4. Design Rationale

The V-Menu works to bring convenience to the customer by providing a quick and reliable service application using modular software. The first step is to build a system that can import data from an online source in a consistent format; one that fits the structure of the database. Database implementation of the various restaurants is a crucial step into building the application. The application also requires the use of modular design, because multiple widgets and components need to work hand-in-hand to provide the accessibility, while also being simple enough for the average user to understand. An example of this would the user’s first interaction with the application. The user selects restaurants in proximity to their location using a combination of an online map, global positioning system, third party systems and a database using simple icons and features.

Critical Issues

- importing data from an online source consistent with the format of the application

- multiple widgets working in tandem with each other

- handling the database

- configuring use cases that require overhead (alerts)

- may have to trade design and good quality of life features for simplicity and consistency

- learning curve variance between group members

- Deciding on which application building tool to use

The issues that might come up that are a priority is making sure, first and foremost, that all the features work in tandem with one another and don’t need constant fixing. Importing data in a certain format may cause the system that handles the view of the information to look messy to the user. The manager, for example, might have a special event that he or she can add to the menu specifically for that restaurant or day that may conflict with the imaging of the interface. There’s also the issue of configuring interfaces that require someone to oversee functionality like alerts. Alerts being that the customer requests immediate service and the employee assigned to that table responds to that alert in an adequate time frame. If the alert is not responded to in a certain time, then the alert is sent to everyone else. It’s important to configure an interface to achieve each step by step process to give the application the feel of convenience and not a waste of memory space.

Because the application requires above average programming knowledge to get all the part working together, an issue brought up is the learning curve differences between group members and the software tool to be used. Because the skill and knowledge difference within the group, more time is focused on getting below average programmers up to speed using new software whether it’s sitting down and teaching them or explaining it to them as the group moves along.

Trade Off’s

- User devices will need wireless internet access to reach the server; either Wi-Fi or a data plan. Restaurant owner may wish to provide free Wi-Fi access to mitigate this risk.

- Performance can be a serious issue if the server is under heavy load due to bottleneck. Likewise, there is a single-point of failure; if the server goes down, the application is useless. Restaurant owner may wish to invest in a cloud-computing platform such as Amazon’s AWS or Microsoft’s Azure 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. In our case, clients are nothing more than internet browsers; therefore, the client’s only responsibility is to render content.

- Restaurant owner may wish to invest in dedicated IT personnel to perform regular basic maintenance duties.

5. High Level Design

a. The V-Menu is an application that tablet and I-phone users can use to order food on the go or in the restaurant at their own convenience. Clients will be able to grab data from the website of a restaurant near their location and the server will process their information to provide the requested service.

6. Detailed Design

Feature Oriented Decomposition

High-Level Design:

The customer has access to restaurant reports the system averages over a period of time. Customers will have the ability to register an account with the app to store their own preferences and order history. The manager has the ability to import their own data into the menu database and add special offers. Customer will use the alert based feature to request assistance from a waiter.

Low-Level Design:

The customer has an option to view reports based on what the average consumer has ordered. The customer then enters the menu and selects different menu items to order. The order is confirmed then processed by the system and sent to the kitchen staff to be processed. The orders from the customer are sent into a database, that the manager can access when he/she is ready. When the customer registers an account they can view their purchase history as well.

Functional Decomposition

Users will create or make an account within the application. Employees and Managers (Administrators as well) will have features assigned to them exclusively. Customers will not have exclusive features within the apps main purpose. The kitchen staff will be a subset of the Employees who will use a queue to process orders and the other Employees who are not part of the kitchen staff will utilize alerts, table assistance, and a menu to assist other customers. The Manager will utilize all available features to the application including an account creation page and report page the contains statistics relative to that location, such as types of orders purchased and the average number of customers per day.

Data-oriented Decomposition

High-level:

The menu is being handled by the open-menu module which has its own database and schema. The orders table will be accessed by the module to find which order to process via timestamp.

Low-level:

Data is sitting in a database and the modules will access the database whenever the system needs to access it.

Process-oriented Decomposition

High-level:

The system’s main task is to append the data selected in the customer’s menu and send that data to a queue so the information can be processed by the Kitchen staff. Once the food is processed, the system alerts the waiter assigned to the table.

Low-level

When the customer orders from the menu, they select items and put them into a list and confirm the order, the system checks the options selected by the customer such as table number, number of orders received and the preferences made by them. The system appends this data and stores this appended data into a queue. The Kitchen staff are processing this order along with others within the queue. When an order is fully processed, the system messages the waiters and tells them the table they are assigned to is ready to receive their order.

Event-oriented Decomposition

High-level:

The customer requests assistance and the alert lights up for the waiter assigned to the table and after the waiter fulfills the service or is free to address the request the employee clears the alert.

7.

Assumptions

- Once we know the initial interface, additional pages will be easier to implement

- Programming the interface is easy to execute

- All necessary features will be implemented successfully

- Potential for profit

- Owner signs up for operation account to get their menu up and running

Once the team knows the basic interface layout and builds one successfully, building additional pages should be easier to implement. The features will take some recalibrating but the overall design will be there to piggyback off of. Adding features like a database and login/logout sessions will be a quick process and adding the functionality to each use case scenario will also be a simple to execute.

The team is assuming that the application is going to be well received and be worth the risk to restaurant owners, increasing profit. Because the application’s primary function is to provide convenience, the team expects the restaurant owner to do a test run of the application and enjoy all the qualities the app has to offer.

Risks

- Chance that learning the new languages will slow down progress

- Databases may not work as intended

- Maybe too much detail is given in the interface

- May need to remove features previously planned / prioritization

- Output might not be deterministic

- If third party software goes down the app may not work

-Team not familiar with Drupal, sunk time learning

Chance of learning new software and its syntax can slow down the progress of building the application, meaning features previously intended to be in the final build may not make it; either due to time constraints or complexity of the implementation.

Databases may not work as intended due to the amount of data being stored, the format which is stored in and various technical faults.

Too much detail may be given in the interface to make it pretty or grab attention. An interface with too much detail can distract users from doing what they planned on doing, such as ordering and watching for alerts, or look messy and just dismiss the app altogether.

If third-party software such as the server where information is stored or the website you get the application goes down, the functionality of the application stops working, making the application itself useless.