Submitted by

Name: Varsha

College: Kathir College of Engineering

Department: Computer and communication Engineering

Year: 2025–2026

1. System Overview

We are developing a cloud-hosted online ticket booking system for buses with a focus on:

Security (prevent ticket loss/theft, avoid incorrect pricing)

High availability and scalability

Seamless user experience

2. Key Features

User Features

Browse bus routes, schedules, and seat availability.

Book tickets online with payment integration.

Receive e-ticket with QR code for boarding.

View booking history and cancel tickets.

Admin Features

Manage routes, buses, schedules, and ticket pricing.

Monitor bookings and generate reports.

Dynamic pricing adjustments (if required).

3. Security Measures

To prevent ticket loss, theft, and pricing errors:

E-ticket system: Tickets issued digitally with unique QR codes.

Payment validation: Integrate secure payment gateways (Stripe, PayPal, Razorpay).

Transaction logging: Track all bookings and cancellations.

Data validation: Ensure correct pricing based on route, seat class, and discounts.

4. Architecture Design

Cloud-Based Architecture:

Users → Load Balancer → Web Servers (Auto-Scaling) → App Servers

|

V

Database Cluster

|

V

Ticket Service

|

V

Payment Gateway

Components:

Frontend: React / Angular for smooth booking UI.

Backend: Node.js / Django / Spring Boot handling booking logic.

Database: Cloud-hosted SQL/NoSQL database (AWS RDS, DynamoDB).

Ticket Service: Microservice generating unique e-tickets and validating bookings.

Payment Gateway: Integration for secure online payments.

Load Balancer & Auto-Scaling: Ensures high traffic handling and reliability.

5. Scalability & Reliability

Auto-scaling servers: Dynamically provision new instances during high traffic.

Caching: Use Redis/Memcached for frequently accessed data (routes, schedules).

Content Delivery Network (CDN): For fast frontend asset delivery.

Database replication: Ensure high availability and fault tolerance.

Monitoring & Alerts: CloudWatch / Grafana / Prometheus for system health.

6. Implementation Steps

Design database schema

Users, Buses, Routes, Tickets, Payments.

Develop backend APIs

User registration/login, ticket booking, payment processing.

Develop frontend interface

Booking pages, seat selection, e-ticket download.

Integrate ticket generation & QR code

Integrate payment gateway

Set up cloud infrastructure

Deploy backend on AWS EC2 / Google Cloud VM or containerized via Kubernetes.

Use S3 / Cloud Storage for ticket storage.

Configure load balancer & auto-scaling groups.

Test system

Load testing for high traffic

Security testing for fraud prevention

Deploy to production

Enable SSL, monitoring, logging, and auto-scaling.

7. Testing

Functional Testing: Booking, cancellation, payment flows.

Load Testing: Simulate high concurrent users.

Security Testing: Validate ticket authenticity and pricing.

Failover Testing: Ensure system stability during server failure.

- Backend nodejs

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bodyParser = require('body-parser');
const QRCode = require('qrcode');

const app = express();
app.use(cors());
app.use(bodyParser.json());

// MongoDB connection
mongoose.connect('mongodb+srv://<username>:<password>@cluster0.mongodb.net/busBookingDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log("MongoDB Connected"))
.catch(err => console.log(err));

// Schemas
const TicketSchema = new mongoose.Schema({
  userName: String,
  busRoute: String,
  seatNumber: Number,
  price: Number,
```

```
  ticketCode: String,
});

const Ticket = mongoose.model('Ticket', TicketSchema);

// API: Book Ticket
app.post('/book', async (req, res) => {
  const { userName, busRoute, seatNumber, price } = req.body;

  // Generate unique ticket code
  const ticketCode = `TICKET-${Date.now()}-${seatNumber}`;

  // Generate QR code (Base64)
  const qrCodeData = await QRCode.toDataURL(ticketCode);

  const ticket = new Ticket({
    userName,
    busRoute,
    seatNumber,
    price,
    ticketCode
  });

  await ticket.save();

  res.json({ message: 'Ticket booked successfully', ticketCode, qrCodeData });
```

```
});

// API: Get all tickets (for admin)
app.get('/tickets', async (req, res) => {
  const tickets = await Ticket.find();
  res.json(tickets);
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Forntend react

```
import React, { useState } from 'react';
import axios from 'axios';

function App() {
 const [form, setForm] = useState({ userName: '', busRoute: '', seatNumber: '', price: '' });
 const [ticket, setTicket] = useState(null);

 const handleChange = e => setForm({ ...form, [e.target.name]: e.target.value });

 const handleSubmit = async e => {
  e.preventDefault();
  const response = await axios.post('http://localhost:5000/book', form);
  setTicket(response.data);
 };
```

```
  return (
    <div style={{ padding: '20px' }}>
      <h1>Bus Ticket Booking</h1>
      <form onSubmit={handleSubmit}>
        <input name="userName" placeholder="Name" onChange={handleChange} required
/><br/>
        <input name="busRoute" placeholder="Bus Route" onChange={handleChange}
required /><br/>
        <input name="seatNumber" placeholder="Seat Number" type="number"
onChange={handleChange} required /><br/>
        <input name="price" placeholder="Price" type="number" onChange={handleChange}
required /><br/>
        <button type="submit">Book Ticket</button>
      </form>

      {ticket && (
        <div style={{ marginTop: '20px' }}>
          <h2>Ticket Booked!</h2>
          <p>Ticket Code: {ticket.ticketCode}</p>
          <img src={ticket.qrCodeData} alt="Ticket QR Code" />
        </div>
      )}
    </div>
  );
}

export default App;
```