1.Introduction:

The Railway Waiting List Management System is designed to efficiently manage and monitor the waiting list of passengers for train reservations. Its primary goal is to streamline the process of allocating seats or berths to passengers who are on the waiting list, ensuring maximum occupancy and minimizing cancellations.

Features:

1.Online Reservation: Passengers can book their train tickets online through the railway's official website or mobile app, indicating their preferred date, time, and class of travel.

2.Real-time Updates: The system provides real-time updates on seat availability and waiting list status. Passengers can check their current position on the waiting list at any time.

3.Automated Seat Allocation: As cancellations or seat releases occur, the system automatically allocates seats to passengers on the waiting list based on predefined criteria such as priority, booking time, and class preferences.

4.Priority Allocation: Passengers may be prioritized based on factors such as medical emergencies, seniority, or special needs.

5.Cancellation Management: Passengers who have booked tickets but wish to cancel them can do so online, freeing up seats for others on the waiting list.

6.Communication: The system sends notifications to passengers via email or SMS regarding their booking status, including confirmation of reservation, waitlist movement, or cancellation.

7.Analytics and Reporting: Railway authorities can generate reports and analyse data on booking patterns, waiting list trends, and seat utilization to optimize operations and capacity planning.

Benefits:

1.Efficiency: The system reduces manual intervention and automates the allocation process, leading to faster and more accurate seat assignments.

2.Customer Satisfaction: Passengers have transparency regarding their booking status and are promptly informed of any changes, enhancing their overall experience.

3.Optimized Capacity Utilization: By efficiently managing the waiting list, the system helps maximize revenue by ensuring that available seats are occupied and minimizing the number of vacant berths.

4.Data-driven Decision Making: Railway authorities can leverage insights from the system's analytics to make informed decisions regarding scheduling, pricing, and resource allocation.

2. Project Description:

The Railway Waiting List Management System is a project designed to streamline the process of managing waiting lists for train reservations. The system aims to automate the allocation of seats or berths to passengers on the waiting list, ensuring optimal occupancy and minimizing cancellations and taking care of special reservations. It facilitates efficient communication with passengers regarding their booking status.

Key Features:

1.User Authentication: The system allows users to register and login securely to access the booking and waiting list management functionalities.

2.Booking Management: Passengers can search for train schedules, book tickets online, and specify their preferences for date, time, and class of travel.

3.Real-time Updates: The system displays real-time information on seat availability and waiting list status, allowing passengers to track their position in the queue.

4.Automated Allocation: As cancellations occur or seats become available, the system automatically allocates seats to passengers on the waiting list based on predefined criteria such as priority and booking time.

5.Priority Allocation: Passengers with special needs, medical emergencies etc.

Concept Used:

1. File Handling

File handling in a Railway Waiting List Management System involves the creation, reading, updating, and deletion (CRUD operations) of various files or databases to manage passenger data, booking records, waiting lists, system logs, and reports. Here's how file handling can be used in different aspects of the system:

Passenger Data Storage: File management is used to store passenger information such as name, contact details, booking history, preferences, and payment records. This data can be organized in structured files or databases for easy retrieval and manipulation.

Booking Records: Each booking transaction generates a record containing details such as booking ID, passenger information, journey details, seat preferences, and payment status. These records are stored in files or databases to maintain a comprehensive history of bookings and facilitate efficient retrieval when needed.

Waiting List Management: The system maintains a waiting list of passengers who have not yet been allocated seats. File management is used to store and update the waiting

list, including passenger details and their position in the queue. As seats become available, the system updates the waiting list accordingly.

Backup and Recovery: File management includes procedures for regular backups of critical data to prevent data loss in case of system failures or disasters. Backup files are stored securely and can be used for restoring data in the event of corruption or loss.

Security and Access Control: File management incorporates mechanisms for ensuring data security and access control. Access to sensitive files containing passenger information and system logs is restricted to authorized users through authentication and authorization mechanisms.

2. Process Scheduling (FCFS)

Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process based on a particular strategy.
Process scheduling is an essential part of a Multiprogramming operating system. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Categories of Scheduling
Scheduling falls into one of two categories:

- **Non-preemptive:** In this case, a process's resource cannot be taken before the process has finished running. When a running process finishes and transitions to a waiting state, resources are switched.
- **Preemptive:** In this case, the OS assigns resources to a process for a predetermined period. The process switches from running state to ready state or from waiting for state to ready state during resource allocation. This switching happens because the CPU may give other processes priority and substitute the currently active process for the higher priority process.

