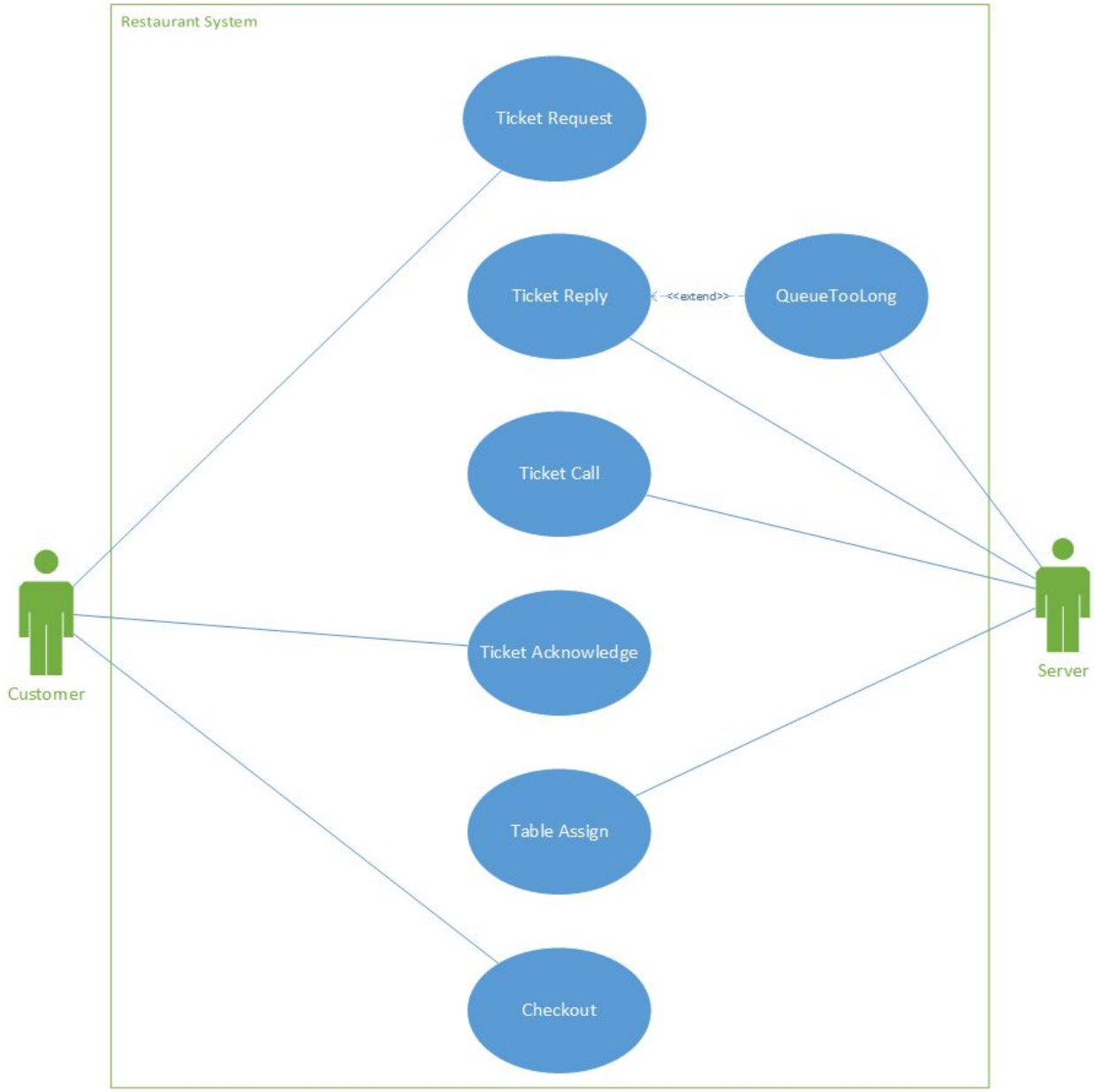
System Design Phase Documentation

17502985
16227018
14253100
14253801
14250926

Table of Contents

Use Case Diagram



Use Case Descriptions

	#1 **Ticket Request**
Goal in Context	Request a ticket for entry and seating into the restaurant.
Preconditions	The server is running.
Success End Condition	The server receives the request.
Failed End Condition	The server is not running or unable to receive the request.
Primary Actor	Customer.
Secondary Actor	Server.
Trigger	The customer presses the button on the ticket machine.
Description	1. The customer presses the button on the ticket machine. 2. The customer waits for a reply.
Extensions or Variations	

