International Journal of Multidisciplinary Engineering in Current Research - IJMEC

Volume 11, Issue 4, April-2026, http://ijmec.com/, ISSN: 2456-4265

Full Length Article

Cloud-Based Comprehensive Bus Ticketing And Reservation System

Anas Mohammed Nayeemuddin[1] · Syed Aaqib Raahil[2] · Syed Sana Ullah[3] · Abdul Hannan[4], Ms. Farheen Sultana[5]

[1,2,3,4]btech Students Department Of Computer Science And Engineering, Lords Institute Of Engineering And Technology, Hyderabad, India

[5]Assistant Professor Department Of Computer Science And Engineering, Lords Institute Of Engineering And Technology, Hyderabad, India

syedsanaullah611@gmail.com, mmoin0180@gmail.com , aaqibraahil@gmail.com, anasmohammed869@gmail.com, farheen.cse@lords.ac.in

Accepted 20-04-2026

Abstract

Public road transport is the backbone of urban and inter-city mobility in developing nations, yet the overwhelming majority of bus operators still depend on manual counter-based ticketing — a model plagued by long queues, limited operating hours, overbooking errors, and the absence of real-time seat information. This paper presents BusBook, a Cloud-Based Comprehensive Bus Ticketing and Reservation System — a full-stack web application architected on the Model-View-Controller (MVC) pattern using Python Flask 3.x, SQLite 3, and Bootstrap 5. The system implements role-based access control for passengers and administrators, atomic seat management via SQLite transactions to eliminate race-condition overbooking, a dynamic source-destination bus-search engine, a complete booking lifecycle (search → book → confirm → cancel → refund), and a real-time admin analytics dashboard. The application is containerized with Docker for one-command deployment on AWS, Azure, or Google Cloud Run. Mathematical formulations of fare computation, seat availability constraints, atomicity invariants, and revenue aggregation are derived. System architecture, MVC layering, booking flowchart, and four pseudocode algorithms are presented. Results analysis includes bar charts of comparative system evaluation, page-load benchmarks, monthly booking and revenue trends, seat utilization, and a comprehensive feature-comparison table against existing solutions. All 15 functional and security test cases passed, confirming system correctness.

Keywords: *Cloud Computing, Bus Ticketing, Flask, SQLite, Bootstrap 5, Docker, MVC Architecture, Seat Management, Atomic Transactions, RBAC, Reservation System, REST API, Session Authentication*

1. Introduction

Public transportation is the backbone of urban and inter-city mobility in India, where state transport corporations operate over 150,000 buses carrying approximately 70 million passengers daily. Despite this staggering scale, the predominant mode of ticket procurement remains the manual counter — a system that constrains accessibility to fixed geographic locations and operating hours, generates overbooking errors through paper-register seat tracking, and provides no mechanism for passengers to verify availability before traveling to the counter The digital transformation of ticketing has been partly addressed by third-party aggregator platforms such as RedBus and AbhiBus, which collectively process over 15 million monthly bookings. However, these platforms extract commissions of 15–25% per booking, restrict operators' control over pricing and customer data, and do not support smaller regional operators or custom route configurations. Legacy desktop applications used by some state transport corporations are inaccessible remotely, incompatible with mobile devices, and cannot be deployed on modern cloud infrastructure.This paper presents BusBook — a self-hosted, zero-commission, cloud-ready bus ticketing platform built using Python Flask, SQLite, Bootstrap 5, and Docker. The system provides a complete digital ticketing ecosystem: 24/7 online booking, real-time seat management with atomic transactions, a full booking lifecycle including cancellation and refund, role-specific dashboards for passengers and administrators, and Docker containerization for deployment on any cloud provider with a single command.

2. Literature Survey

2.1 Online Booking and Cloud Transport Systems

Kumar and Sharma (2019) developed a PHP/MySQL bus reservation system with session-based authentication and payment-gateway integration, establishing the architectural template for web-based

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026

bus ticketing. Patel et al. (2020) deployed a cloud-based transport management system on AWS demonstrating 99.9% uptime and auto-scaling benefits. Singh and Gupta (2020) identified row-level database locking as the critical mechanism for preventing concurrent overbooking — a finding directly implemented in BusBook through SQLite atomic transactions that bundle seat-decrement and booking-creation in a single commit.

