Completed on,
11.09.2024

DATA STRUCTURES
MINI-PROJECT

PRESENTED BY,

Dharshan T
Shalini S R
Syed S

Problem Statement

In a modern ticketing environment, efficient management of ticket reservations across various categories is crucial for optimizing customer satisfaction and operational efficiency. Businesses that offer ticketed events, travel services, or entertainment experiences often deal with a range of ticket categories, each with its own pricing, availability, and demand patterns. To address these complexities, a sophisticated ticket booking system is needed that can handle different ticket categories through a queuing mechanism, ensuring fair and orderly processing of booking requests.

METHODOLOGY

FRONT END

The front-end is responsible for creating the user interface where customers interact with the ticket booking system. This includes designing and implementing web pages or mobile screens where users can view available ticket categories, select options, and make bookings.

QUEUE

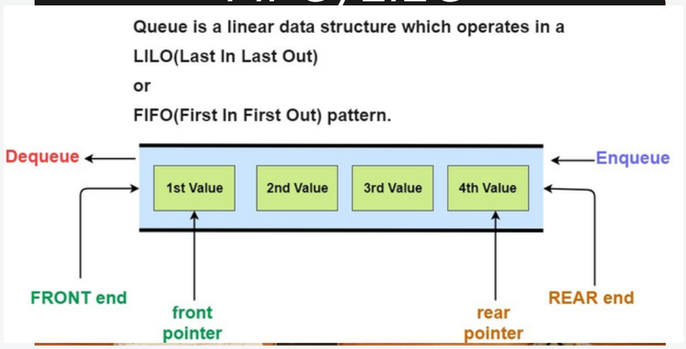
- Queues are used to manage and process booking requests in an orderly manner. In the context of a ticket booking system, each ticket category may have its own queue to handle incoming booking requests. By using queues, the system ensures that requests are processed in the order they are received (FIFO: First In, First Out), which is crucial for maintaining fairness and avoiding conflicts in high-demand scenarios.

BACK END

The back-end is responsible for handling requests made by the front-end. This includes processing booking requests, calculating prices, managing ticket inventory, and enforcing business rules.It manages transactions, including handling booking confirmations, cancellations, and any adjustments required in the ticket inventory.

QUEUE DATA STRUCTURE

FIFO/LILO



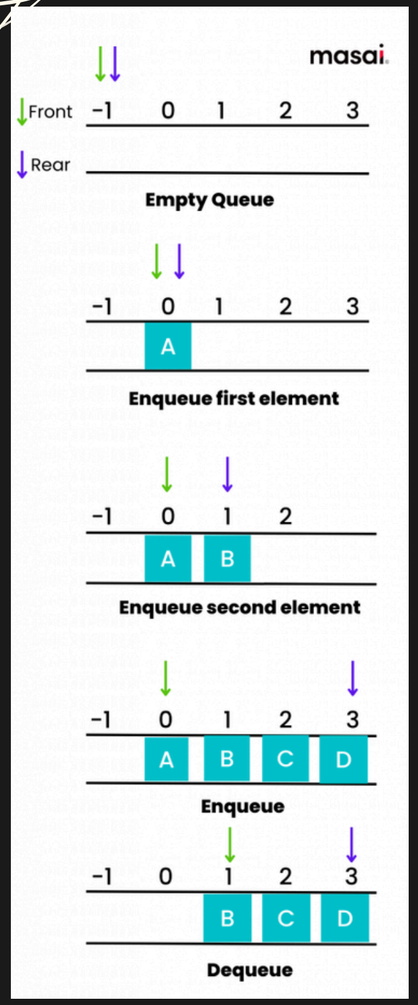
- **Request Management**: Queues are used to manage and process booking requests in an orderly manner. In the context of a ticket booking system, each ticket category may have its own queue to handle incoming booking requests.

Fairness and Order: By using queues, the system ensures that requests are processed in the order they are received (FIFO: First In, First Out), which is crucial for maintaining fairness and avoiding conflicts in high-demand scenarios.

- **Simple Queue**: A basic FIFO queue where requests are added to the end and processed from the front.

Priority Queue: A more advanced type of queue where requests can have different priorities, and those with higher priority are processed before others.

