Nanyang Business School

Bank Queue Management System

Technical Report

PBX BANK Co.

Team A10
G2200754J #CHIANG, JO-CHEN#
G2201047B #KAUR, ISHMITA#
G2200732H #LIU, YI-CHUN (Victoria)#
G2200305H #SHI ZHUOYA#
G2102278D #TSAI HSING FEN#

Contents

1. Introduction

1.1 Overview

This report discusses the technical aspects of design and implementation of PBX Bank Queue Management System. The report mentions the use of Node.js over Flask as the technology preference and provides information on the API used at the backend, the technical specifications required for implementation, and the data structure and algorithms used for the solution.

Overall, the report summarizes the technologies, API, data structure and algorithms that are being considered for the implementation of the PBX Bank Queue Management System. It was important to carefully review and consider this information in order to ensure that the solution is implemented in an effective and efficient manner.

1.2 Scope Definition

Based on the business requirements, the scope identified for PBX Bank Queue Management System is as follows:

(i) Provision for customers to register and obtain a 'Queue Number' in below ways:

a. Website – Customer can register remotely in advance at a branch of choice and for a preferred date and time. An email will be triggered on the entered email ID upon successful registration for records.
b. Walk-in – Customer present at a bank branch, can access the kiosk and register to obtain a 'Queue Number'.

(ii) Provision to support the counter staff:
The counter staff will serve all the queues. Each counter will have their own UI to communicate with the backend queue management system. The staff at the counter can do the below:

a. call for the customer to the counter
b. 'hold' certain queue number as 'missed queue number'
c. 'missed queue number' to 're-schedule' i.e., to be re-instated in the queue.

(iii) Support customer relationship officer (CRO)
CRO will have his/her own UI (different from counters) to communicate with the backend queue management system.

a. to view the entire queue list for the respective categories
b. to stop/re-initiate the taking of queue number for either queue category.

(iv) Counter Display
There will be a screen at the branch to display the ongoing and queues being served by various counters.

Below points are out of scope:

1. No logins or authentication required for any functionality.
2. No dedicated database will be built. The system will function on the server database.

2. Technology Specifications

2.1 Development Environment

	Programming Language
Front End	HTML/ JavaScript/ CSS
Back End	Node JS

The design of a web page is created using HTML and CSS, while JavaScript and Node.js serve as the connection between the front-end and back-end. The dist directory in the project folder contains the compiled CSS and JavaScript files for beautifying the user interface. However, access to the styles from the W3-school CSS package requires an internet connection.

In the Node.js project, the node_modules directory is necessary for storing the required dependencies and packages. The Node Package Manager (npm) is used to manage these dependencies, making it easy to install and update packages from the npm registry.

To begin, several packages from the npm registry need to be installed. npm helps to manage these packages, including libraries, modules, and more, to ensure that the project has everything it needs to run smoothly.

2.2 Installation Requirement

Following is the steps of installing the required packages.

Steps	Installation Instructions (in Terminal)	Purpose
1	npm init	Set up project to use npm, a package.json file will be create In package.json file, modify the following codes: “Scripts”: { "dev": "nodemon server.js"}
2	npm install	For the use of node package manager
3	npm install express	For building APIs in Node.js framework
4	npm install path	For working with file and directory paths
5	npm install nodemon	For automatically restarts the Node.js application
6	npm install cors	Cross-Origin Resource Sharing, allows a server to respond to requests from a different domain
7	npm install nodemailer	For sending email
8	npm install body-parser	For parsing incoming request bodies
9	npm install ejs	ejs is a template engine for node.js, allowing us to embed JavaScript code into HTML templates.