#2	**Ticket Reply**
Goal in Context	The server responds to one ticket request.
Preconditions	Customer makes a ticket request.
Success End Condition	Reply with a ticket number.
Failed End Condition	The queue is full and a ticket cannot be issued.
Primary Actor	Server
Secondary Actor	NA
Trigger	A new ticket request
Description	1. The Server receives a new ticket request. 2. The Server tries to find the best suitable queue for customer. 3. The Server replies with a ticketID.
Extensions or Variations	3a. No tickets available for the customer. • Return “Queue too long” exception.

#3	**Ticket Call**
Goal in Context	The server calls out the ticket number which has been deemed available for seating
Preconditions	A table is available for the customer.
Success End Condition	The server calls out the ticket number.
Failed End Condition	The server does not call out the ticket number
Primary Actor	Server
Secondary Actor	
Trigger	When a table is vacant
Description	1. The server detects a vacant table 2. The server calls out the next ticket number
Extensions or Variations	NA

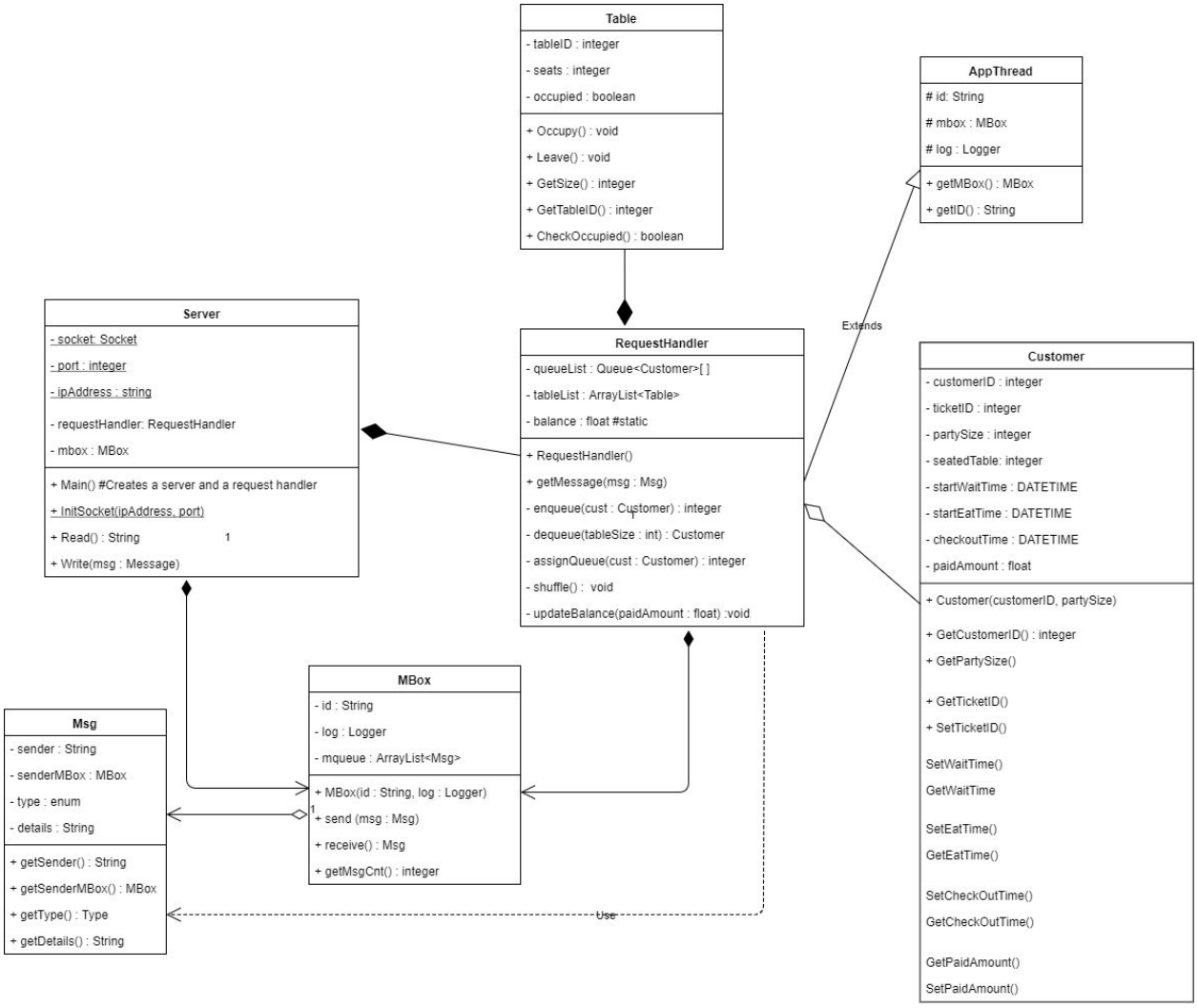
#4	**Ticket Acknowledge**
Goal in Context	The customer replies to the ticket call, acknowledging he is ready to be seated
Preconditions	The customer's ticket was called.
Success End Condition	The server receives the acknowledgement.
Failed End Condition	The customer does not respond to the ticket call.
Primary Actor	Customer
Secondary Actor	Server
Trigger	Server has sent a ticket call to customer
Description	1. Customer receives Ticket Call. 2. Customer sends message to server, acknowledging ticket call.
Extensions or Variations	2a. Customer does not acknowledge the ticket and causes a time-out.

