

EX NO:1	Write the complete problem statement
DATE	

AIM:

To prepare PROBLEM STATEMENT for any project.

ALGORITHM:

1. The problem statement is the initial starting point for a project.
2. A problem statement describes what needs to be done without describing how.
3. It is basically a one-to-three-page statement that everyone on the project agrees with that describes what will be done at a high level.
4. The problem statement is intended for a broad audience and should be written in non-technical terms.
5. It helps the non-technical and technical personnel communicate by providing a description of a problem.
6. It doesn't describe the solution to the problem.

INPUT:

1. The input to requirement engineering is the problem statement prepared by customer.
2. It may give an overview of the existing system along with broad expectations from the new system.
3. The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements.
4. Here, requirements are identified with the help of customer and existing system processes.

Problem:

A Bus Ticket Reservation System plays a vital role in facilitating the booking of tickets for passengers traveling between cities, regions, or countries. However, many bus companies still rely on manual or outdated systems to handle customer reservations, ticket sales, and schedule management. This leads to inefficiencies such as overbooking, mismanagement of seat allocation, and delays in ticket issuance. To address these challenges, a centralized and automated Bus Ticket Reservation System is needed to streamline ticket booking, seat allocation, schedule management, and payment processes, ensuring smooth operations and better customer experience.

Background:

The transportation industry, particularly bus services, has witnessed a significant increase in passenger demand due to urbanization and growing travel needs. However, many bus companies are still dependent on manual methods or outdated systems to handle reservations, which often results in overbooking, inaccurate seat availability, and delayed ticket processing. A centralized system could help automate ticket reservations, provide real-time updates on seat availability, streamline payment processing, and allow passengers to easily view schedules, track their bookings, and make adjustments. Such a system would enhance customer satisfaction, improve operational efficiency, and ensure better management of available resources.

Relevance:

An efficient Bus Ticket Reservation System is critical for both bus operators and passengers. By automating the ticket booking and management process, it ensures accurate seat availability, prevents overbooking, and provides real-time updates on bus schedules. This system enhances the passenger experience by offering easy access to tickets, reservations, and payment options. Additionally, it enables bus operators to manage their fleet more effectively,

reduce operational costs, and improve their overall service quality. A well-designed system ensures that buses are operating at full capacity, leading to better revenue management and an improved travel experience for passengers.

Objectives:

The primary objective of this project is to develop a centralized Blood Bank Management System that enhances operational efficiency, reduces errors, and ensures a steady supply of blood to meet patient and hospital needs. Specific objectives include:

1. **Patient Registration:** Enable efficient registration of new patients, capturing their details and medical history.
2. **Appointment Booking:** Allow patients to book appointments for blood donation or transfusion with ease.
3. **Hospital Blood Requests:** Facilitate hospitals in requesting specific blood types and quantities based on real-time inventory levels.
4. **Approval and Management of Requests:** Provide blood bank administrators the tools to review and approve requests from hospitals, ensuring alignment with available inventory.
5. **Inventory Management:** Implement real-time tracking of blood stock levels, including donor blood types, and manage the storage and expiration of blood units.
6. **Report Generation:** Generate comprehensive reports on blood inventory, donation appointments, and request fulfillments, aiding in compliance and resource planning.

Result:

EX NO:2	Write the software requirement specification document
DATE	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet(SRS) for any Project.

ALGORITHM:

SRS shall address are the following:

- a) **Functionality.** What is the software supposed to do?
- b) **External interfaces.** How does the software interact with people, the system's hardware, other hardware, and other software?
- c) **Performance.** What is the speed, availability, response time, recovery time of various software functions, etc.?
- d) **Attributes.** What is the portability, correctness, maintainability, security, etc. considerations?
- e) **Design constraints imposed on an implementation.** Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

1. Introduction*1.1 Purpose*

This document describes the requirements for developing a Bus Reservation System (BRS). The goal of this system is to simplify bus ticket booking, seat allocation, schedule management, payment processing, and reporting for bus operators and passengers. By implementing BRS, bus operators will be able to efficiently manage bus schedules, reservations, and payments, enhancing customer service and operational efficiency.

1.2 Scope

The BRS will be a web-based application accessible by bus operators, passengers, and administrators. This system will handle core functionalities such as searching for buses, booking tickets, managing reservations, handling payments, and generating reports to ensure smooth operations and an optimal travel experience.

