Assessment 2: Object Design

Software Architecture & Design
Semester 1 2020

Luke McWha & Cory Mifsud
#101 092 956 #101 484 964

Table of Contents

1. Executive Summary

When expanding a business, improving the processes of that business is a must to make it cost effective and sustainable into the future. In expanding the number of customers the client is wanting to cater to means they can expand their business into the future. Tasks such as waiting tables, passing orders to the kitchen and having customers pay for their order need to be efficient to get the most out of the business and staff.

The solution is to have a piece of software that will automate many of these tasks without the need for staff intervention. Through this software, time will be saved, the number of orders will increase and order/ transaction mistakes are reduced. In the long run, the business will save more money on food by making better informed decisions when ordering stock through the statistics gathered; save on staff wages as the software can take orders and create invoices that would otherwise need human intervention to complete.

The software is being designed in a modular fashion. The use of an Event Dispatcher creates a simple and well defined interface for internal communication throughout the app. Within the current scope this is extremely useful in notifying the FOH staff when an order has taken too long to be complete. This architecture allows for the software to be extended with new features in the future due to the generic nature of the pattern.

The belief is that this solution presented will greatly improve the workflows of this cafe and help encourage growth into the future.

2. Problem Analysis

The Software Requirements Specification outlined various points, views and requirements for the project in both function and quality that must be adhered to in the planning and development phase. The key functional points described within that document encapsulate the end goal and use cases for the project.

These key functional points are outlined below:

- Allow a customer to place a reservation online
- Allow a customer to order
- Display the menu online for customers to view
- Notify and give details of a customer's order to the kitchen
- Notify front of house staff whether a customer has waited too long for their order
- Create an invoice for a table’s order and receipt for payment
- Handle monetary transactions between a customer and the cafe
- Logging order and payment statistics and data to a database

2.1. Assumptions

The following are assumptions made when designing the project:

A1 All servers / equipment required is available.

A2 3rd party software used for the transaction process is integrated with the project.

A3 3rd party accounting software can process split payments.

A4 System will not have more than 150 concurrent users.

A5 All menu items will always be available.

A6 Admin users will create accounts.

A7 Frontend (UI) has interaction functions to modify the ViewModel.

A8 Orders can not be made online.

A9 Orders will have a timer allocated to them on creation.

A10 Payment Service will deal with an external payment entity.

A11 Database service will interact with a sole database.

A12 When an object with a dispatcher attached, the dispatcher will notify all subscribers of an event.

A13 There will only be one instance of the Menu.

A14 Menu items can be changed.

A15 Reservations can be made by a customer online or in person.

2.2. Simplifications

Based on assumptions made in Section 2.2,

- The external payment services are assumed to be plug and play, we assume no matter the actual product to process the payments, it will interface the same way.
- The database is assumed to be plug and play, no matter the database type used, it will interface the same way.
- The system is loosely built as a Model View Viewmodel architecture. Our design does not include classes for the View as this model will work regardless of the frontend framework used.

2.3. Design Justification

It is assumed that the software will contain a database and it is specified that varying services will interface with that database. In the same way that the external banking software exists and will interface with the project through varying services.

Below outlines the various patterns and classes that were considered but we decided not to incorporate into the solution. The final classes are outlined and justified further in the document. The design is created to make a flexible environment for future growth and simply the objects we use.

2.4. Patterns Considered

2.5.1. Command Pattern

The Command pattern is a behavioural design pattern where command objects will modify the behaviour of other objects. For example, if an object or the user through a user interface invokes the 'Create Reservation' command. This command would trigger processes by the required objects to create a reservation.

Instead of this design we opted for an Event Dispatch pattern to handle internal communication between objects upon an event and we assume the User Interface will interface with objects themselves to invoke their behaviour.

2.5. Discarded Classes

2.5.1. Application Session

The application session class is designed to be a central point for data regarding the overall session of the app. This could contain user data, order data, table data for the app.

We removed this from our design as it was a God class if implemented. The usage of singletons for major container classes erased the need for a single access point.

2.5.2. Menu Components

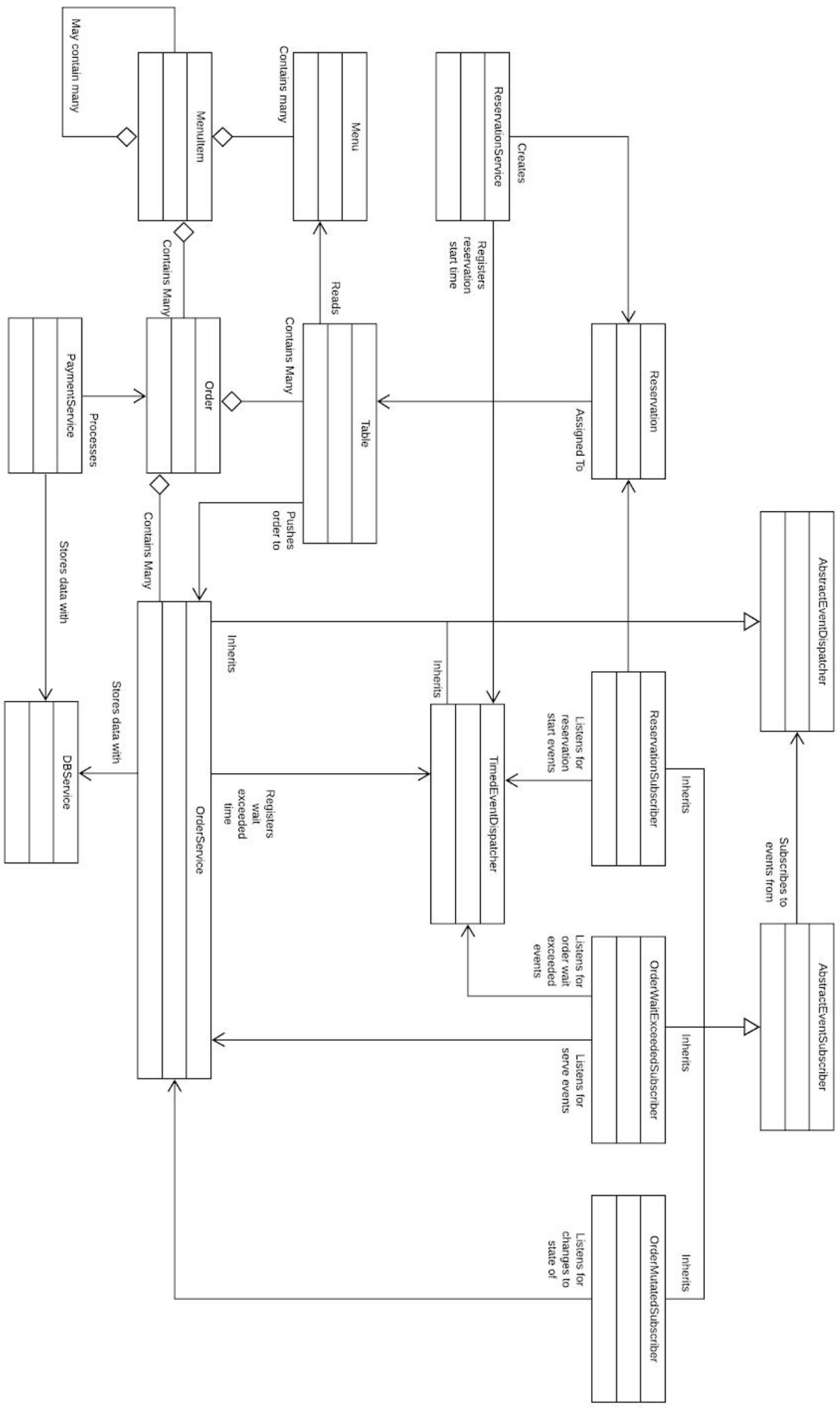
Initially, due to different instances of the application requiring different usage permissions in the menu (Web portal displaying the menu, In-restaurant devices displaying and allowing orders, admin portal allowing editing) we considered adding ‘permission’ components. However we decided that this was an unnecessary complication and could be implemented in other ways.