#5	**Table Assign**
Goal in Context	Server assign a table to customer.
Preconditions	The server receives the ticket acknowledgement.
Success End Condition	Server sends the table assignment.
Failed End Condition	Server fails to send table assignment.
Primary Actor	Server
Secondary Actor	Customer
Trigger	Server receive an acknowledgement
Description	1. The server assign the available table to the corresponding customer.
Extensions or Variations	NA

#6	**Checkout**
Goal in Context	Customer checks out after eating.
Preconditions	1. Customer has been assigned a table by the server. 2. Customer has finished eating.
Success End Condition	The server receives payment from customer
Failed End Condition	The server does not receive payment from customer
Primary Actor	Customer
Secondary Actor	Server
Trigger	The customer calls to the server for checking out.
Description	1. The server receives a checkout from the customer 2. The server de-assigns the table
Extensions or Variations	NA

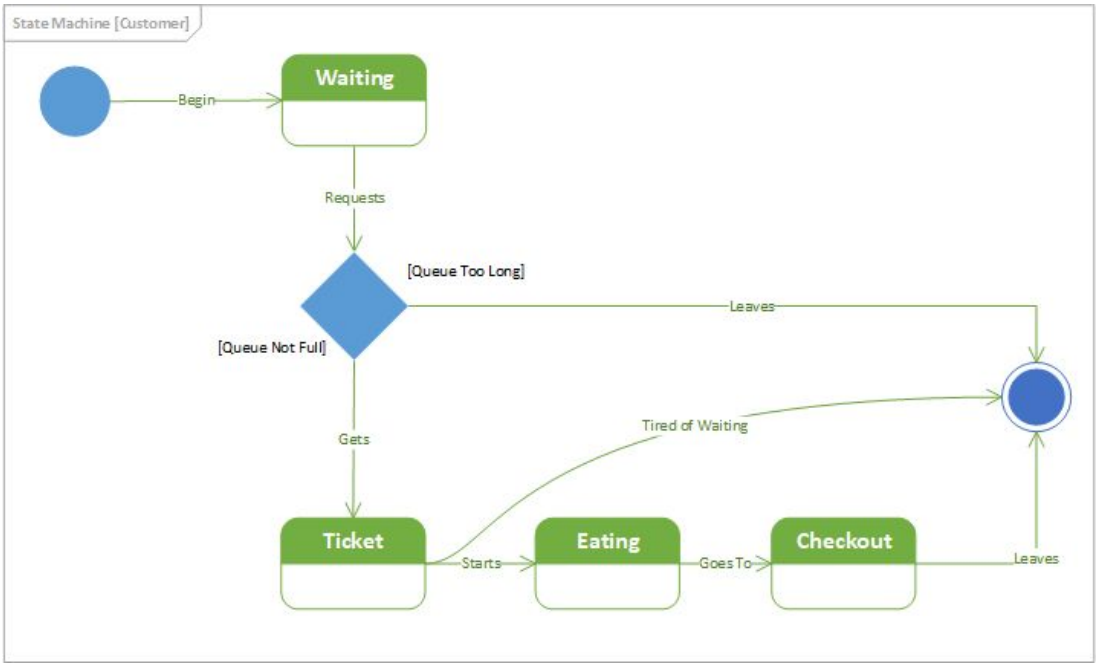
#7	**Queue Too Long**