Following is the steps on initiating the system:

Steps	Instruction	Purpose
1	const PORT = process.env.PORT || 9000;	Set up port number
2	app.listen(PORT,console.log(`Server started on port ${PORT}`));	Set up server
3	npm run devStart	Run the application
4	Use FireFox browser to view the webpage	To prevent asynchronicity issue on giving out queue number

2.3 Back-End Language Preference – Node JS

For the assumption that the system will serve hundreds of thousands of customers per day, it is important for our queue system to operate smoothly without interrupting. Thus, we use Node.js instead of flask in python to complete the queue system, since it is rapid, expandable, and efficient. For applications that need real-time processing and queue management, Node.js is a good option. In our queue system, we apply an in-memory queue, where messages are kept in the Node.js process memory. Since they don't need to write to or retrieve from a database, in-memory queues often run faster than persistent queues. They are therefore well adapted for handling and processing huge amounts of users in real-time.

In addition, JavaScript is a very flexible language that can be used to create both the client-side and server-side of an application, making it a full-stack language. JavaScript can be used to create dynamic, interactive web apps on the client side, giving web pages extra features and behavior. JavaScript can also be used to create server-side applications on frameworks like Node.js.

Although we have used Python in the past, we think that employing JavaScript will provide us with a different and challenging experience. We will learn new ideas and concepts by applying JavaScript to our project, which will improve our understanding of programming overall. In conclusion, we think that adopting JavaScript into our project will be a valuable learning opportunity that will increase our knowledge of programming and our programming skills.

3. API Description

3.1 Customer Registration – Online Registration

The user will register on the bank website by providing some details, for which the system will generate and assign a queue number. Details entered by the customer are to be parsed to server as it shall be further used to determine the priority of the customer.

Below are the APIs to be developed in order to perform this business operation:

No	API Name	Method	Functionality	Input	Output
1.	/onlineBooking	GET	Site page for website registration	NA	Load the html page: homepage.html
2.	/newEntryWeb	GET	generate client number automatically	NA	Queue number
3.	/send-phonenumber	POST	Get phone number to identify the category	Phone number	NA
4.	/send-slot-details	POST	Get booking details from the front end	Slot, branch	NA
5.	/send-email	POST	send confirmation email to the client	Email ID	Acknowledgement message for Pass/ Fail

3.2 Customer Registration – Walk In

The user will walk in the branch to register by providing phone number, for which the system will generate and assign a queue number. Phone number entered by the customer are to be parsed to server as it shall be further used to determine the priority of the customer.

Below are the APIs to be developed in order to perform this business operation:

No	API Name	Method	Functionality	Input	Output
1.	/walk_in/Jurong	GET	Site page for walk-in registration at Jurong	NA	Load the html page: walkin.ejs
2.	/walk_in/Orchard	GET	Site page for walk-in registration at Orchard	NA	Load the html page: walkin.ejs
3.	/walk_in/branch	GET	Get branch name	NA	Branch
4.	/walk_in/submit	GET	Get phone number, identify customer level and branch, registration status, and generate client number if success	Phone number, branch, queue stop variable	Load the html page: walkinsubmit.ejs. , client number, arrive time, registration status
5.	/walk_in/sendbranch	POST	Get branch name	Branch	NA

3.3 Counter Queue Management

Our system supports multiple branches and multiple counters, so the counter employee first needs to select his or her specific counter. Then the employee can call for next client to the given counter to get service. The client who doesn't show up will be hold and get rescheduled by the CRO officer later upon arriving.
Below are the APIs for this business operation:

No.	API Name	Method	Functionality	Input	Output
1.	/selectcounter	GET	Site page for branch and counter selection	N.A.	Load the html page: selectCounter.ejs
2.	/counter	GET	Site page for counter operation	The selected branch name and counter number will be shown in the url as parameters	Load the html page: counter_function.html
3.	/counter/ sendbranch_counter	POST	Each time an action is performed, check the branch name and counter number to facilitate data storage in the backend	The selected branch name and counter number	N.A.
4.	/counter/nextClient	GET	To call for next client who ranked first in the waiting list and update the waiting list	N.A.	The number of the next client
5.	/counter/ clearingNumber	POST	Complete the service of current client and remove his number from the display screen	Branch name, counter number, client number	N.A.
6.	/counter/ holdingNumber	POST	Hold the number of missing client and push it into the missing list which will also be displayed on the screen	Branch name, counter number, client number	N.A.