3. Candidate Classes

3.1. Class List

- Menu
- MenuItem
- ApplicationSession
- Table
- Order
- PaymentService
- DatabaseService
- OrderService
- AbstractEventSubscriber
- AbstractEventDispatcher
- TimedEventDispatcher
- OrderMutatedSubscriber
- OrderWaitExceededSubscriber
- ReservationSubscriber
- Reservation

3.2. Class Diagram



3.3. CRC Cards

3.3.1. Menu

The singleton menu provides access to a single unified list of menu items for viewing, ordering, creating, and editing.

Component: Menu	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Record items within the menu.	MenuItem
Display items from within the menu.	MenuItem
Provide interface for editing items within the menu.	MenuItem

3.3.2. MenuItem

Encapsulates the data associated with a singular item orderable within the restaurant.

Component: MenuItem	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Records name, ingredients, price, availability etc. of a food, drink or other orderable item.	n/a

3.3.3. Table

Encapsulates the details associated with a physical table in the restaurant.

Component: Table	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Create anew, or change existing associated orders of items from the menu.	Menu, MenuItem, Order, OrderService
Record any associated reservations.	Reservation
Record any created orders.	Order
Determine aggregated payment details of all recorded orders.	Order

3.3.4. Order

An Order object encapsulates a single order of menu items made by customers at a table.

Component: Order	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Record a list of ordered MenuItems.	MenuItem
Determine aggregated payment details of recorded MenuItems.	MenuItem

3.3.5. PaymentService

Provides an interface for handling payment operations, involving; preparing invoices and receipts, using external payment services, and recording all payment interactions persistently.

Component:	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Create an invoice from a table’s orders’ payment details.	Table, Order
Process payment for a table’s orders through an external service.	Table, Order
Create a receipt for a completed payment.	Order
Log completed payment details to persistent database.	DatabaseService

3.3.6. DatabaseService

Provides an interface for maintaining a connection to the application’s database, and the associated CRUD operations.

Component: DatabaseService	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Manage connection instance to application database.	n/a
Provide interface for storage and retrieval of data.	n/a

3.3.7. OrderService

The singleton OrderService provides a unified point of truth regarding operations upon pending orders. This not only includes a managed CRUD interface, but the firing of related events to the relevant parts of the application.

Component: OrderService	
Sub-Classes: n/a	**Super Classes: AbstractEventDispatcher**
Responsibilities	**Collaborators**
Hold a weak queue of orders currently being processed by the system	Order
Provide an interface for the creation, mutation, and finalisation of orders.	n/a
Notify subscribers of the creation and mutation of orders.	OrderMutatedSubscriber
Notify subscribers of the finalisation (serving) of orders.	OrderWaitExceededSubscriber
Register timed schedules (waiting limits) upon creation of new orders.	TimedEventDispatcher
Log statistics of orders passed through the queue in the application database	DatabaseService

3.3.8. AbstractEventSubscriber

Denotes an object that listens for notification of the firing of a certain type of event, and reacts in part.

Major base component of Event Listener pattern.

In the context of this application, instances of subclasses of this class logically create user roles, as each application instance will listen for different system events.

Component: AbstractEventSubscriber	
Sub-Classes: OrderMutatedSubscriber, OrderWaitExceededSubscriber, ReservationSubscriber	**Super Classes: n/a**
Responsibilities	**Collaborators**
Subscribe to be notified upon the triggering of a certain type of event.	AbstractEventDispatcher
Listen for, and respond to notifications of subscribed events.	AbstractEventDispatcher

3.3.9. AbstractEventDispatcher

Denotes an object that notifies a number of subscriber objects of the triggering of an event, either in response to external stimulus or from an internal process.

Major base component of Event Listener pattern.

Component: AbstractEventDispatcher	
Sub-Classes: OrderService, TimedEventDispatcher	**Super Classes: n/a**
Responsibilities	**Collaborators**
Keep record of subscribers	AbstractEventSubscriber
Notify subscribers of event fire.	AbstractEventSubscriber

3.3.10. OrderMutatedSubscriber

Notifies the kitchen that a new order has been created, or an existing order has been changed.

Component: OrderMutatedSubscriber	
Sub-Classes: n/a	**Super Classes: AbstractEventSubscriber**
Responsibilities	**Collaborators**
Listen for creation or mutation events for orders currently in the order queue.	OrderService
Update GUI with new order state.	Order

3.3.11. OrderWaitExceededSubscriber

Notifies FOH staff in the case that a table has made an order, and too long a time has elapsed without the order being finalised (served). Allows FOH staff to adequately compensate the table.

Component: OrderWaitExceededSubscriber	
Sub-Classes: n/a	**Super Classes:**
Responsibilities	**Collaborators**
Listen for order finalisation (serving) events.	QueueService
Listen for scheduled time elapsed events.	TimedEventDispatcher
Update GUI/Notify users of wait exceeded.	Order, Table

3.3.12. ReservationService

Provides an entry point for reservations into the application from an external source. Creates reservations and registers their times to schedule a notification to front of house staff.

Component: ReservationService	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Provide an external interface for reservation creation inside the application.	n/a
Create reservations.	Reservation, Table
Schedule created reservation times to notify FOH staff.	TimedEventDispatcher
Record Reservations	Reservation

3.3.13. ReservationSubscriber

Notifies front of house staff that customers with a reservation are due to arrive, and to prepare the associated table.

Component: ReservationSubscriber	
Sub-Classes: n/a	**Super Classes: AbstractEventSubscriber**
Responsibilities	**Collaborators**
Update GUI/notify user upon a reservation being due to arrive.	Reservation, Table
Listen for reservation time arrival events.	TimedEventDispatcher

3.3.14. Reservation

Encapsulates the data of a singular reservation within a given timeslot assigned to a table.

Component: Reservation	
Sub-Classes: n/a	**Super Classes: n/a**
Responsibilities	**Collaborators**
Record data associated with a reservation (ie. date & time, number of people, special dietary requirements, notes, assigned table)	Table

3.3.15. TimedEventDispatcher

Acts as an alarm clock, allowing for scheduling the triggering of events at given future times, then notifying any listeners at those times.