1.3 Definitions, Acronyms, and Abbreviations

- **BRS:** Bus Reservation System
- **Admin:** Bus Reservation System Administrator
- **Passenger:** Individual booking a bus ticket
- **Bus Operator:** Organization managing the buses and routes
- **Ticket:** A travel document issued to a passenger confirming their reservation
- **Payment:** Transaction confirming ticket purchase

1.4 Overview

This document details the required functionalities, interfaces, and performance standards for the BRS. It serves as a guide for the development team and a reference for users to understand what the system will offer.

2. Overall Description

2.1 Product Perspective

BRS is a centralized solution designed to handle all aspects of bus reservation, ticketing, and payment processing. The system will be integrated with existing bus databases and accessible online, making it easier for users to book tickets, manage reservations, and access bus schedules.

2.2 Product Functions

- **Search Buses:** Passengers can search for available buses based on their departure and arrival locations.
- **Ticket Booking:** Passengers can book and cancel tickets for their desired bus.
- **Seat Allocation:** The system will allocate seats based on passenger preferences and availability.
- **Payment Processing:** The system will handle online payments for bus tickets.
- **Schedule Management:** Bus operators can update bus schedules, routes, and available buses.
- **Report Generation:** The system will generate reports on reservations, payments, and bus schedules.

2.3 User Classes and Characteristics

- **Admin:** Manages system configurations, user roles, and overall bus reservation operations.
- **Passenger:** Books tickets, manages reservations, and processes payments.
- **Bus Operator:** Responsible for managing bus schedules, routes, and seat availability.
- **Payment System:** Processes online payments for tickets.

2.4 Operating Environment

The system will be accessible via a web browser, supporting both desktop and mobile devices.

2.5 Design and Implementation Constraints

- The system must ensure secure transactions for ticket bookings and payments.
- It should handle multiple users concurrently, including passengers and bus operators.

2.6 Assumptions and Dependencies

- Users will have internet access to log into the system.
- The system will require a database to store passenger, bus, and payment data.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Search Buses Module

- The system allows passengers to search for available buses by route, date, and time.
- The system displays available buses along with details such as departure time, bus type, and price.

3.1.2 Ticket Booking Module

- Passengers can book, cancel, and modify reservations.

- Available seats are displayed for each bus, and passengers can select their desired seats.
- Users receive confirmation of their booking via email or SMS.

3.1.3 Seat Allocation Module

- The system ensures that passengers can select from available seats.
- Reserved seats are marked as unavailable to prevent overbooking.

3.1.4 Payment Processing Module

- Passengers can securely process payments through integrated payment gateways (e.g., credit cards, digital wallets).
- The system confirms payment and generates tickets.
- A history of all payments made is available for passengers.

3.1.5 Schedule Management Module

- Bus operators can update bus schedules, routes, and seat availability.
- The system allows the bus operator to manage cancellations, delays, or changes to the schedule.

3.1.6 Report Generation Module

- The system generates reports on reservations, revenue, available buses, and passenger activity.
- Reports can be filtered by date, route, or bus, and downloaded in PDF and Excel formats.

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

- The system should handle up to 2,000 users concurrently without affecting performance.
- Response time for actions such as booking a ticket or processing payment should be under 3 seconds.

3.2.2 Security Requirements

- User logins should be required for accessing the system.
- All sensitive data, such as payment details, should be encrypted.
- Only authorized users (admins and bus operators) should have access to critical system functions.

3.2.3 Usability Requirements

- The interface should be user-friendly and intuitive, supporting both desktop and mobile versions.
- It should be easy to navigate and book tickets for passengers with minimal effort.

3.2.4 Reliability Requirements

- The system should be operational 99.9% of the time.
- Any issues or bugs should be resolved within 10 minutes.

4. External Interface Requirements

4.1 User Interfaces

- The system will be responsive, with a dynamic interface that adapts to various screen sizes for a seamless user experience.
- Each user type (admin, bus operator, passenger) will have access to relevant functions based on their role.

4.2 Hardware Interfaces

- The system will be compatible with standard desktop and mobile devices, including laptops, tablets, and smartphones.

4.3 Software Interfaces

- The system will use an SQL database for storing bus schedules, passenger details, ticket bookings, and payment data.
- The system will integrate with third-party payment gateways to process ticket payments securely.
- Email and SMS services will be used to send booking confirmations, reminders, and updates.