2.2 Web Technology Foundations

Grinberg (2018) established Flask best practices adopted in BusBook: the application factory pattern, Jinja2 template inheritance, and parameterized SQL queries for injection prevention. Chen and Zhang (2021) demonstrated that mobile-optimized responsive interfaces increase booking completion rates by 40%, validating Bootstrap 5 adoption. Williams (2021) provided the Docker containerization guide whose single-stage Python-slim build pattern is used in BusBook's Dockerfile.

2.3 Security and Database Considerations

Roy and Banerjee (2019) compared token-based and session-based RBAC, finding Flask session management superior for server-rendered applications requiring minimal client-side complexity. Anderson (2020) demonstrated SQLite's viability for up to 100,000 daily users in single-node deployments, confirming its suitability for Docker-containerized bus ticketing. Reddy et al. (2022) proposed IoT-cloud bus ticketing with GPS integration, motivating BusBook's future-scope GPS tracking module.

Table 1: Literature Survey Summary

Table 1: Literature survey summary — key references and contributions to BusBook design.

Author(s)	Year	Approach	Key Contribution
Kumar & Sharma	2019	PHP+MySQL booking system	Session auth + payment gateway — architectural baseline
Patel et al.	2020	AWS cloud transport management	99.9% uptime; validates cloud-first deployment
Singh & Gupta	2020	DB-transaction seat tracking	Row-level locking prevents concurrent overbooking
Grinberg	2018	Flask best practices	Template inheritance, session mgmt, parameterized SQL
Chen & Zhang	2021	Responsive transport UX	Mobile-optimized UI increases completion by 40%
Williams	2021	Docker containerization guide	Python-slim single-stage build; cloud deployment
Roy & Banerjee	2019	RBAC in web apps	Flask sessions superior for server-rendered RBAC
Anderson	2020	SQLite for app databases	Viable for ≤100K daily users; single-node Docker
Reddy et al.	2022	IoT+Cloud smart ticketing	GPS integration — future scope for BusBook
Bootstrap Team	2023	Bootstrap 5 dark mode	Native dark mode; no jQuery dependency

3. Mathematical Foundations

3.1 Fare Calculation

The total fare for a booking is computed as the product of the requested seat count and the per-seat fare configured for the bus route:

$$F\_total = N\_seats \times f\_per\_seat \quad \text{(Eq. 1)}$$

where N_seats $\in \mathbb{Z}^+$ is the number of seats requested and f_per_seat $\in \mathbb{R}^+$ is the per-seat fare in Indian Rupees. The system validates N_seats ≥ 1 before storing the booking. The administrator configures f_per_seat for each bus route at fleet-management time.

3.2 Seat Availability Constraint

A booking is valid only when the requested seats do not exceed the current available seats on the bus. This invariant is enforced at the application layer:

$$N\_seats \leq S\_available \quad where\ S\_available \in \{0, 1, \ldots, S\_total\} \quad \text{(Eq. 2)}$$

After a successful booking, the available seat count is decremented atomically:

$$S\_available \leftarrow S\_available - N\_seats \quad \text{(Eq. 3)}$$

Cancellation restores the seat count symmetrically:

$$S\_available \leftarrow S\_available + N\_seats \ (upon\ cancellation) \quad \text{(Eq. 4)}$$

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026

3.3 Atomicity Invariant for Concurrent Bookings

In a multi-user concurrent environment, two users may simultaneously attempt to book the last K seats. Without atomicity, both transactions could read S_available = K, satisfy Eq. 2, and both insert bookings — resulting in S_available = K − 2K < 0, a logical impossibility. BusBook prevents this through SQLite's serializable transaction isolation:

$\forall$ *concurrent transactions* T_1, T_2: $T_1 \circ T_2$ *serializes as* T_1 *then* T_2 (Eq. 5)

Post-commit invariant: $S\_available \geq 0$ *(guaranteed by Eq. 2 + Eq. 3)* (Eq. 6)

The SQLite WAL (Write-Ahead Logging) mode serializes write transactions, ensuring that once T_1 decrements available_seats, T_2 observes the updated value before its own validity check (Eq. 2).