Component: TimedEventDispatcher	
Sub-Classes: n/a	**Super Classes: AbstractEventDispatcher**
Responsibilities	**Collaborators**
Allow interface for registering scheduled events.	n/a
Notify subscribers of scheduled time elapsing or arriving.	ReservationSubscriber, OrderWaitExceededSubscriber

4. Design Quality

4.1. Design Heuristics

H1	No 'God' classes should exist.
H2	All classes have one key goal.
H3	Abstract class must be base classes
H4	Base classes should be abstract classes
H5	Abstract classes are created as generic as possible.
H6	Generic methods are created high in the inheritance chain
H7	Concrete classes provide specific functionality based on its abstract parent class.
H8	Service classes modify only one specific aspect of the data.
H9	Only one service class of each type can exist.
H10	Each external entity must have a Service class.
H11	Code employs a consistent syntax/ structure for classes and architecture.

4.2. Design Patterns

4.2.1. Creational Patterns

4.2.1.1. Singleton Pattern

Implementing a class as a singleton ensures only one instance of that class is instantiated, and provides a universal reference to that instance.

It is implemented within classes that should only exist once globally, including:

- ReservationService
- OrderService
- Menu
- PaymentService
- DatabaseService

4.2.2. Structural Patterns

4.2.2.1. Model View ViewModel (MVVM)

MVVM splits an application into three logical structures:

Views, entailing classes related to the graphical representation of the application. These have been omitted from our design.

Models, entail classes related to the structuring and storage of data. These include:

- Reservation
- Order
- Table
- MenuItem

Finally ViewModels, entailing classes related to the interfacing and binding of models and views. These include:

- OrderService
- ReservationService

This approach allows a layered approach for modifiability, and allows a design specification which omits technology-specific graphical classes.

4.2.3. Behavioural Patterns

4.2.3.1. Observer Pattern

The observer pattern allows an arbitrary number of listeners to subscribe to an event publisher to receive notification of the triggering of events.

Publisher classes include:

- AbstractEventDispatcher
- OrderService
- TimedEventDispatcher

Subscriber services include:

- OrderMutatedSubscriber
- ReservationSubscriber
- OrderWaitExceededSubscriber

This pattern has been employed here to logically create user roles - that is, each instance of the application running on a different device can be subscribed to different system events, depending on what the user of that device needs to know about and respond to.

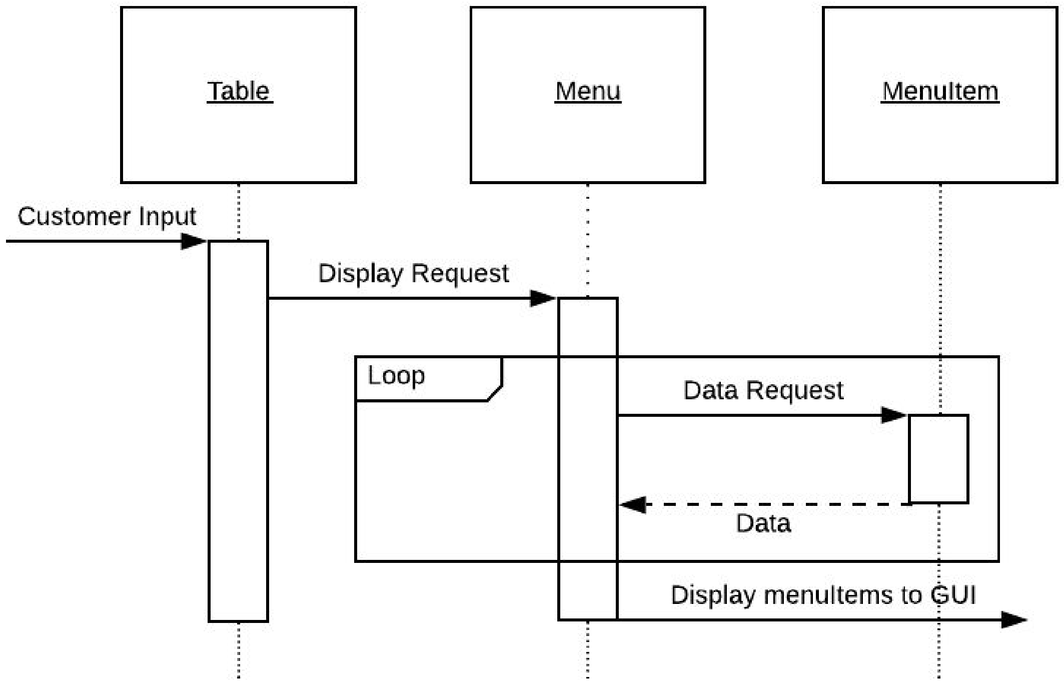
5. Bootstrap Process

1. *Main* instantiates **Menu**
2. **Menu** instanciates **MenuItem**s
3. *Main* instantiates *services* (**OrderService**, **DatabaseService**, **PaymentService**, **ReservationService**)
4. *Main* instantiates **Table**s
5. *Main* instantiates **TimedEventDispatcher**
6. *Main* instantiates *subscribers*, based on user roles:
    a. *Kitchen*: **OrderMutatedSubscriber**
    b. *FOH*: **ReservationSubscriber**, **OrderWaitExceededSubscriber**
7. Upon customer input, **Table** instantiates order and pushes to **OrderService**.
8. Upon external API input, **ReservationService** instantiates **Reservation**.

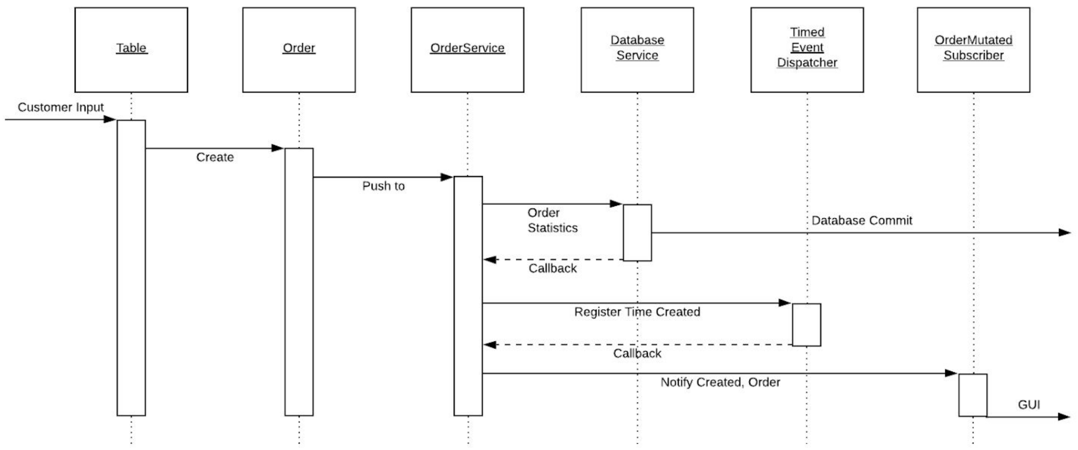
6. Verification

Below will present verification of this design through use case examples.