5. Additional Requirements

5.1 Data Privacy and Compliance

All passenger data, including personal and payment details, will be handled in compliance with relevant data protection laws (e.g., GDPR, PCI-DSS).

5.2 Documentation

User manuals and system documentation will be provided for end-users (passengers, bus operators, admins) and technical staff for system maintenance and troubleshooting.

Result:

EX NO:3	Draw the entity relationship diagram
DATE	

AIM:

To Draw the Entity Relationship Diagram for any project.

ALGORITHM:

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relation Types

Step 4: Mapping of Binary 1:N Relationship Types.

Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

INPUT:

Entities

Entity Relationship Matrix

Primary Keys

Attributes

Mapping of Attributes with Entities

Result:

EX NO:4	Draw the data flow diagrams at level 0 and level 1
DATE	

AIM:

To Draw the Data Flow Diagram for any project and List the Modules in the Application.

ALGORITHM:

1. Open the Visual Paradigm to draw DFD (Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram

INPUT:

Processes

Datastores

External Entities

Result:

EX NO:5	Draw use case diagram
DATE	

AIM:

To Draw the Use Case Diagram for any project

ALGORITHM:

Step 1: Identify Actors

Step 2: Identify Use Cases

Step 3: Connect Actors and Use Cases

Step 4: Add System Boundary

Step 5: Define Relationships

Step 6: Review and Refine

Step 7: Validate

INPUTS:

Actors

Use Cases

Relations

Result:

EX NO:6	Draw activity diagram of all use cases.
DATE	

AIM:

To Draw the activity Diagram for any project

ALGORITHM:

Step 1: Identify the Initial State and Final States

Step 2: Identify the Intermediate Activities Needed

Step 3: Identify the Conditions or Constraints

Step 4: Draw the Diagram with Appropriate Notations

INPUTS:

Activities

Decision Points

Guards

Parallel Activities

Conditions

Result:

EX NO:7	Draw state chart diagram of all use cases.
DATE	

AIM:

To Draw the State Chart Diagram for any project

ALGORITHM:

STEP-1: Identify the important objects to be analysed.

STEP-2: Identify the states.

STEP-3: Identify the events.

INPUTS:

Objects

States

Events

Result:

EX NO:8	Draw sequence diagram of all use cases.
DATE	

AIM:

To Draw the Sequence Diagram for any project

ALGORITHM:

1. Identify the Scenario
2. List the Participants
3. Define Lifelines
4. Arrange Lifelines
5. Add Activation Bars
6. Draw Messages
7. Include Return Messages
8. Indicate Timing and Order
9. Include Conditions and Loops
10. Consider Parallel Execution
11. Review and Refine
12. Add Annotations and Comments
13. Document Assumptions and Constraints
14. Use a Tool to create a neat sequence diagram

INPUTS:

Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:

EX NO:9	Draw collaboration diagram of all use cases
DATE	

Draw collaboration diagram of all use cases

AIM:

To Draw the Collaboration Diagram for any project

ALGORITHM:

Step 1: Identify Objects/Participants

Step 2: Define Interactions

Step 3: Add Messages

Step 4: Consider Relationships

Step 5: Document the collaboration diagram along with any relevant explanations or annotations.

INPUTS:

Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:

EX NO:10	Assign objects in sequence diagram to classes and make class diagram.
DATE	

AIM:

To Draw the Class Diagram for any project

ALGORITHM:

1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

INPUTS:

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

Result:

EX NO:1	Mini Project-Bus Reservation System
DATE	

Aim:

The Bus Ticket Reservation System aims to efficiently manage passenger bookings, seat availability, and bus schedules, ensuring smooth travel experiences. It helps optimize ticket sales and seat allocation, facilitating better organization and accessibility for both passengers and bus operators. The system streamlines the booking process, reducing errors and enhancing operational efficiency.

Algorithm:

- **Passenger Registration:** Collect and verify passenger details, ensuring accurate information for bookings.
- **Ticket Booking:** Record the passenger's ticket reservation, updating seat availability and bus schedules.
- **Seat Allocation:** Manage seat assignments, ensuring no overbooking and optimizing seat usage for each bus.
- **Schedule Management:** Update and maintain bus schedules, routes, and departure times for available buses.
- **Payment Processing:** Allow passengers to make payments securely, confirming ticket bookings once payment is received.
- **Booking Confirmation:** Approve or deny reservations based on seat availability, notifying passengers of their booking status.
- **Generate Reports:** Summarize data on bookings, revenue, and seat occupancy for administrative review.