3.4 Customer Relationship Officer Management

Customer relationship officer is able to perform three action in the system—Requeue Management, Queue Category Management, and Queue Display.

I. Requeue Management
When a customer fails to appear at the counter when their queue number is called, they will be placed on a missing queue list. When the customer returns to the bank, they can inform the CRO desk of their presence. The officer will then manually input the customer's queue number and reinsert them into the current queue line.

II. Queue Category Management
Initially, all customer categories are open for receiving customers. The CRO can assess the bank's crowding situation and then temporarily stop receiving for a particular customer category to ensure safety and prevent overcrowding.

III. Queue Display
The CRO officer is able to select the desired customer category in order to display the current list of customers in line.

Below are the APIs to be developed in order to perform this business operation:

No.	API Name	Method	Functionality	Input	Output
1.	/selectcro	GET	Site page for branch and counter selection	N.A.	Load html page: cro_select.ejs
2.	/cro	GET	Site page for CRO operation	N.A.	Load html page: cro_select.ejs
3.	/cro/getInfo	GET	Get the number of waiting clients of each category	N.A.	Queue quantity in each category
4.	/cro/sendbranch	POST	Each time an action is performed, check the branch name and counter number to facilitate data storage in the backend	The selected branch name	N.A.
5.	/cro/refresh	GET	Refresh the number of waiting clients of each category		Queue quantity in each category
6.	/cro/requeue/number	GET	Requeue the on-hold customer queue number	Requeued queue number	N.A.
7.	/cro/d_stop	GET	Change the service status of diamond category	Diamond category customer receiving status	N.A.
8.	/cro/g_stop	GET	Change the service status of gold category	Gold category customer receiving status	N.A.
9.	/cro/s_stop	GET	Change the service status of silver category	Silver category customer receiving status	N.A.

10.	/cro/n_stop	GET	Change the service status of normal category	Normal category customer receiving status	N.A.
11.	/cro/queue/detail	GET	Get current waiting list queue number	The selected category	Load html page: cro_display.ejs; Current list of queue no. in line

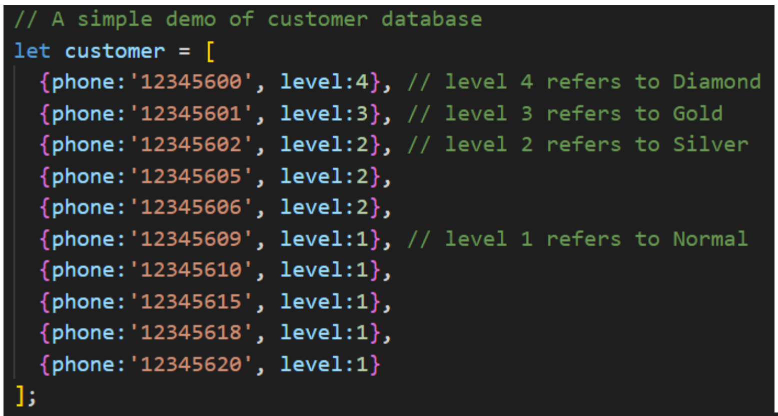
3.5 Display Screen

Each branch has its display page that will show the queue number each counter is serving, the waiting list, and the missing list. The display page will update every second to catch the latest information.
The client who makes the reservation online should check in when they arrive at the branch, once they check in successfully, their queue number will be append to the waiting list.
The following are the APIs for the operations:

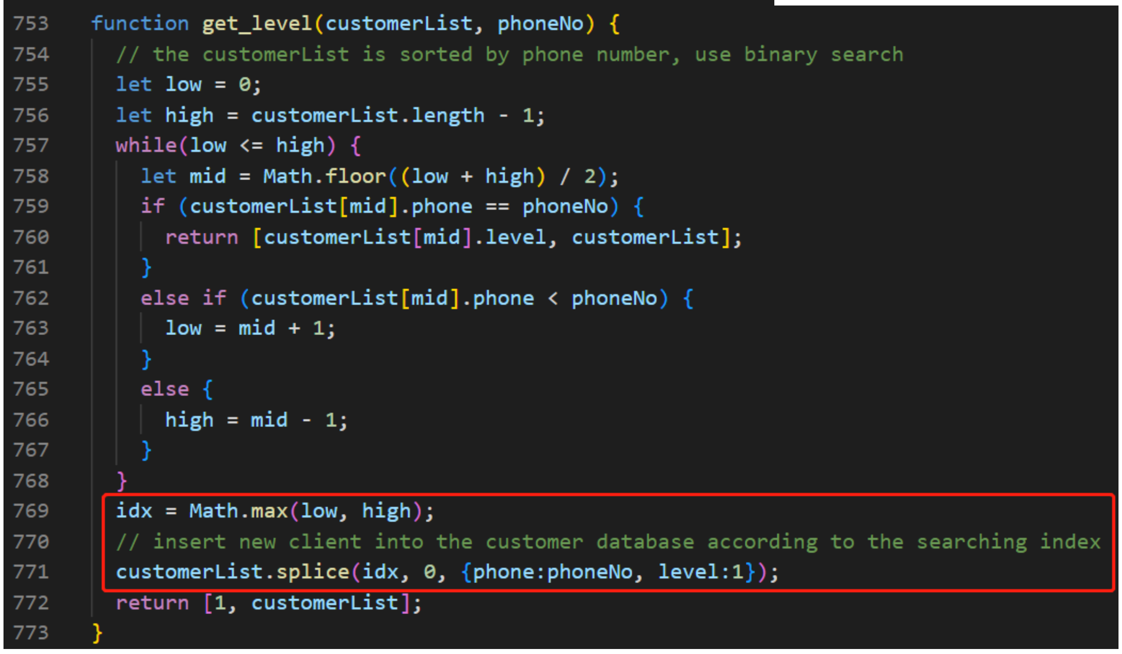
Sr. No	API Name	Method	Functionality	Input	Output
1.	/display/Jurong	GET	Display the queue number that each counter is serving, current waiting list for Jurong branch, and missing list for Jurong branch.	NA	Branch name, client number that each counter is serving, waiting list for Jurong branch, and missing list for Jurong branch.
2.	/display/Orchard	GET	Display the queue number that each counter is serving, current waiting list for Orchard branch, and missing list for Orchard branch.	NA	Branch name, client number that each counter is serving, waiting list for Orchard branch, and missing list for Orchard branch.
3.	/sign_in/Jurong	GET	For the clients who reserved online to check in when they arrive at the branch.	The queue number the client got online.	NA
4.	/sign_in/Orchard				
5.	/get-number	GET	Display whether the client successfully check in or not after they enter their queue number in the sign in page.	NA	Display whether the client check in successfully.

4. Data Structure and Algorithm

The primary key of the customer database is the phone number. No matter the client walks in or books online, our system is able to identify his level by searching the database. This database will always be sorted by phone-number so binary search is applied here. Besides, new clients will be inserted into the database accordingly based on the final index of binary search and regarded as level 1 (normal).