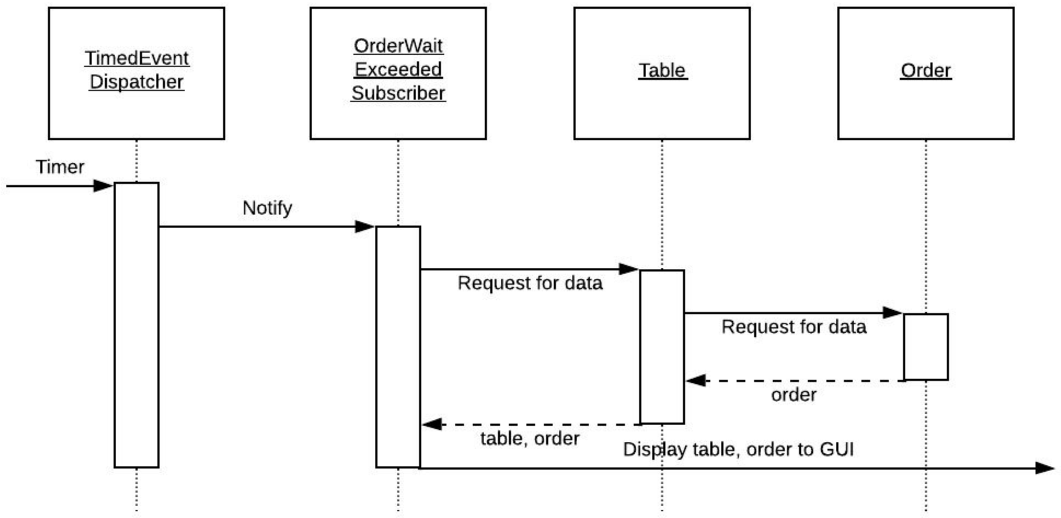
Example 1.1 - Customer views menu



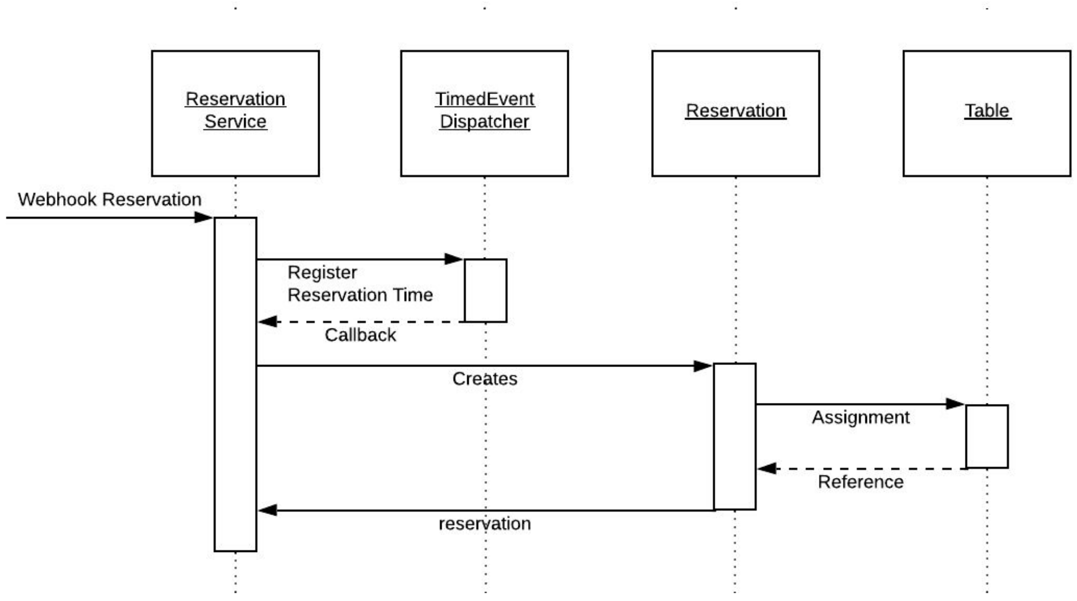
Example 1.2 - Customer makes order



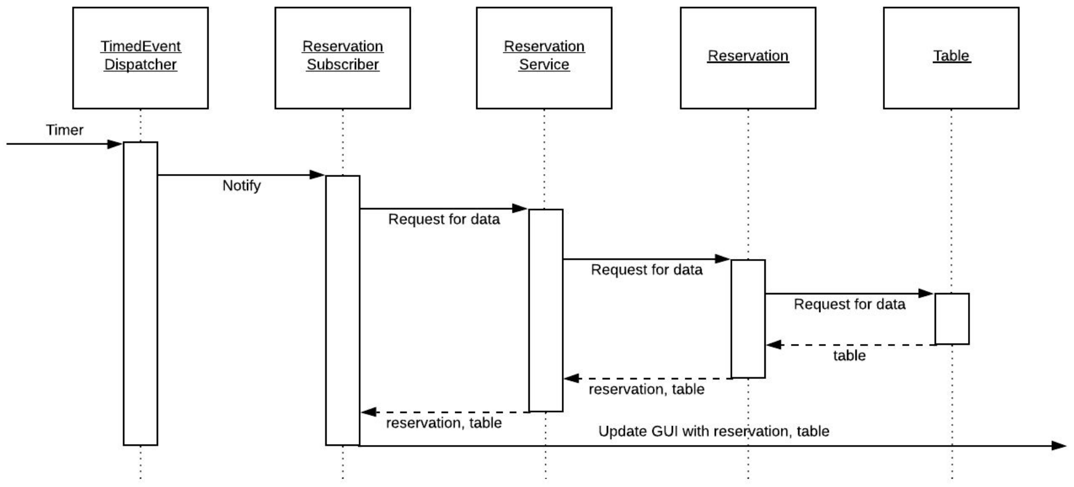
Example 1.3 - Order takes too long to be served



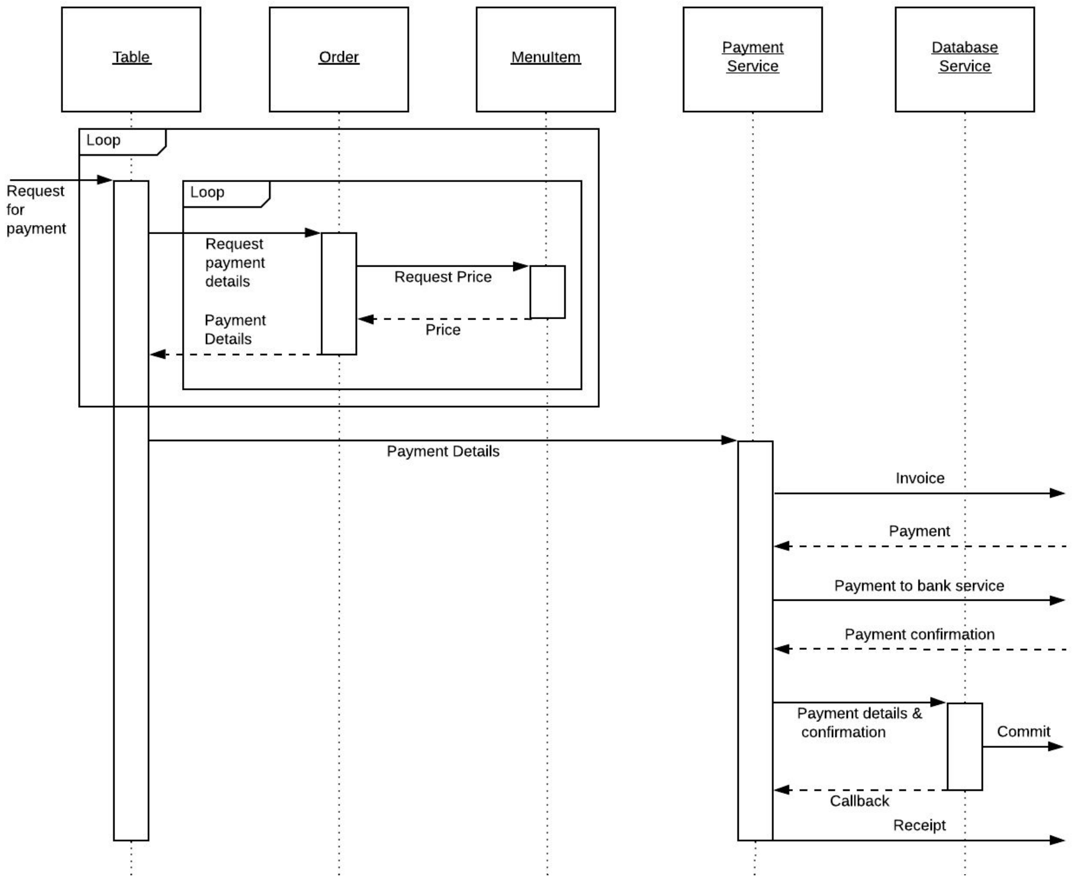
Example 2.1 - Customer makes reservation remotely