In this project we have used first come first serve algorithm

First Come First Serve – CPU Scheduling (Non-Preemptive)

Simplest CPU scheduling algorithm that schedules according to arrival times of processes The first come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked to the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non-preemptive scheduling algorithm.

Characteristics of FCFS

- FCFS supports non-preemptive and preemptive CPU scheduling algorithms.
- Tasks are always executed on a First-come, First-serve concept.
- FCFS is easy to implement and use.
- This algorithm is not very efficient in performance, and the wait time is quite high.

Advantages of FCFS

Here, are pros/benefits of using FCFS scheduling algorithm:

- The simplest form of a CPU scheduling algorithm
- Easy to program
- First come first served

Disadvantages of FCFS

Here, are cons/ drawbacks of using FCFS scheduling algorithm:

- It is a Non-preemptive CPU scheduling algorithm, so after the process has been allocated to the CPU, it will never release the CPU until it finishes executing.
- The Average Waiting Time is high.
- Short processes that are at the back of the queue have to wait for the long process at the front to finish.
- Not an ideal technique for time-sharing systems.
- Because of its simplicity, FCFS is not very efficient.

USE OF FCFS IN RAILWAY WAITING LIST MANAGEMENT SYSTEM

FCFS (First-Come-First-Served) scheduling is utilized to manage the waiting list for passengers who couldn't get a booked ticket due to all seats being occupied. FCFS is applied to passengers under 60 years old. Here's how FCFS is used:

1. When a passenger tries to book a ticket but all seats are already booked, they are added to the general waiting list and either the FCFS or age priority waiting list depending on their age.
2. If the passenger is under 60 years old, they are added to the FCFS waiting list.
3. When a booked ticket is cancelled, the next passenger from the FCFS waiting list is given the newly available seat, adhering to the FCFS principle, i.e., the passenger who joined the waiting list first will get the seat first.
4. The waiting list is saved to a file (**waiting_list-fcfs.txt**) to persist the order of passengers in the FCFS queue.

3. Special reservations for senior citizen.

Senior citizen quota berths are berths remarked only for age of 60 years when traveling.

Implementation of senior citizen quota in a Railway Waiting List Management System involves allocating a certain percentage of seats specifically for senior citizens and managing their reservations accordingly. Here's how it could be done:

Identification of Senior Citizens:

Passengers who qualify for the senior citizen quota need to be identified during the booking process. This can be done by asking for their age or providing an option to select senior citizen status during registration.

Quota Allocation:

A certain percentage of seats on each train should be reserved for senior citizens. The quota allocation can be predefined based on government regulations or railway policies. The system should ensure that this quota is not exceeded, and seats within the quota are reserved exclusively for senior citizens until the quota is filled.

Booking Process:

When a senior citizen books a ticket, the system should check the availability of seats within the senior citizen quota. If seats are available, the booking is confirmed within the senior citizen quota. If the senior citizen quota is full, the system should either place the passenger on a separate waiting list for senior citizens or allocate seats from the general quota if available.

Priority Allocation:

Senior citizens who book within the senior citizen quota should be given priority over those booking under the general quota. If a senior citizen is on the waiting list, they should be given priority for seat allocation when seats become available, even if they are lower on the waiting list compared to non-senior citizens.

Notification and Communication:

The system should send notifications to senior citizens regarding their booking status, including confirmation, waitlist movement, or cancellation. Senior citizens should also be informed of any special services or assistance available to them during their journey.

Reporting and Monitoring:

The system should provide reports on the utilization of the senior citizen quota, including the number of seats booked, waiting list status, and overall occupancy. Railway authorities can

monitor the effectiveness of the senior citizen quota and make adjustments as needed based on demand and usage patterns.

Benefits of creating railway waiting list management system:

Efficiency: Automating the waiting list management process reduces manual effort and improves efficiency.

Customer Satisfaction: Passengers have transparency regarding their booking status, leading to enhanced satisfaction.

Revenue Optimization: By maximizing seat occupancy and minimizing cancellations, the system helps optimize revenue generation for the railway authorities.

Data-driven Insights: The system provides valuable insights through analytics and reporting, enabling informed decision-making and resource optimization.

Future Enhancements:

Integration with payment gateways for online ticket booking and cancellation.

Mobile application development for enhanced accessibility and convenience.

Integration with external systems for seamless interoperability.