ENQUEUE

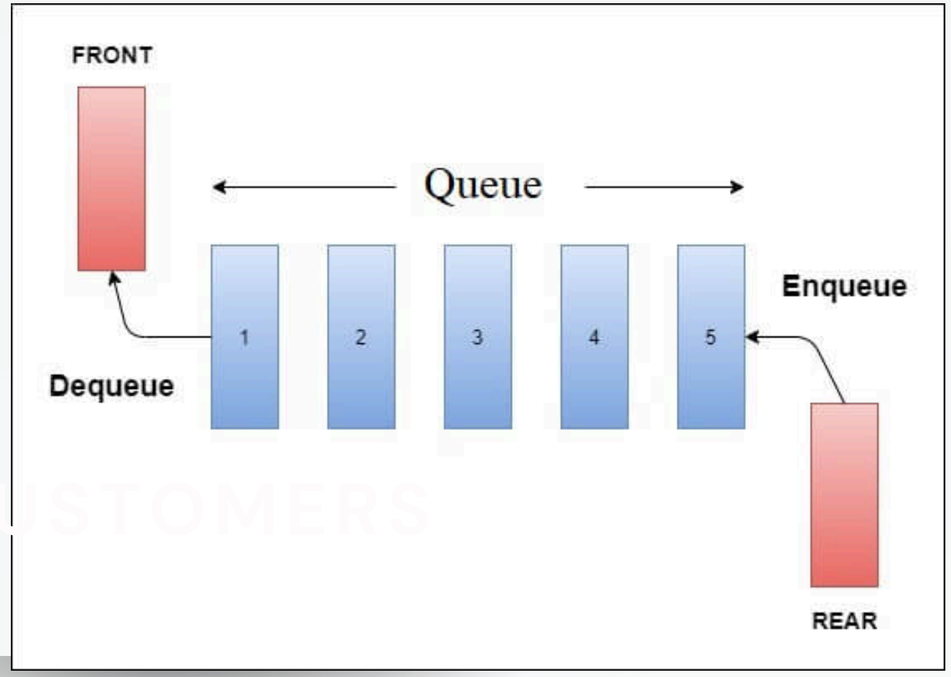


Enqueue is the operation used to add an element to the back of the queue. When you enqueue an element, it gets added to the end of the queue, and the order of elements is preserved.

DEQUEUE

Dequeue is the operation used to remove an element from the front of the queue. When you dequeue an element, it is removed from the front, and the remaining elements shift forward to fill the gap. The element removed is the one that has been in the queue the longest.

Example: Using the queue [5, 10, 15], if you perform the following dequeue operation: Dequeue(): The element 5 is removed, and the queue becomes [10, 15]



CODE

```
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <STRING.H>

#DEFINE MAX 10

// USER STRUCTURE
STRUCT USER {
 CHAR NAME[50];
 CHAR PHONE[15];
};

// QUEUE STRUCTURE
STRUCT QUEUE {
STRUCT USER USERS[MAX];
 INT FRONT, REAR;
};

// INITIALIZE QUEUE
VOID INITQUEUE(STRUCT QUEUE* Q) {
 Q->FRONT = -1;
 Q->REAR = -1;
}

// CHECK IF QUEUE IS FULL
INT ISFULL(STRUCT QUEUE* Q) {
 RETURN Q->REAR == MAX - 1;
}

// CHECK IF QUEUE IS EMPTY
INT ISEMPTY(STRUCT QUEUE* Q) {
 RETURN Q->FRONT == -1;
}

// ADD USER TO QUEUE
VOID ENQUEUE(STRUCT QUEUE* Q, STRUCT
 USER USER) {
 IF (ISFULL(Q)) {
PRINTF("TODAY'S TICKET IS FULL. MEET
 YOU NEXT DAY.\N");
 } ELSE {
 IF (Q->FRONT == -1)
 Q->FRONT = 0;
 Q->REAR++;
 Q->USERS[Q->REAR] = USER;
PRINTF("TICKET FOR %S (%S) BOOKED
 SUCCESSFULLY.\N", USER.NAME,
 USER.PHONE);
 }
}// REMOVE A USER BY SEARCHING NAME AND
 PHONE NUMBER
VOID CANCELTICKET(STRUCT QUEUE* Q, CHAR*
 NAME, CHAR* PHONE) {
 IF (ISEMPTY(Q)) {
PRINTF("NO TICKETS BOOKED.\N");
 RETURN;
 }

 INT FOUND = 0;
FOR (INT I = Q->FRONT; I <= Q->REAR; I++) {
IF (STRCMP(Q->USERS[I].NAME, NAME) == 0 &&
STRCMP(Q->USERS[I].PHONE, PHONE) == 0) {
 FOUND = 1; // SHIFT ELEMENTS TO REMOVE
 THE TICKET
 FOR (INT J = I; J < Q->REAR; J++) {
 Q->USERS[J] = Q->USERS[J + 1];
 }
 Q->REAR--;
PRINTF("TICKET FOR %S (%S) CANCELLED
SUCCESSFULLY.\N", NAME, PHONE);
 IF (Q->REAR < Q->FRONT) {
 Q->FRONT = Q->REAR = -1;
 IF (!FOUND) {
PRINTF("TICKET NOT FOUND.\N");
 }
}

// DISPLAY USERS IN THE QUEUE
VOID DISPLAYQUEUE(STRUCT QUEUE* Q) {
 IF (ISEMPTY(Q)) {
PRINTF("NO TICKETS BOOKED YET.\N");
 } ELSE {
PRINTF("TICKETS BOOKED:\N");
FOR (INT I = Q->FRONT; I <= Q->REAR; I++) {
PRINTF("%D. NAME: %S, PHONE: %S\N", I + 1, Q-
>USERS[I].NAME, Q->USERS[I].PHONE);
 }
 }
}
INT MAIN() {
STRUCT QUEUE Q;
INITQUEUE(&Q);

STRUCT USER USER;
CHAR NAME[50], PHONE[15];
INT CHOICE;

DO {
PRINTF("\N1. BOOK TICKET\N2. DISPLAY TICKETS\N3.
CANCEL TICKET\N4. EXIT\NENTER CHOICE: ");

SCANF("%D", &CHOICE);
```