Example 2.2 - Reservation time arrives



Example 3 - Customer pays for order



7. Appendix - Software Requirements Specification

Table of Contents

1. Introduction

This specification document has been created to define the software requirements, provide an outline and scope for this software project. This document will cover a range of points on the project lifecycle and intends to define all functionality, assumptions, problem areas, tasks, relationships and possible solutions. Further, this document will incorporate thoughts and a priority of attributes that should be taken into consideration in planning and development.

2. Project Background

2.1. Overview

The Cuddly Wombat (client) is a café/restaurant that is planning to expand its business. In expanding, it will go from a capacity of about 50 people, to allow the cafe to host approx 150 customers.

The Cuddly Wombat's system is lowtech and requires many hours of manual labour/ intervention. The tasks completed include taking orders from guests, passing the orders on to the kitchen, accounting etc. To further expand the business the Cuddly Wombat realise they need to improve their processes which includes the introduction of new software to streamline these processes.

2.2. Goals and Objectives

The new system will support a number of features: Table reservations, taking customer orders, informing the kitchen about the orders, informing staff about customers who have waited a long time for their order, customer receipts, handling payments. There will also be an online ordering solution incorporated for takeaway meals. These aspects combined complete the goal of scaling the cafe's processes to accommodate more customers.

2.3. Domain Vocabulary

Below defines various words used throughout this document:

Word	**Definition**
Quality Attributes	Aspects, when considered together, that define the quality of the project.
CRUD	Create, Read, Update, Delete function for a database
FOH	Front of House cafe staff members
Front of House	Staff members who interact with customers
Accounting processes	The movement of money from the customer to cafe.
Standard Web Search	Searching for website through a search engine eg. Google
Standard system data exchange	Data (such as an order) moving from one section of the Applications to another section
Standard Database saving	Saving data from the application to an external data store
UX	User Experience
UI	User Interface

3. Requirements

3.1. Functional Requirements

- Create a table reservation for a customer
- Create a customer order
- Create an online customer order for takeaway
- Notify and give details of a customer's order to the kitchen
- Notify front of house staff whether a customer has waited a long time
- Create customer receipts for a customer's order
- Handle monetary transactions between a customer and the cafe
- Logging data to a database

3.2. Assumptions

- When paying the customer will interact with a staff member to fix up payment.
- Accounting processes are external to the system.
- External entities processing speed will be adequate for eftpos purposes.

- Website does not facilitate remote orders, only viewing of the menu.
- Required hardware will be procured.
- System will not have more than 150 concurrent users.

3.3. Scope

The system shall support reservations, taking orders from customers, informing the kitchen about these orders, informing the staff about customers who have had to wait for longer than a predetermined time for their meal (to allow these customers to be reimbursed), creating invoices and receipts for customers, and handling payments. The gathering of statistics will be incorporated to give a better overview of menu items customers are ordering. Menus will be available online to inform customers of the food that they offer and allow customers to order from a takeaway menu.

4. Problem Domain

4.1. Pain Points

- Manual work
- Current processes are not scalable
- Speed of orders
- Accuracy of orders
- No way to record

4.2. Domain Entities

- Menu Item - A single drink or dish that a customer can order.
- Menu - A set of menu items. Presented on the website.
- Order - A set of menu items ordered by a customer.
- Invoice - Entails the price, quantity, and table details of an outstanding order.
- Table - Corresponds to a physical table in the restaurant.
- Table Assignment - Corresponds to a customer's occupation of a table. A weak entity between tables and reservations, differs by having allocated customers in time periods and associated orders.
- Reservation - A future table assignment allocation.
- Payment - A completed payment process corresponding to an invoice.

4.3. Actors

There are 5 actors in this system. Each of these interact with the system in a different way. These actors include:

- Customer
- Kitchen
- FOH Staff
- External Banking/Accounting System
- Statistic Database

4.4. List of Tasks

- Make reservation through website
- Manage reservation
- View menu through website
- Manage menu through website
- Generate invoice for order
- Make payment on invoice and generate receipt.
- Allocate customer cover to table.
- Place or change order on table.
- Log information to database.
- Notify FOH staff on order wait exceeded.
- Receive and action order.

4.5. CRUD Check

Task/Entity	Order	Invoice	Payment	Table Allocation	Table	Reservation	Menu	Menu Item
Make reservation through website						C		
Manage reservation						RUD		
View menu through website							R	R
Place order	C			R				R
Generate invoice for order on table	R	C						
Receive and action order	R							
Make payment on invoice		R	C					
Allocate customer cover to table				CR	R	R		
Log order statistics	R							
Notify FOH staff on order wait exceeded	R							

Task/Entity	**Order**	**Invoice**	**Payment**	**Table Allocation**	**Table**	**Reservation**	**Menu**	**Menu Item**
Manage Menu							U	CUD
Missing?		UD	UD	UD	CUD		C	

4.5.2. Missing Tasks / Justification

As shown above, Table Allocations and Tables are missing Update and Delete, and Create, Update and Delete operations respectively. These are required to support movements of customers and changes in the restaurant's layout. This results in new tasks:

- Move or remove customer table allocation.
- Edit table layout.

There will never be more than one menu, and it will never be deleted, so create and delete operations are unneeded.

Invoices and payments are immutable records and thus do not require update or deletion operations. In the event of a customer refund, new records are created to balance the old ones.

The lack of a read operation on payments highlights a need for a new task to view historical payment data.

- View payment history.