Goal in Context	The server determines that the queue is too long and alerts the customer
Preconditions	The length of the queue exceeds the maximum number of customers that may be present in the queue
Success End Condition	The server notifies the user of the current state of the queue and refuses a ticket to the customer
Failed End Condition	Server does not notify the customer of the queue status and the client times out.
Primary Actor	Server
Secondary Actor	Customer
Trigger	When the customer requests a ticket
Description	1. Customer requests ticket 2. Server checks the length of the queue 3. Server determines that the queue is too long and refuses a ticket to the customer
Extensions or Variations	-

Class Diagram



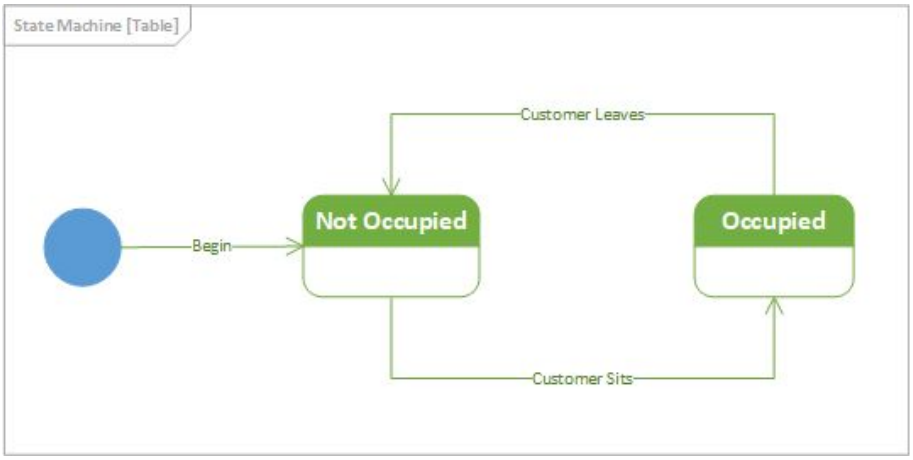
State Diagrams

Customer State Diagram



Description: Initially, a customer waits for a ticket. He can request a ticket, but if the Queue is Too Long, he will leave (Server sends **QueueTooLong**). Otherwise, he gets a ticket. If he's tired of waiting (he times out), he leaves. Otherwise, his next state is Eating. After Eating, he Checks Out. Then he leaves.