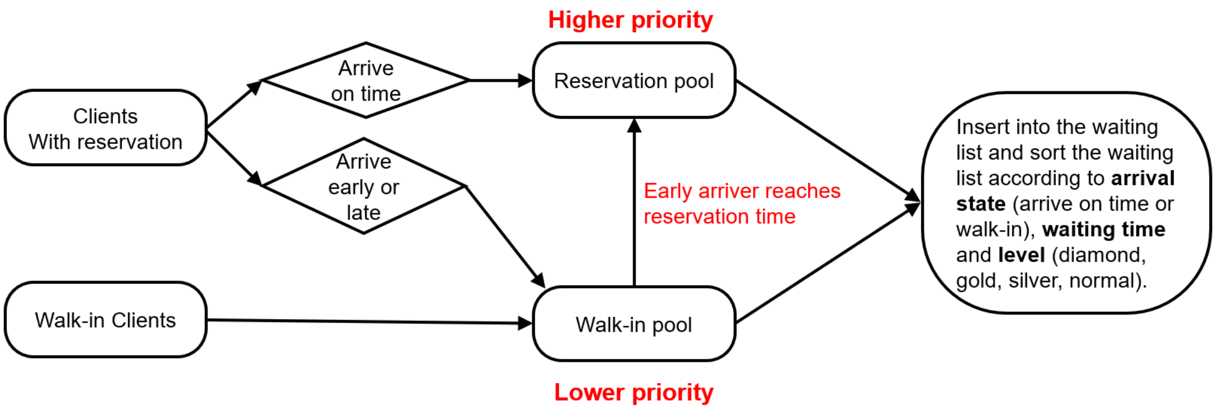
```
// A simple demo of customer database
let customer = [
  {phone:'12345600', level:4}, // level 4 refers to Diamond
  {phone:'12345601', level:3}, // level 3 refers to Gold
  {phone:'12345602', level:2}, // level 2 refers to Silver
  {phone:'12345605', level:2},
  {phone:'12345606', level:2},
  {phone:'12345609', level:1}, // level 1 refers to Normal
  {phone:'12345610', level:1},
  {phone:'12345615', level:1},
  {phone:'12345618', level:1},
  {phone:'12345620', level:1}
];
```



```
function get_level(customerList, phoneNo) {
  // the customerList is sorted by phone number, use binary search
  let low = 0;
  let high = customerList.length - 1;
  while(low <= high) {
    let mid = Math.floor((low + high) / 2);
    if (customerList[mid].phone == phoneNo) {
      return [customerList[mid].level, customerList];
    }
    else if (customerList[mid].phone < phoneNo) {
      low = mid + 1;
    }
    else {
      high = mid - 1;
    }
  }
  idx = Math.max(low, high);
  // insert new client into the customer database according to the searching index
  customerList.splice(idx, 0, {phone:phoneNo, level:1});
  return [1, customerList];
}
```

The priority of waiting clients is shown in the flow chart below. Since we encourage our clients to book in advance, clients with reservations will always get higher priority than walk-in customers. If clients with reservations arrive late, his priority will be cancelled, and he will be treated as walk-in clients because we appreciate punctuality. If clients with reservations arrive early and there are other customers waiting before him, his priority will not be activated until the start time of his reservation.

All the clients will be sorted according to arrival state (arrive on time or walk-in), waiting time and level (diamond, gold, silver, normal) dynamically. The whole logic is applicable in terms of rescheduling missing clients.


Higher priority
Clients With reservation
Arrive on time
Arrive early or late
Reservation pool
Early arriver reaches reservation time
Insert into the waiting list and sort the waiting list according to **arrival state** (arrive on time or walk-in), **waiting time** and **level** (diamond, gold, silver, normal).
Walk-in Clients
Walk-in pool
Lower priority

The information of waiting clients is also stored in the backend in the format of embedded array. The key factors used for sorting are demonstrated below.

No.: To make things clear, number of reserved clients starts with ‘A’ and number of walk-in clients starts with ‘B’.

Arrive_on_time: This feature will always be marked as 0 for walk-in clients. If reserved clients arrive on time, this feature will be marked as 1. If the reserved client arrives early, this feature will be marked as 0 at first and changed to 1 when it reaches his reservation time. If the reserved client arrives late, this feature will be marked as 0.
Exceed_maximum_waiting_time: This feature is set to be 0 by default. The maximum waiting time is set to be 15 minutes for diamond and gold clients and set to be 30 minutes for silver and normal clients. Once the waiting time of some client exceeds the maximum value, this feature will be marked as 1.
Level: Other things being equal, waiting clients are sorted from diamond to gold, silver and normal.

No.	Arrive_on_time	Exceed_maximum_waiting_time	Level
A003	1	1	Normal
B001	0	1	Gold