Program:

Mysql code:

```
CREATE DATABASE bus_reservation;
```

```
USE bus_reservation;
```

```
CREATE TABLE tickets (
```

```
    ticket_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    name VARCHAR(100),
```

```
    age INT,
```

```
    phone VARCHAR(15),
```

```
    from_city VARCHAR(50),
```



```
to_city VARCHAR(50),  
seat_number INT  
);
```

Python code:

```
import streamlit as st
```

```
import pandas as pd
```

```
import mysql.connector
```

```
# MySQL Database Connection
```

```
def get_db_connection():
```

```
    return mysql.connector.connect(  
        host="localhost", # Update with your MySQL host
```

```
        user="root",      # Update with your MySQL username
```

```
        password="password", # Update with your MySQL password
```

```
        database="bus_reservation"
```

```
    )
```

```
# List of cities in Tamil Nadu
```

```
cities = [  
    "Chennai", "Coimbatore", "Madurai", "Trichy", "Salem",
```

```
    "Erode", "Tirunelveli", "Thanjavur", "Vellore", "Thoothukudi"
```

```
]
```

```
# Function to book a ticket
```

```
def book_ticket():
```

```
    st.title("Bus Reservation System - Book Ticket")
```

```
    name = st.text_input("Enter your name:")
```

```
    age = st.number_input("Enter your age:", min_value=1, max_value=120)
```

```
    phone = st.text_input("Enter your phone number:")
```

```
    from_city = st.selectbox("From:", cities)
```

```
    to_city = st.selectbox("To:", [city for city in cities if city != from_city]) # Prevent selecting the same city
```

```
    seat_number = st.number_input("Select seat number:", min_value=1, max_value=50)
```

```

if st.button("Book Ticket"):
    if name.strip() == "":
        st.error("Name is a mandatory field. Please enter your name.")
    else:
        try:
            connection = get_db_connection()
            cursor = connection.cursor()

            # Insert ticket into the database
            query = """
            INSERT INTO tickets (name, age, phone, from_city, to_city, seat_number)
            VALUES (%s, %s, %s, %s, %s, %s)
            """
            cursor.execute(query, (name, age, phone, from_city, to_city, seat_number))
            connection.commit()
            ticket_id = cursor.lastrowid

            st.success(f"Ticket booked successfully! Your Ticket ID is {ticket_id}")
        except mysql.connector.Error as err:
            st.error(f"Error: {err}")
        finally:
            if connection.is_connected():
                cursor.close()
                connection.close()

# Function to view a ticket
def view_ticket():
    st.title("Bus Reservation System - View Ticket")
    ticket_id = st.number_input("Enter your Ticket ID:", min_value=1, step=1)

    if st.button("View Ticket"):
        try:
            connection = get_db_connection()
            cursor = connection.cursor(dictionary=True)

```

```

# Fetch ticket details

query = "SELECT * FROM tickets WHERE ticket_id = %s"
cursor.execute(query, (ticket_id,))
ticket = cursor.fetchone()

if ticket:
    st.write("### Ticket Details")
    st.write(pd.DataFrame([ticket]))
else:
    st.error("Ticket not found. Please check the Ticket ID.")
except mysql.connector.Error as err:
    st.error(f"Error: {err}")
finally:
    if connection.is_connected():
        cursor.close()
        connection.close()

# Main function for Streamlit app
def main():
    st.sidebar.title("Bus Reservation System")
    option = st.sidebar.selectbox("Choose an option", ["Book Ticket", "View Ticket"])

    if option == "Book Ticket":
        book_ticket()
    elif option == "View Ticket":
        view_ticket()

if __name__ == "__main__":
    main()

```

Conclusion :

In conclusion, the bus reservation system streamlines the booking process by providing users with a convenient, efficient platform to reserve tickets. It enhances customer experience through real-time availability updates and multiple payment options. The system reduces manual errors and improves operational efficiency for bus operators. Additionally, it offers flexibility with features like seat selection and journey tracking. Overall, it is an essential tool for modernizing bus transportation and improving both user and operator satisfaction.