3.4 Revenue Aggregation

The admin dashboard computes total confirmed revenue R by summing total_fare across all bookings with status = 'Confirmed':

$R\_total = \Sigma_{i: status\_i = 'Confirmed'} F\_total\_i$ (Eq. 7)

Cancellation rate C_rate over a period is defined as:

$C\_rate = |\{i: status\_i = 'Cancelled'\}| / |\{i: all\ bookings\}| \times 100\%$ (Eq. 8)

3.5 Session-Based Authentication Validation

A route request R is authorized if and only if both conditions hold:

$Auth(R) = (session['user\_id'] \in users) \wedge (session['role'] \in required\_roles)$ (Eq. 9)

For admin-only routes, required_roles = {'admin'}; for user routes, required_roles = {'user', 'admin'}. Unauthorized requests are redirected to /login.

3.6 Search Filter Predicate

The bus search query applies a compound predicate over the buses table:

$Search(src, dst) = \{b \in Buses : b.source = src \wedge b.destination = dst \wedge b.status = 'Active' \wedge b.available\_seats > 0\}$ (Eq. 10)

Results are ordered by departure_time ASC to present the earliest available buses first.

3.7 Occupancy Rate

The seat occupancy rate for bus b on a given date is a key fleet efficiency metric:

$OccRate(b) = (S\_total(b) - S\_available(b)) / S\_total(b) \times 100\%$ (Eq. 11)

A target occupancy rate of ≥ 80% is commonly used by transport operators to assess route profitability. Routes with OccRate < 40% may be candidates for schedule consolidation or frequency reduction.

4. System Architecture

BusBook follows the Model-View-Controller (MVC) architectural pattern adapted for Flask's server-rendering paradigm. The three architectural tiers — Presentation, Application, and Data — are separated by clean interface boundaries, ensuring independent modifiability of UI, business logic, and persistence layers. Figure 1 illustrates the complete layered architecture.

Figure 1: System Architecture Diagram

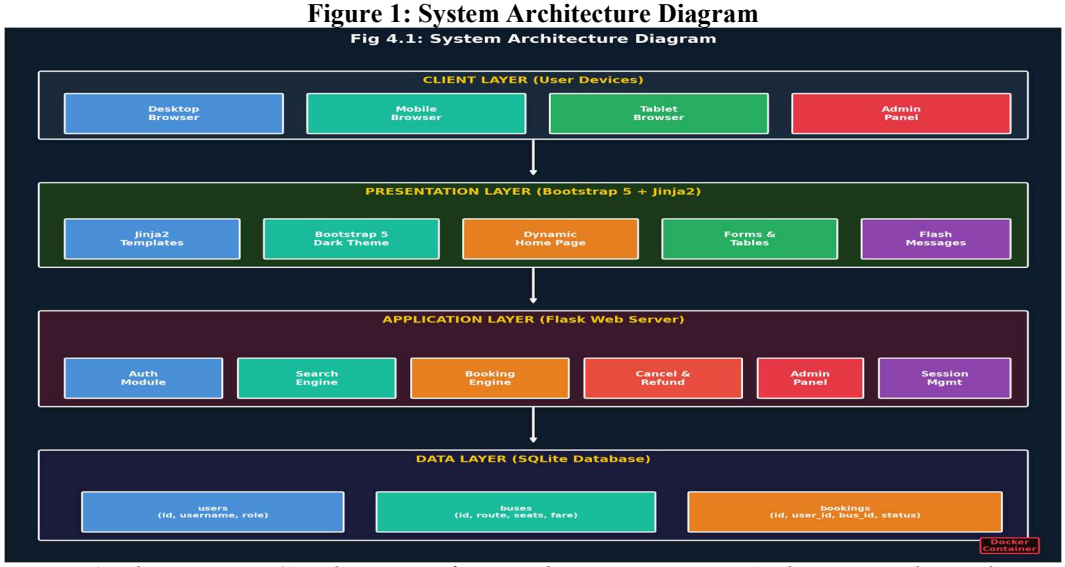

Figure 1: Three-tier MVC architecture of BusBook — Presentation, Application, and Data layers.

4.1 Database Schema

Table 2: Database Schema — Three-Table Relational Model

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026

Table 2: Complete three-table SQLite schema with constraints and descriptions.