SOURCE CODE

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.Scanner;

public class ospb4 {

    private static final int TOTAL_SEATS = 3;
    private static final String BOOKED_FILE = "booked_tickets.txt";
    private static final String WAITING_LIST_FILE_FCFS = "waiting_list-
fcfs.txt";
    private static final String WAITING_LIST_FILE_AGE = "waiting_list-
age.txt";
      private static final String GENERAL_WAITING_LIST_FILE =
"general_waiting_list.txt";

    private static Map<String, Passenger> bookedSeats;
    private static LinkedList<Passenger> waitingListFcfs;
    private static PriorityQueue<Passenger> waitingListAgePriority;
    private static LinkedList<Passenger> generalWaitingList;
    private static int nextSeatNumber = 1;

    public static void main(String[] args) {
        bookedSeats = loadBookedTickets();
        waitingListFcfs = loadWaitingListFcfs();
        waitingListAgePriority = loadWaitingListAgePriority();
        generalWaitingList = loadGeneralWaitingList();

        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\nTrain Ticket Booking System");
            System.out.println("1. Book Ticket");
            System.out.println("2. Cancel Ticket");
            System.out.println("3. Display Booked Tickets");
            System.out.println("4. Display Waiting Lists");
            System.out.println("5. Display General Waiting List");
```

```
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    bookTicket();
                    break;
                case 2:
                    cancelTicket();
                    break;
                case 3:
                    displayBookedTickets();
                    break;
                case 4:
                    displayWaitingLists();
                    break;
                case 5:
                    displayGeneralWaitingList();
                    break;
                case 6:
                    System.out.println("Exiting the system...");
                default:
                    System.out.println("Invalid choice!");
            }
        } while (choice != 6);

        scanner.close();
        saveBookedTickets();
        saveWaitingListFcfs();
        saveWaitingListAgePriority();
        saveGeneralWaitingList();
    }

    private static void bookTicket() {
        Scanner scanner = new Scanner(System.in);

        if (bookedSeats.size() < TOTAL_SEATS) {
            System.out.print("Enter passenger name: ");
            String name = scanner.nextLine();
            System.out.print("Enter passenger age: ");
            int age = scanner.nextInt();
            System.out.print("Enter passenger gender (M/F): ");
            String gender = scanner.next().toUpperCase();

            bookedSeats.put(name, new Passenger(name, age,
gender).setSeatNumber(nextSeatNumber++));
```

```
            System.out.println("Ticket booked successfully for " + name + ".
Seat number: " + bookedSeats.get(name).getSeatNumber());
            saveBookedTickets();
        } else {
            System.out.println("wait");
            System.out.print("Enter passenger name: ");
            String name = scanner.nextLine();
            System.out.print("Enter passenger age: ");
            int age = scanner.nextInt();
            System.out.print("Enter passenger gender (M/F): ");
            String gender = scanner.next().toUpperCase();

            generalWaitingList.add(new Passenger(name, age, gender));
            saveGeneralWaitingList();

            if (age > 60) {
                waitingListAgePriority.add(new Passenger(name, age, gender));
            } else {
                waitingListFcfs.add(new Passenger(name, age, gender));
            }
            saveWaitingListFcfs();
            saveWaitingListAgePriority();
        }
    }

    private static void cancelTicket() {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter passenger name to cancel: ");
        String name = scanner.nextLine();

        if (bookedSeats.containsKey(name)) {
            Passenger passenger = bookedSeats.remove(name);
                        System.out.println("Ticket canceled for " + name + ".
Seat number: " + passenger.getSeatNumber());
             nextSeatNumber--;
              generalWaitingList.remove(new Passenger(name,
passenger.getAge(), passenger.getGender()));
            saveGeneralWaitingList();

            Passenger nextPassenger = null;
            if (!waitingListAgePriority.isEmpty()) {
                nextPassenger = waitingListAgePriority.poll();
            } else if (!waitingListFcfs.isEmpty()) {
                nextPassenger = waitingListFcfs.poll();
            }

            if (nextPassenger != null) {
```

```
                bookedSeats.put(nextPassenger.getName(),
nextPassenger.setSeatNumber(bookedSeats.size() + 1));
                System.out.println(nextPassenger.getName() + " from the
waiting list has been assigned seat number " + nextPassenger.getSeatNumber());
            }

            saveBookedTickets();
            saveWaitingListFcfs();
            saveWaitingListAgePriority();
        } else {
            System.out.println(name + " does not have a booked ticket.");
        }
    }