4.6. Entity Relationship Model

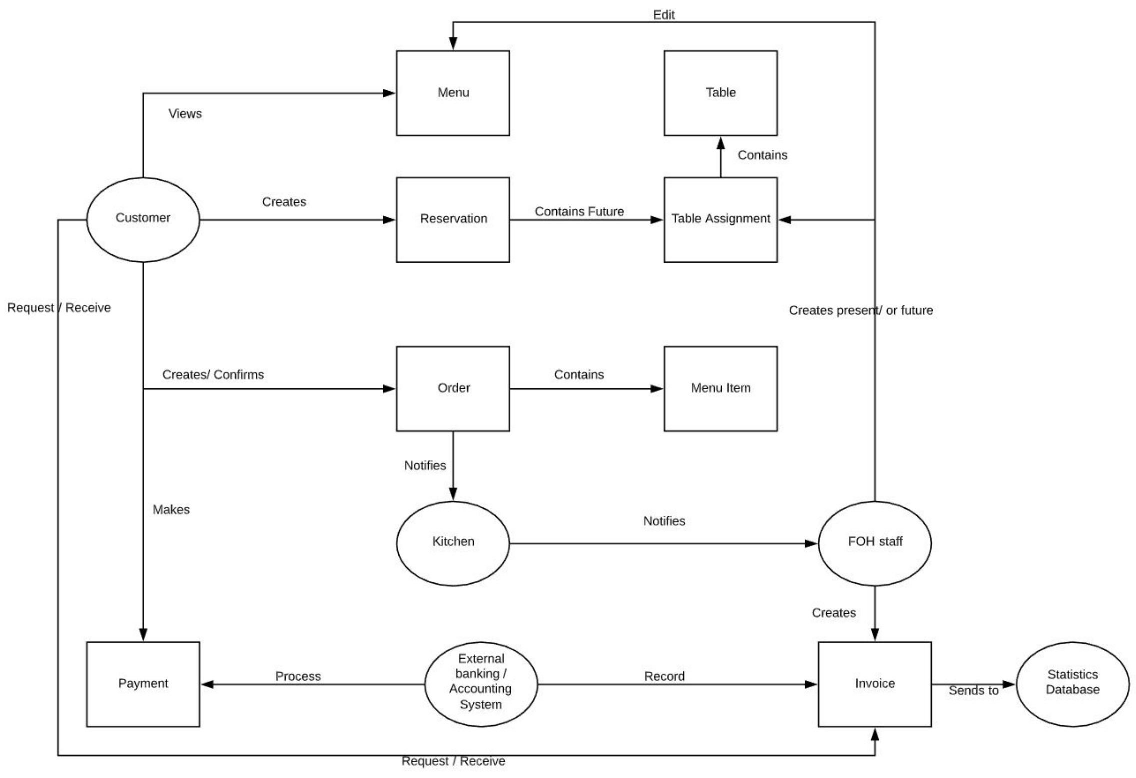


Fig. 1, ERM

4.7. Workflows

4.7.1. Online Menu Viewing

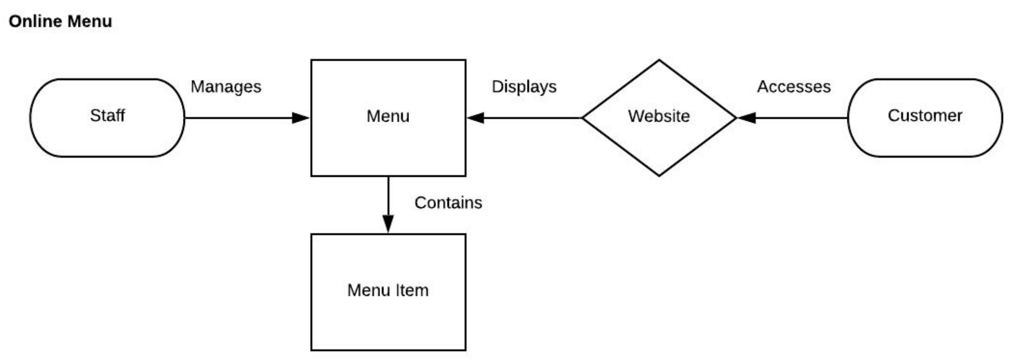


Fig. 2, Workflow 1

4.7.2. Payment Processing

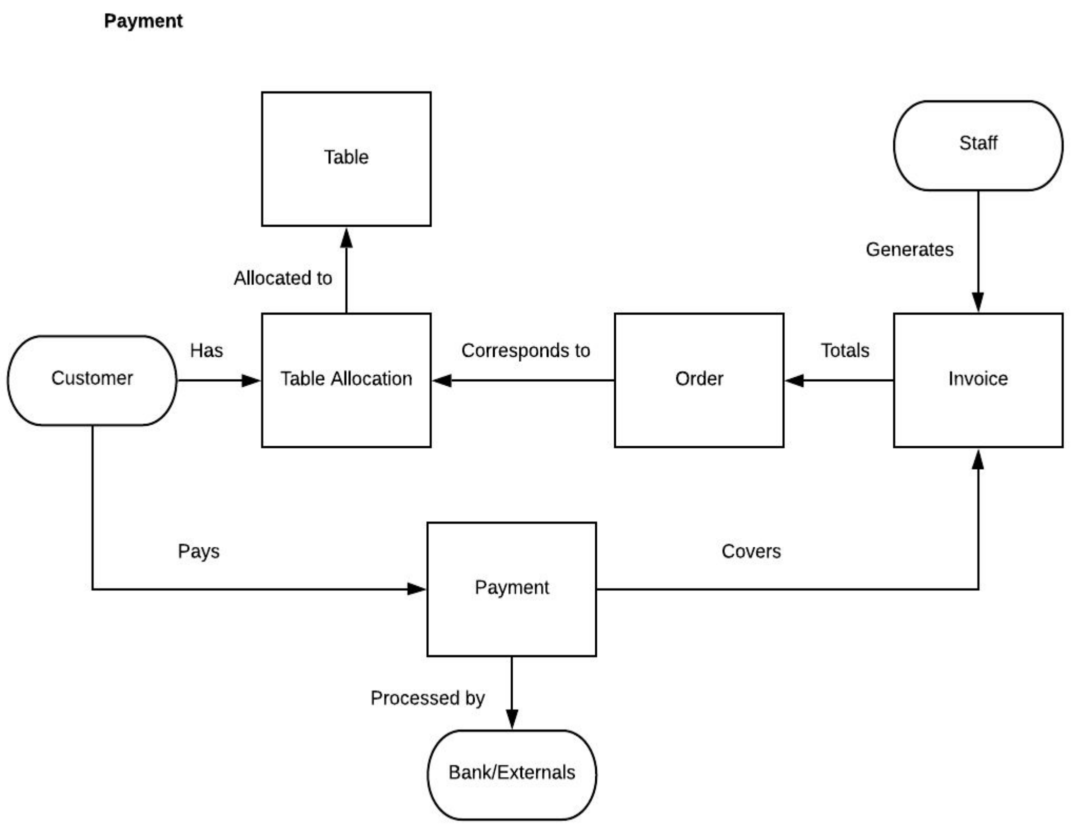


Fig. 3, Workflow 2

4.7.3. Ordering

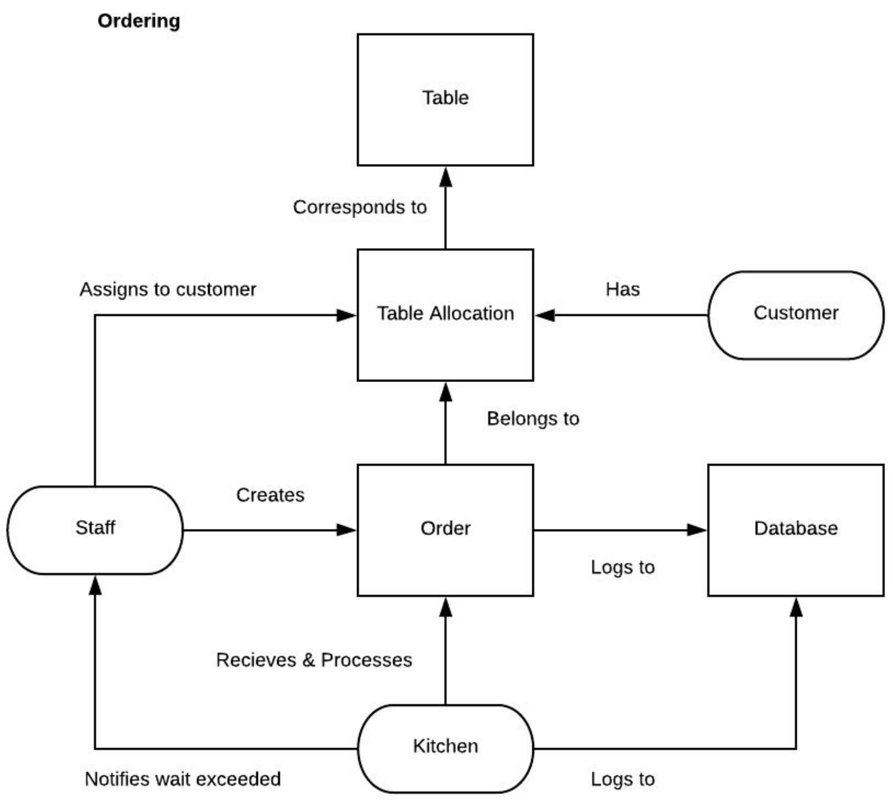


Fig. 4, Workflow 3

4.7.4. Online Reservation Creation

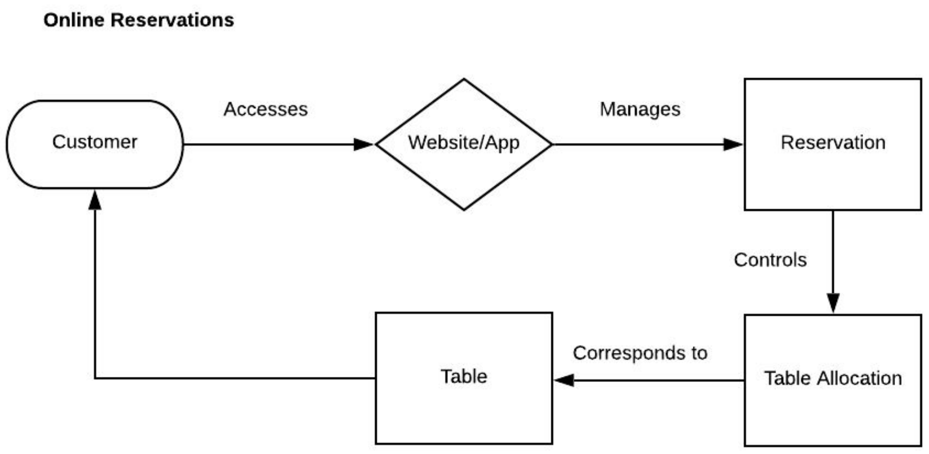


Fig. 5, Workflow 4

5. Task & Support

5.1.

Task	Make table reservation
Actor	Customer
Purpose	To reserve a table at the venue
Trigger/ Precondition	Customer decides to make a reservation
Frequency	High
Critical	Medium
Sub Tasks	**Example Solution**
1. Find website	(Standard web search)
2. Create reservation **Problem:** Cannot find reservation section on website	Have good UI design to make finding the section easy.
3. Enter requested reservation details	(Standard data entry)
4. Confirm reservation **Problem:** Customer does not have email	Allow for mobile/text confirmation.
Variants	
1a. Customer books over the phone 1b. Customer books in person	Staff members may process the booking.

5.2.

Task	Views menu through website
Actor	Customer
Purpose	See the cafe's menu
Trigger/ Precondition	Customer decides to look at menu
Frequency	High
Critical	Low
Sub Tasks	**Example Solution**
1. Find website	(Standard web search)
2. Find menu on the website	Ensure a good UI/UX to guide the user.

Problem: Cannot find menu on website	
3. Open menu to be viewable **Problem:** User does not have a PDF viewer	Embed a PDF viewer within the website itself to present the menu; Assume the user is using Chrome, Edge or Firefox and use the native PDF viewer in the browser;

5.3.

Task	Place order
Actor	Customer/Staff
Purpose	Place an order for the cafe to make
Trigger/ Precondition	Customer decides what they would like.
Frequency	Very High
Critical	Very High
Sub Tasks	**Example Solution**
1. Views items on menu	
2. Add menu items to the order	Click button on page
3. View items for order	
4. Confirm Order and send to kitchen	Click button on page
Variants	
3a. Change order 4a. Change of mind, cancel order	Items can be removed from the order. Orders can be cancelled.

5.4.

Task	Manage menu through website
Actor	Staff / Management
Purpose	Update menu
Trigger/ Precondition	Management want to change cafe menu
Frequency	Low
Critical	Moderate
Sub Tasks	**Example Solution**