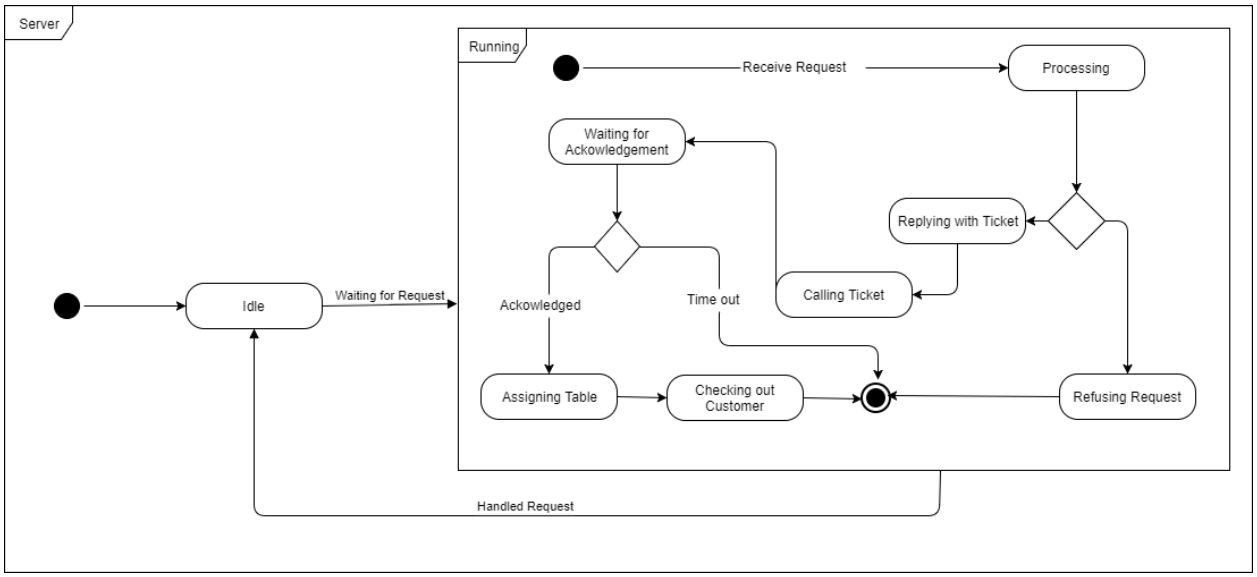
Table State Diagram



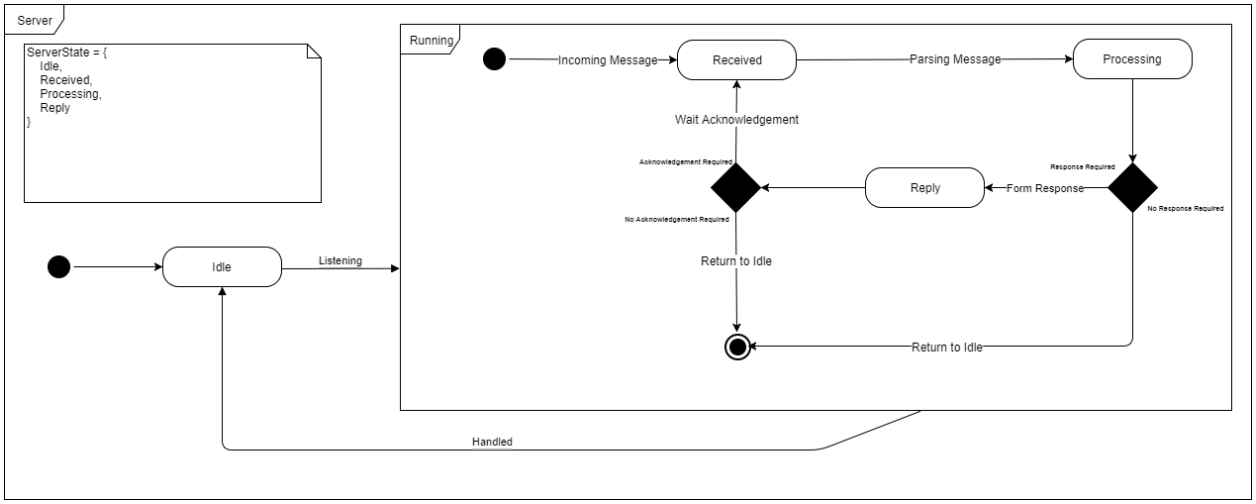
Description: A table can either be **Not Occupied** or **Occupied**. This is continuous. There is no end state as the states will infinitely loop over and over.

