for a given menu group. The arrows show that the indicated table is functionally dependent on the table that preceded it. Each instance of **om_menu_groups** has a set of menu items with which it is associated. This is shown on the next page.



om_menu_items

MenuItemID	int(10) unsigned
MenuGroupID	int(10) unsigned
item_uid	binary(16)
disabled	tinyint(1)
special	tinyint(1)
vegetarian	tinyint(1)
vegan	tinyint(1)
kosher	tinyint(1)
halal	tinyint(1)
gluten_free	tinyint(1)
menu_item_name	varchar(75)
menu_item_description	varchar(450)
menu_item_price	double(7,2)
menu_item_calories	int
menu_item_heat_index	tinyint(1)
menu_item_allergy_information	varchar(450)
menu_item_allergy_information_allergens	varchar(120)
fDateAdded	timestamp

om_menu_item_images

ItemImageID	int(10) unsigned
MenuItemID	int(10) unsigned
image_url	varchar(120)
width	int
height	int
ImageMediaID	int
ImageTypeID	int
fDateAdded	timestamp

om_menu_item_options

OptionsID	int(10) unsigned
MenuItemID	int(10) unsigned
item_options_name	varchar(50)
menu_item_option_information	varchar(255)
menu_item_option_min_selected	smallint(3)
menu_item_option_max_selected	smallint(3)
fDateAdded	timestamp

om_menu_item_option_item

OptionItemID	int(10) unsigned
OptionsID	int(10) unsigned
menu_item_option_name	varchar(50)
menu_item_option_additional_cost	double(6,2)
fDateAdded	timestamp

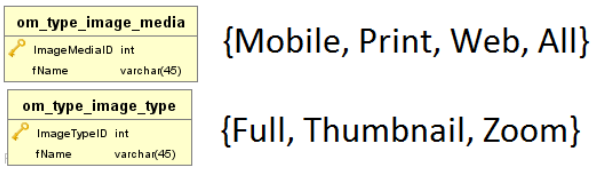
om_menu_item_sizes

SizeID	int(10) unsigned
MenuItemID	int(10) unsigned
menu_item_size_name	varchar(75)
menu_item_size_description	varchar(450)
menu_item_size_price	double(7,2)
menu_item_size_calories	int
fDateAdded	timestamp

om_menu_item_tags

TagID	int(10) unsigned
MenuItemID	int(10) unsigned
menu_item_tag	varchar(35)
fDateAdded	timestamp

These are the relevant tables for individual menu items. The arrows share the same semantics as the diagram on the previous page; each table being point to is functionally dependent on the table that precedes it (to the left) where the arrow originates. Each menu item has an image and description associated with it. There is a one-to-many relationship between the **om_menu_groups** and **om_menu_items** tables. Each menu group contains one or more menu items.



om_type_image_media

ImageMediaID	int
fName	varchar(45)

{Mobile, Print, Web, All}

om_type_image_type

ImageTypeID	int
fName	varchar(45)

{Full, Thumbnail, Zoom}

The above two tables have their values hardcoded, and I have provided them in the diagram. This is to facilitate a rich media experience. Is the thumbnail image we displayed on the menu too small? Simply click (if using a mouse) or tap (if using a touch interface) to see a full screen image. Want to zoom in? No problem! Additionally, images can be optimized for mobile (low resolution), print (high resolution, web (medium resolution) or scaled dynamically as an attempt at a “one size fits all” solution. OpenMenu provides a robust feature set and is easily extendable, which is one of the primary reasons we chose to use it.

Section 6: Assumptions and Risks

Section 6A: Assumptions

- We assume each client device will be accessing our application using a supported web browser. We plan on supporting the latest stable versions of Chrome, Safari, Internet Explorer, and Edge. We have chosen these specific browsers because they are the defaults for most mobile devices. Chrome is the default browser on Android-based phones/tablets (and their Chrome OS cousins). Safari is the default browser on iPhones/iPads (and their Mac OS cousins). Internet Explorer and Edge are the defaults on Windows PCs and Microsoft phones/tablets respectively. Therefore, we are confident that the application will be accessible to virtually all users regardless of the make and model of their device.
- We assume that each restaurant owner that chooses to use our product will sign up for an account with the OpenMenu system. This is not strictly necessary, it will allow them to search for and use other menus as a starting point, so that they don't have to create a menu completely from scratch (though they will also be able to do that, if they want).
- We assume that each restaurant owner that chooses to use our product will sign up for an account with Google to use the Google Analytics, Google AdWords, and ReCAPTCHA services. These are not strictly necessary, but the services are legitimately useful. Every owner should seriously consider using them.

Section 6B: Risks

- Matthew, our team leader, is the only member of the team with experience using Drupal. This presents a challenge due to the learning curve involved in learning a new technology. We have been and will continue to put a great deal of effort into overcoming this risk by meeting regularly and having him guide our development. We also considered WordPress, but the team's experience with that platform wasn't much better; the only team members with experience using WordPress are Matthew and Slaven.
- It is entirely possible that there will not be near as much demand for this service as we expected, and if that is the case, then it is unlikely that our application will be adopted in any production environments.