1. Management login to management portal	Access admin login screen through website

2. Navigate to the “Update Menu” page	
3. Make menu changes	Be able to edit text on a menu document
4. Save menu changes	Button for user to click to save changes

5.5.

Task	Generate invoice for order on table
Actor	External Banking/Accounting System
Purpose	Generate invoice for an order
Trigger/ Precondition	Customer paid for order
Frequency	Very High
Critical	Very High
Sub Tasks	**Example Solution**
1. Get the customer order details	(Standard system data exchange)
2. Sends order data to statistics database	(See task: “Log order statistics”)
3. Create invoice based on the order details	System adds order detail to invoice template
4. Presents invoice to FOH and Customer **Problem:** User does not agree with the invoice, or there is a mistake.	Displays invoice on screen

5.6.

Task	Receive and action order
Actor	Customer/Staff/Kitchen
Purpose	Move order to kitchen to be made
Trigger/ Precondition	Completes order and confirms
Frequency	High
Critical	High
Sub Tasks	**Example Solution**
1. Order is confirmed	Button to confirm order
2. Present order to the kitchen	(Standard system data exchange)

3. Action the order	Order appears on screen in kitchen to be made

5.7.

Task	Make payment on order
Actor	Customer
Purpose	Pay for order made
Trigger/ Precondition	Customer contacts staff to pay
Frequency	High
Critical	High
Sub Tasks	**Example Solution**
1. View generated invoice	System displays invoice on a screen
2. Exchange money to pay for invoice	Through cash or card payment
3. Confirm and verify payment	A prompt comes on the screen asking whether complete payment has been made
Variants	
2a. Cash payment	Till is opened and cash is received by customer Change is given if required
2b. Card payment	Card payment is made on an EFTPOS machine. Transaction is sent to the external accounting system. Confirmation of payment is given by an external accounting system.

5.8.

Task	Generate receipt for payment
Actor	External Banking/Accounting System
Purpose	Give customer receipt for purchase
Trigger/ Precondition	Payment for order is made
Frequency	High
Critical	High
Sub Tasks	**Example Solution**
1. Order invoice is created	(See Task: “Generate invoice for order on table”)

2. Print receipt	Receipt for payment is printed.
Variants	
1a. No receipt required	Receipt printing is optional.

5.9.