    private static void displayBookedTickets() {
        if (!bookedSeats.isEmpty()) {
            System.out.println("\nBooked Tickets:");
            for (Passenger passenger : bookedSeats.values()) {
                System.out.println("Passenger: " + passenger.getName() + ",
Age: " + passenger.getAge() + ", Gender: " + passenger.getGender() + ", Seat
number: " + passenger.getSeatNumber());
            }
        } else {
            System.out.println("There are no booked tickets currently.");
        }
    }

    private static void displayWaitingLists() {
        System.out.println("\nWaiting Lists:");
        if (!waitingListAgePriority.isEmpty()) {
            System.out.println("  - Age Priority (over 60):");
            for (Passenger passenger : waitingListAgePriority) {
                System.out.println("    Passenger: " + passenger.getName() +
", Age: " + passenger.getAge() + ", Gender: " + passenger.getGender());
            }
        } else {
            System.out.println("    - Age Priority (over 60): Empty");
        }
        if (!waitingListFcfs.isEmpty()) {
            System.out.println("  - FCFS (under 60):");
            for (Passenger passenger : waitingListFcfs) {
                System.out.println("    Passenger: " + passenger.getName() +
", Age: " + passenger.getAge() + ", Gender: " + passenger.getGender());
            }
        } else {
            System.out.println("    - FCFS (under 60): Empty");
        }
    }
```

```
    private static Map<String, Passenger> loadBookedTickets() {
        Map<String, Passenger> loadedSeats = new HashMap<>();
        try (Scanner scanner = new Scanner(new File(BOOKED_FILE))) {
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split(",");
                loadedSeats.put(parts[0], new Passenger(parts[0],
Integer.parseInt(parts[1]), parts[2]));
            }
        } catch (FileNotFoundException e) {
            System.out.println("Booked tickets file not found. Creating a new
one.");

        }
        return loadedSeats;
    }

    private static void saveBookedTickets() {
        try (FileWriter writer = new FileWriter(BOOKED_FILE)) {
            for (Passenger passenger : bookedSeats.values()) {
                writer.write(passenger.getName() + "," +
passenger.getSeatNumber() + "," + passenger.getGender() + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static LinkedList<Passenger> loadWaitingListFcfs() {
        LinkedList<Passenger> loadedList = new LinkedList<>();
        try (Scanner scanner = new Scanner(new File(WAITING_LIST_FILE_FCFS)))
{
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split(",");
                loadedList.add(new Passenger(parts[0],
Integer.parseInt(parts[1]), parts[2]));
            }
        } catch (FileNotFoundException e) {
            System.out.println("FCFS waiting list file not found. Creating a
new one.");

        }
        return loadedList;
    }

    private static void saveWaitingListFcfs() {
```

```
        try (FileWriter writer = new FileWriter(WAITING_LIST_FILE_FCFS)) {
            for (Passenger passenger : waitingListFcfs) {
                writer.write(passenger.getName() + "," + passenger.getAge() +
"," + passenger.getGender() + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static PriorityQueue<Passenger> loadWaitingListAgePriority() {
        PriorityQueue<Passenger> loadedList = new PriorityQueue<>((p1, p2) ->
Integer.compare(p2.getAge(), p1.getAge())); // Sorts by age descending (older
first)
        try (Scanner scanner = new Scanner(new File(WAITING_LIST_FILE_AGE))) {
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split(",");
                loadedList.add(new Passenger(parts[0],
Integer.parseInt(parts[1]), parts[2]));
            }
        } catch (FileNotFoundException e) {
            System.out.println("Age priority waiting list file not found.
Creating a new one.");}

        return loadedList;
    }

    private static void saveWaitingListAgePriority() {
        try (FileWriter writer = new FileWriter(WAITING_LIST_FILE_AGE)) {
            for (Passenger passenger : waitingListAgePriority) {
                writer.write(passenger.getName() + "," + passenger.getAge() +
"," + passenger.getGender() + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static LinkedList<Passenger> loadGeneralWaitingList() {
    LinkedList<Passenger> loadedList = new LinkedList<>();
    try (Scanner scanner = new Scanner(new File(GENERAL_WAITING_LIST_FILE))) {
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split(",");
            loadedList.add(new Passenger(parts[0], Integer.parseInt(parts[1]),
parts[2]));
        }
```

