# Bus Booking System

## Project Report

**Developed By:** Akil Shaikh

**Tech Stack:** Django, PostgreSQL (Supabase), Render

**Live Demo:** https://bus-booking-m9bt.onrender.com/

**Date:** 28 Feb 2026

**Abstract**
The Bus Booking System is a Django-based web application that enables users to search bus trips between cities, view route-wise trip listings with optional date filtering, select seats, and create bookings. The system includes seat locking with expiry to prevent double-booking, booking status lifecycle (Pending/Confirmed/Cancelled), auto-generated PNR, passenger mapping, and bus reviews. Data is stored in Supabase PostgreSQL and the application is deployed on Render.

## 1. Project Overview

This project provides a streamlined platform for bus trip discovery and booking. Users can search for trips by selecting a source city and destination city. The date field is optional: if no date is selected, the system lists upcoming trips starting today for the selected route.

## 2. Problem Statement

Manual or fragmented booking processes can be time-consuming and error-prone. Users often struggle to quickly find trip options, compare timings, and confirm seat availability reliably. This project aims to provide a centralized, user-friendly, and data-consistent solution for trip search and seat-aware bookings.

## 3. Objectives

1. Provide trip search by source, destination, and optional date.

2. If date is not selected, show upcoming trips from today onward for the selected route.

3. Maintain clean relational models for City, Route, Bus, Seat, and Trip.

4. Prevent double-booking through a SeatLock mechanism with expiry.

5. Create bookings with unique PNR and support booking statuses.

6. Store passenger details and seat-wise fare mapping.

7. Allow users to rate and review buses (one review per user per bus).

## 4. Technology Stack

**Backend:** Django (Python web framework)

**WSGI Server:** Gunicorn

**Database:** PostgreSQL on Supabase

**Deployment:** Render

**Static Files:** WhiteNoise

## 5. System Architecture

High-level request flow:
**User Browser** → **Django App (Render)** → **Supabase PostgreSQL**. The Django app renders templates for search and trip listing pages and interacts with the database using Django ORM.

## 6. Database Design

### 6.1 Core Models

**City**: Stores city details (name, state).
**Route**: Connects source city to destination city, includes distance. Uniqueness is enforced on (source, destination).

### 6.2 Bus, Seat & Trip Models

**Bus**: Operator name, bus number (unique), bus type, and total seats.
**Seat**: Linked to a bus, includes seat number, type (seater/sleeper), deck (lower/upper), and layout coordinates (row/col). Uniqueness is enforced on (bus, seat_number).
**Trip**: Linked to a bus and route with journey date, departure/arrival times, base fare, and active status. Uniqueness is enforced on (bus, journey_date, departure_time).

### 6.3 Booking Models

**SeatLock**: Prevents double-booking by locking a specific seat for a trip and user until an expiry time. Uniqueness is enforced on (trip, seat).
**Booking**: Stores booking record with user, trip, status, total fare, created time, and auto-generated PNR.
**Passenger**: Stores passenger details against a booking and seat.
**BookingSeat**: Maps seats to a booking with per-seat fare; uniqueness enforced on (booking, seat).

### 6.4 Review Model

**Review**: Users can rate and comment on buses. A unique constraint ensures one review per user per bus.

## 7. Features Implemented

• Route-based trip search with optional date filter.

• Upcoming trips fallback when date is not provided.

• Optimized trip listing using select_related for bus/route/city relations.

• Seat layout management per bus (seater/sleeper + decks).

• Seat locking with expiry to avoid double-booking.

• Booking workflow with PNR generation and booking statuses.

• Passenger mapping and seat-wise fares.

• Bus review system with one-review-per-user constraint.

## 8. Implementation Highlights

**Trip Search Behavior**
If the user selects source and destination but does not select a date, the system shows upcoming trips from today onward. If a date is provided, the system lists trips only for that date. This improves UX by allowing flexible discovery of available trips.

**Seat Locking**
SeatLock is designed to ensure only one user can hold a lock for a given seat on a given trip at a time. An expiry timestamp and an is_expired() helper function make it easy to release locks that are no longer valid.

## 9. Testing

Manual testing included route search scenarios (with and without date), verification of upcoming-trip fallback, seat lock uniqueness, booking creation with PNR generation, and Supabase database connectivity.

## 10. Challenges & Solutions

During deployment, issues related to dependency builds and cloud environment differences were addressed by pinning a stable Python version and aligning environment variables for Supabase connectivity. Cold-start behavior on free-tier hosting was mitigated using a keep-alive ping approach.

## 11. Future Enhancements

• Payment gateway integration (Razorpay/Stripe).

• Ticket generation (PDF) and email/SMS notifications.

• Real-time seat availability updates and lock countdown UI.

• Admin analytics dashboard (occupancy, popular routes, revenue).

• Cancellation and refund policy management.

## 12. Conclusion

The Bus Booking System demonstrates a complete route-to-booking workflow with solid data integrity guarantees through constraints and locking. It combines Django’s productivity with a cloud-managed PostgreSQL database on Supabase and cloud hosting on Render, making it suitable for extension into a production-grade booking platform.