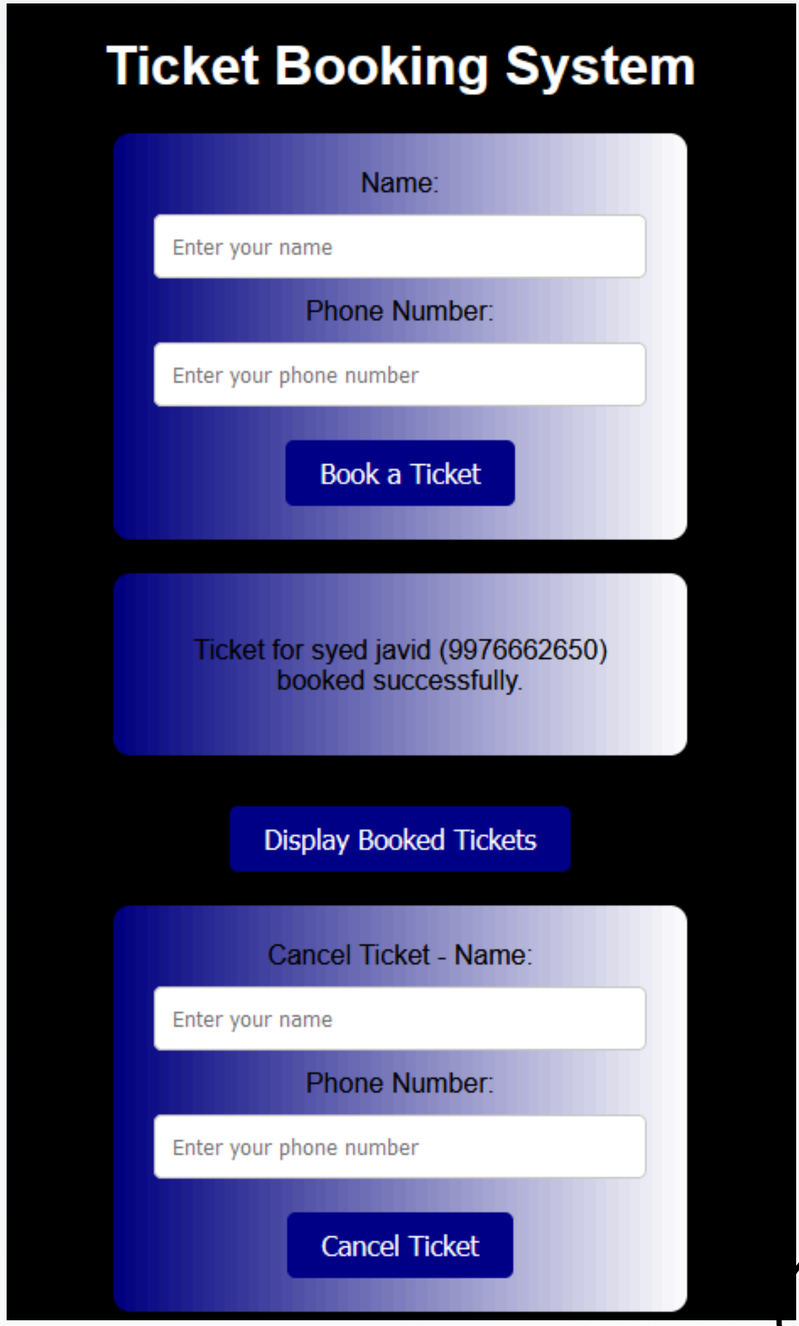

11.09.2024

CODE

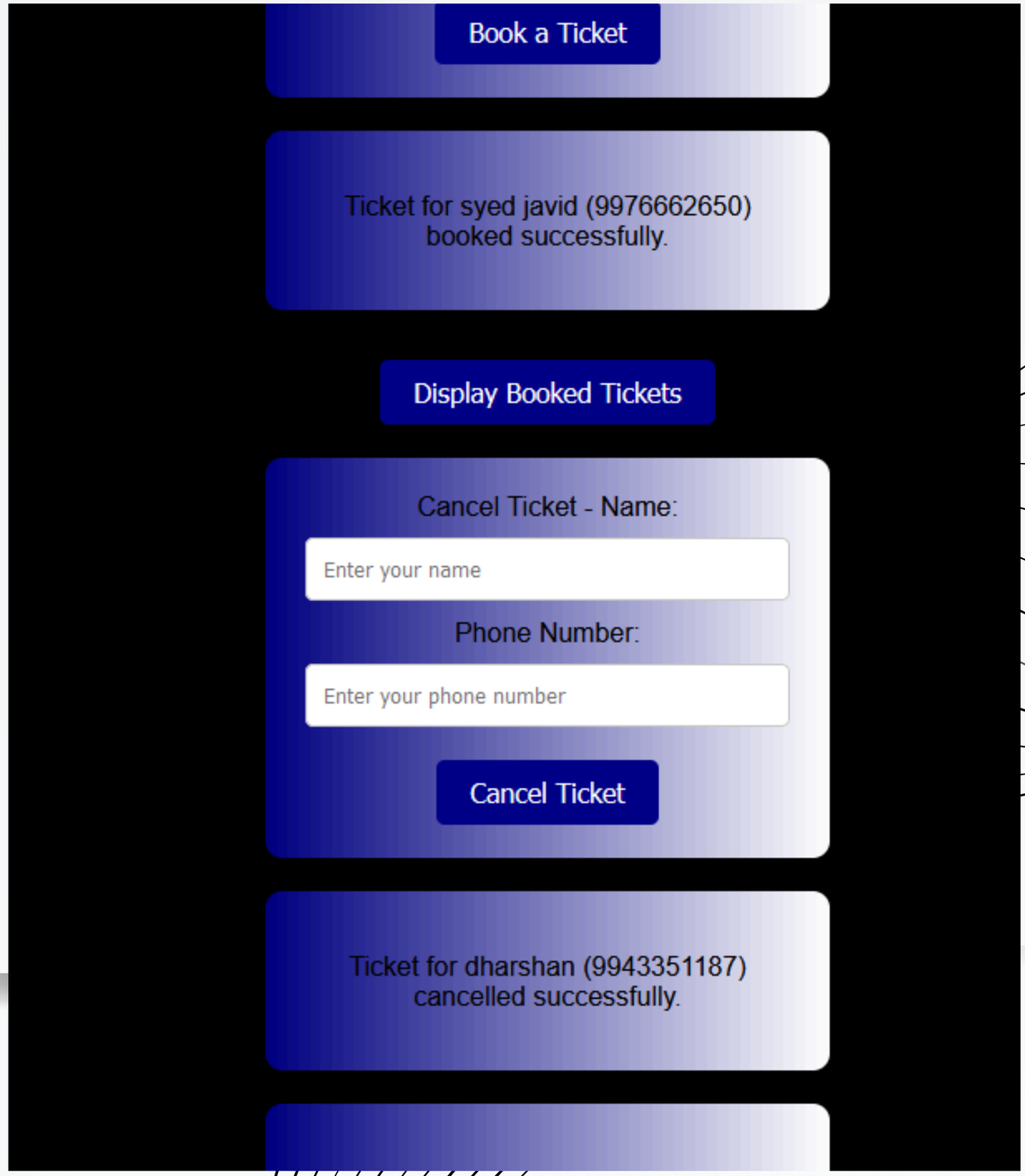
```
SWITCH (CHOICE) {
        CASE 1:
    PRINTF("ENTER NAME: ");
    SCANF("%S", USER.NAME);
PRINTF("ENTER PHONE NUMBER: ");
    SCANF("%S", USER.PHONE);
    ENQUEUE(&Q, USER);
        BREAK;
        CASE 2:
    DISPLAYQUEUE(&Q);
        BREAK;
        CASE 3:
    PRINTF("ENTER NAME: ");
    SCANF("%S", NAME);
PRINTF("ENTER PHONE NUMBER: ");
    SCANF("%S", PHONE);
CANCELTICKET(&Q, NAME, PHONE);
        BREAK;
        CASE 4:
    PRINTF("EXITING...\N");
        BREAK;
        DEFAULT:
    PRINTF("INVALID CHOICE!\N");
            }
    } WHILE (CHOICE != 4);

        RETURN 0;
            }
```

OUTPUT



OUTPUT


Book a Ticket
Ticket for syed javid (9976662650) booked successfully.
Display Booked Tickets
Cancel Ticket - Name:
Enter your name
Phone Number:
Enter your phone number
Cancel Ticket
Ticket for dharshan (9943351187) cancelled successfully.

THANK YOU