Server State Diagram:
We had initially arrived at an initial conclusion, but revised it two more times. See below Iteration 1, 2, and 3, in that order. We will provide a justification for the revision and description of the state diagram.

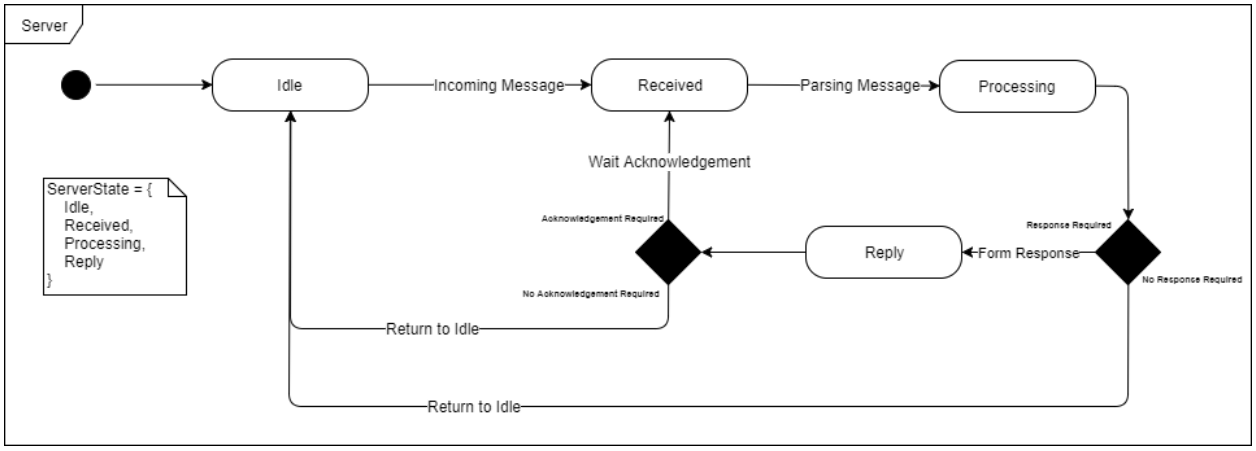
Iteration 1:



Iteration 2:



Iteration 3:



Justification for the iterations:

We felt there was a need to abstract and simplify some of the model. For example, “Replying with Ticket”, “Calling Ticket”, “Assigning Table” can all be modelled under a simple state called “Reply”. As a server has three fundemental states, which is consistent with all servers, namely Idle, Receiving, and Replying, we felt it important to model this setup in such a way too.

Further, the branch of “Acknowledged - Assigning Table - Checking out Customer - Finished”, can be modelled more simply with a conditional of “Acknowledgement required”, and if so, simply loop once again through the process.

To specify exactly what it is “Assigning Table … Checking Out Customer … “ could justifiably be modelled in the state diagram, but we felt that is more of implementation level - and as per our justification here, not necessarily a ***state*** of the system - the state would be “Reply”. This level of detail is justifiably not pertintent to the understanding and therefore the correctness of the state diagram, and should be reflected in the implementation instead.

That being said, iteration 1 and iteration 3 are equally as correct as eachother, with justification, since different people can model their systems however they wish as long as it’s justifiable. We ended up determining a more abstract solution is the better way - but someone else might prefer to go more in-depth which is okay too.

Iteration 3 Description:

The server will start and become **Idle**. Then it will receive a message and enter the **Received** state. It will parse it, to which it is in the **Processing** state. Then, if a reply is required, it enters the **Reply** state after forming a message to reply. Otherwise, it returns to **Idle.** After replying, it will determine if an acknowledgement is required. If so, it will **loop** once again. Otherwise, it will return to **Idle**.