```
    } catch (FileNotFoundException e) {
        System.out.println("General waiting list file not found. Creating a
new one.");}

    return loadedList;
}

private static void saveGeneralWaitingList() {
    try (FileWriter writer = new FileWriter(GENERAL_WAITING_LIST_FILE)) {
        for (Passenger passenger : generalWaitingList) {
            writer.write(passenger.getName() + "," + passenger.getAge() + ","
+ passenger.getGender() + "\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}


    private static void displayGeneralWaitingList() {
        System.out.println("\nGeneral Waiting List (Informational):");
        if (!generalWaitingList.isEmpty()) {
            for (Passenger passenger : generalWaitingList) {
                System.out.println("    Passenger: " + passenger.getName() +
", Age: " + passenger.getAge() + ", Gender: " + passenger.getGender());
            }
        } else {
            System.out.println("    - Empty");
        }
    }

}

class Passenger {
    private String name;
    private int age;
    private String gender;
    private int seatNumber;

    public Passenger(String name, int age, String gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.seatNumber=-1;
    }

    public String getName() {
        return name;
    }
```

```
    public int getAge() {
        return age;
    }

    public String getGender() {
        return gender;
    }

    public int getSeatNumber() {
        return seatNumber;
    }

    public Passenger setSeatNumber(int seatNumber) {
         if (seatNumber <= 0) {
            throw new IllegalArgumentException("Seat number must be
positive.");
        }
        this.seatNumber = seatNumber;
        return this;
    }
}
```

OUTPUTS :

```
PS C:\Users\Aniket> cd "c:\Users\Aniket\OneDrive\Desktop\" ; if ($?) { javac ospb4.java } ; if ($?) { java ospb4 }

Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 1
Enter passenger name: xyz
Enter passenger age: 21
Enter passenger gender (M/F): m
Ticket booked successfully for xyz. Seat number: 1
```

After adding some dummy passengers.

DISPLAYING THE BOOKED TICKETS.

```
Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 3

Booked Tickets:
Passenger: abc, Age: 69, Gender: M, Seat number: 2
Passenger: xyz, Age: 21, Gender: M, Seat number: 1
Passenger: ghi, Age: 53, Gender: F, Seat number: 3
```

DISPLAYING THE WAITING LIST.

```
Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 4

Waiting Lists:
  - Age Priority (over 60):
    Passenger: john, Age: 84, Gender: M
    Passenger: sid, Age: 73, Gender: M
  - FCFS (under 60):
    Passenger: ani, Age: 21, Gender: M
    Passenger: sara, Age: 16, Gender: F
```

DISPLAYING THE GENERAL WAITING LIST.

```
Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 5

General Waiting List (Informational):
    Passenger: ani, Age: 21, Gender: M
    Passenger: sid, Age: 73, Gender: M
    Passenger: john, Age: 84, Gender: M
    Passenger: sara, Age: 16, Gender: F
```

CANCELLING THE TICKET FOR PASSENGER (NAMED XYZ).

```
Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 2
Enter passenger name to cancel: xyz
Ticket canceled for xyz. Seat number: 1
john from the waiting list has been assigned seat number 3
```

DISPLAYING THE BOOKED SEATS AFTER CANCELLATION.

```
Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 3

Booked Tickets:
Passenger: abc, Age: 69, Gender: M, Seat number: 2
Passenger: ghi, Age: 53, Gender: F, Seat number: 3
Passenger: john, Age: 84, Gender: M, Seat number: 3
```

PASSENGER (JOHN) HAS BEEN ALLOCATED WITH A CONFIRMED TICKET.

```
Train Ticket Booking System
1. Book Ticket
2. Cancel Ticket
3. Display Booked Tickets
4. Display Waiting Lists
5. Display General Waiting List
6. Exit
Enter your choice: 4

Waiting Lists:
  - Age Priority (over 60):
    Passenger: sid, Age: 73, Gender: M
  - FCFS (under 60):
    Passenger: ani, Age: 21, Gender: M
    Passenger: sara, Age: 16, Gender: F
```

CONCLUSION

The Railway Waiting List Management System is a comprehensive solution designed to modernize and streamline the process of managing waiting lists for train reservations. By leveraging automation, real-time updates, and process scheduling algorithms,this system aims to automate the allocation of seats or berths to passengers on the waiting list, ensuring optimal occupancy and minimizing cancellations and taking care of special reservations. It provides updates on seat availability, prioritizes passenger allocations, and facilitates efficient communication with passengers regarding their booking status, the system aims to enhance operational efficiency, customer satisfaction, and revenue generation for railway authorities.