Task	Allocate customer cover to table
Actor	FOH
Purpose	Allocate number of people on table
Trigger/ Precondition	Customer is seated at table
Frequency	High
Critical	Low
Sub Tasks	**Example Solution**
1. Declares number of customers on table	FOH clicks on table in app and fills out form fields
2. Allocates FOH staff member to service the table	FOH adds team member to service table through drop down list of active team members

5.10.

Task	Log order statistics
Actor	External Banking/Accounting System
Purpose	Log data for statistical analysis
Trigger/ Precondition	Receipt created for order
Frequency	High
Critical	Moderate
Sub Tasks	**Example Solution**
1. Receive invoice details	(See Task: “Generate invoice for order on table”)
2. Save invoice data between respective data tables	(Standard Database saving)
3. Send response that data has been logged	A response indicating that data has been logged.

5.11.

Task	Notify FOH staff on order wait exceeded
Actor	Kitchen
Purpose	Notify FOH on long order wait time
Trigger/ Precondition	Order timer exceeds certain time
Frequency	Moderate
Critical	High
Sub Tasks	**Example Solution**
1. Sends out notification of order time	Automatic when timer reaches certain time
2. Receive detail that order time has expired	Popup on the screen indicating the order waiting time.
3. Offer customer reimbursement	FOH offer some form of goods as reimbursement

6. Quality Attributes

6.1. Usability

Usability is a core attribute that will define the quality of this product. The software is to be used in a fast paced cafe thus human-understanding caused errors should be minimized.

The main aspects of usability in this scenario are: User Experience (UX), User Interface (UI) and Performance.

User Experience refers to the workflow and use case adherence of the software. It focuses on how the software will be used while ordering a meal, staff are taking payment or the kitchen is receiving orders. User experience meeting user expectations reduces the risk of errors.

User Interface refers to the graphical representation of the application interface. Good UI design promotes a pleasant experience and subtly highlights component importance hierarchy, reducing user frustration and improving usage fluidity.

Performance refers to the responsiveness of the application during use. In a fast paced, real-time environment low performance results in low customer satisfaction and loss of profit due to late and comped meals, and frustrated staff due to an unresponsive experience. Simple navigation events should be near-instant, while heavier processes should not take longer than 2 seconds to complete. Low frequency processes may exceed this limit.

6.2. Correctness

Correctness is another core attribute that must be considered throughout this project. The key areas where correctness must be adhered are:

1. The customer ordering processing, the kitchen receiving the order.

Incorrect orders result in angry customers, longer waits and lost profits while the correct orders are prepared.

2. The customer paying the correct amount for their order with an accurate invoice.

An incorrect invoice amount or payment process can result in over or under charging one or more customers, and undesirable audits.

3. Reservation timing and accuracy.

Clashing or null reservations may result in over or under booked tables, resulting in customer dissatisfaction or loss of profit.

These scenarios all affect both customer and company satisfaction and must be taken seriously.

6.3. Modifiability

The system must be designed with expansion in mind. The owners considered features outside those defined in the current scope which may be incorporated into a second phase. Thus to avoid unnecessary blow-out of project resources, the system must be built in a modifiable manner.

Scalability also falls under modifiability. It is not unreasonable to expect that in the future the cafe may expand and require the ability to handle more customers, tables, restaurant locations, menu selections etc. as it has on this occasion.

Interoperability also falls under this category somewhat. Future integration with common 3rd party entities such as MenuLog, UberEats is to be considered throughout the design phase.

6.4. Reliability

A full, non-recoverable failure during business hours would majorly hamper earning potential and dissatisfied customers. The system must function as intended for 98% of opening hours. In the case of operational failure, the system must be able to recover functionality within 3 minutes, without loss of important data such as invoices.

7. Possible Solutions

7.1. Solution 1 - Remote Infrastructure

7.1.1. Overview

Our recommended solution involves a web-app acting as a single source of operation. A number of tablet devices are dispersed throughout the cafe for staff use.

Upon arrival of a customer, a staff member opens and logs in to the app, which defaults to a cafe layout view. They select a table suitable for the customers, and assign a customer cover to the table.

When the customer would like to order, the staff selects the table to navigate to an order screen. They then input the menu items and confirm the order, notifying the tablets in the kitchen that a new item has arrived.

The kitchen staff, who have selected the 'Current Orders' view from the app menu, can then set the state of the order, and view other orders in the queue similarly. When too much time elapses without a state change, a front-of-house device notifies the staff members to attend to the waiting table.

Upon completion of their meal, the customer moves to the till, where the staff member selects their table and presses the invoice button, displaying an invoice on the screen. The customer can then pay with cash, or with the attached (or built in) EFTPOS utility on the tablet. The customer can decide whether to print a receipt or not. Either way, the interaction is recorded in the application's data-store.

The staff can then visit 'History' in the app menu to view order statistics and historical payment data.

The staff can also select 'Update Menu' in the app menu, and make the required changes which are pushed to the customer-facing website.

The customer website will contain three main pages:

1. An attractive homepage.
2. A menu page.
3. A reservation page.

The reservation page allows customers to leave their details and a preferred reservation time, which will notify the staff on a tablet, who can review and either accept or reject reservation requests under the 'Reservations' menu. The customer will then be notified of the result via their contact details.

7.1.2. Advantages

- Single point of development. Flow of data clearer.
- Less redundancy in processes (eg. changing menus)

7.1.3. Disadvantages

- Control of data deferred to external party. Reliant on availability.
- Reliant on connection to the internet. Service drops halt functionality.

7.2. Solution 2 - Local infrastructure

7.2.1. Overview

This solution further removes staff intervention and interaction with the customer moving towards a more complete technology approach.

The customer can access the cafe's menu through the website and can reserve a table at the cafe. Through entering the customers name, email, phone number, selecting a time and how many people will be on the table, they will then create a reservation and get a confirmation email for that reservation.

When a customer arrives at the cafe they are greeted by a FOH staff member who will show them to their table if they have a reservation or allocate them a table if they don't.

Each table will have an integrated iPad or Tablet device which will act as their menu. Here a customer can look through menu items and add items to their order. When they are happy with their order they can press 'Confirm' and this action will send the order to the kitchen to be made.

A timer is kept with each order to track the length of time it takes to get the order to the customer. If this timer elapses a certain time, FOH staff will be notified to compensate the customer for this wait time. When an order is made by the kitchen, they will notify staff to take the order to the customers table.

When the customer is finished, they will go to the front desk where they will make a payment. The system will ask whether the payment is cash or EFTPOS. It will process the payment and ask whether the customer wants a receipt.

Certain staff will be able to access the admin portal of the website. Here they can change the viewable menu on the website.

The staff can also select 'Update Menu' in the app menu, these changes are device only and will need to be done for each table device.

The staff can also visit 'History' in the app menu to view order statistics and historical payment data.

7.2.2. Pros:

- All internal processes are handled internally. Removes reliance on service providers for core business functionality.
- Customers handling orders themselves removes reliance on staff, a limited business resource.

7.2.3. Cons:

- Mix of tech, part online part local, increase in complexity.
- Requires investment in local infrastructure.
- Customers can't be trained to use software.