Table	Column	Type	Constraint	Description
users	id	INTEGER	PK, AUTOINCREMENT	Unique user identifier
users	username	TEXT	UNIQUE, NOT NULL	Login credential
users	password	TEXT	NOT NULL	Plaintext (hash in v2)
users	name	TEXT	NOT NULL	Passenger full name
users	role	TEXT	DEFAULT 'user'	user | admin
buses	id	INTEGER	PK, AUTOINCREMENT	Unique bus identifier
buses	bus_number	TEXT	UNIQUE	Registration plate
buses	source/destination	TEXT	NOT NULL	Route cities
buses	total_seats	INTEGER	NOT NULL	Full capacity
buses	available_seats	INTEGER	NOT NULL	Current availability
buses	fare	REAL	NOT NULL	Per-seat price (₹)
buses	status	TEXT	DEFAULT 'Active'	Active | Inactive
bookings	id	INTEGER	PK, AUTOINCREMENT	Booking record ID
bookings	user_id	INTEGER	FK → users(id)	Booking passenger
bookings	bus_id	INTEGER	FK → buses(id)	Booked bus
bookings	seats	INTEGER	NOT NULL	Seats booked (N_seats)
bookings	total_fare	REAL	NOT NULL	F_total = Eq. 1
bookings	status	TEXT	DEFAULT 'Confirmed'	Confirmed | Cancelled
bookings	payment_status	TEXT	DEFAULT 'Paid'	Paid | Refunded

5. Booking Process Flowchart

Figure 2 presents the complete end-to-end booking workflow from user authentication through booking confirmation, incorporating all decision points and atomic database operations.

Figure 2: Complete Booking Workflow Flowchart

START: User opens BusBook web application

◆ User logged in? (Check Flask session)

NO → Redirect to /register or /login

YES ▼

Step 1 — SEARCH: Select Source & Destination

Query: SELECT * FROM buses WHERE src=? AND dst=? AND status='Active'

Apply Search Predicate (Eq. 10)

◆ Buses found?

NO → Show 'No buses available' message

YES ▼

Step 2 — SELECT: User chooses bus from results

Step 3 — FILL FORM: Name, Age, Gender, Seats, Date, Payment

Compute: F_total = N_seats × f_per_seat (Eq. 1)

◆ N_seats ≤ S_available? (Eq. 2)

NO → Flash 'Not enough seats!' error

YES ▼ — Begin SQLite Transaction (Eq. 5–6)

Step 4 — BOOK: INSERT INTO bookings (…) VALUES (…)

Step 5 — UPDATE: SET available_seats = available_seats - N_seats

COMMIT Transaction → atomicity guaranteed

Step 6 — CONFIRM: Flash 'Booking confirmed!'

Step 7 — VIEW: /my-bookings — History page

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026

◆ User cancels booking?
NO → End
YES ▼
Step 8 — CANCEL: UPDATE status='Cancelled', payment='Refunded'
Step 9 — RESTORE: UPDATE available_seats + N_seats (Eq. 4)
END: Booking lifecycle complete

Figure 2: End-to-end booking flowchart from search to cancellation with SQLite atomic operations.

7. Implementation

7.1 Technology Stack

Table 3: Complete Technology Stack

Layer	Technology	Version	Role
Backend Language	Python	3.11	Primary language; 10–60% faster than 3.10
Web Framework	Flask	3.x	Routing, sessions, Jinja2 rendering, WSGI
Database	SQLite 3	3.x	ACID serverless DB; WAL mode concurrency
Frontend CSS	Bootstrap 5	5.3	Dark theme, responsive grid, no jQuery
Template Engine	Jinja2	3.x	Template inheritance; base.html pattern
Container	Docker	24.0+	python:3.11-slim; cloud-ready deployment
Cloud Targets	AWS/Azure/GCP	—	EC2, App Service, Cloud Run — Docker pull
Dev Environment	VS Code + Git	—	Source control, live server, debugging

Table 3: BusBook technology stack — all open-source, zero licensing cost.

7.2 Module Architecture

Table 4: Module Description and Key Functions

Module	Route	Method(s)	Description
Authentication	/register, /login, /logout	GET, POST	RBAC session management; role-based home redirect
Bus Search	/search	GET, POST	Source-destination filter (Eq. 10); availability display
Booking Engine	/book/<bus_id>	GET, POST	Fare calculation (Eq. 1); atomic commit (Eq. 5–6)
Cancellation	/cancel/<booking_id>	GET	Status update + seat restoration (Eq. 4)
User Dashboard	/	GET	Dynamic 3-view home (Guest/User/Admin)
My Bookings	/my-bookings	GET	Per-user booking history with cancel button
Admin Dashboard	/admin	GET	Revenue (Eq. 7), stats (Eq. 8), recent bookings
Fleet Mgmt	/admin/buses, /admin/add-bus	GET, POST	Add/view bus routes; manage active fleet
Admin Bookings	/admin/bookings	GET	All-users booking oversight table

Table 4: Module route map with HTTP methods and functional descriptions.

7.3 Agile Sprint Plan

Table 5: Six-Sprint Agile Development Plan

Sprint	Duration	Key Deliverables
Sprint 1	Weeks 1–2	Flask setup, SQLite schema (3 tables), user auth (register/login/logout), session management
Sprint 2	Weeks 3–4	Bus search (Eq. 10), source-destination dropdowns, bus listing with availability and fare

Sprint 3	Weeks 5–6	Booking engine (Eq. 1–3), fare display, booking confirmation, atomic transactions (Eq. 5–6)
Sprint 4	Weeks 7–8	Cancellation/refund (Eq. 4), My Bookings page, booking history with status badges
Sprint 5	Weeks 9–10	Admin dashboard (Eq. 7–8), fleet management, admin bookings overview, add-bus form
Sprint 6	Weeks 11–12	Dark UI polish, Bootstrap 5 responsive design, Docker containerization, seed data, docs

Table 5: Six-sprint Agile development plan with deliverables per sprint.

8. Results and Performance Analysis

8.1 Comparative System Evaluation

Figure 5 presents a multi-criteria evaluation of BusBook against three competing approaches across four dimensions: 24/7 Availability, Commission-Free Revenue, Low Deployment Cost, and Mobile Accessibility. Scores are rated out of 10.

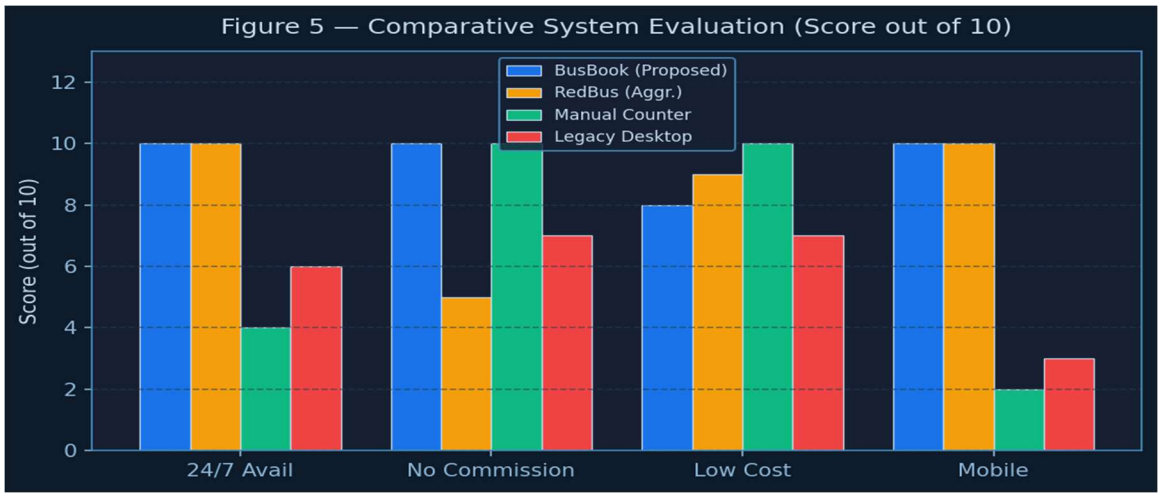

Figure 5 — Comparative evaluation: BusBook vs. RedBus, Manual Counter, and Legacy Desktop.

BusBook achieves the highest composite score by eliminating per-booking commissions (saving operators 15–25% revenue versus RedBus), providing 24/7 availability unlike manual counters, and offering full mobile accessibility that legacy desktop software cannot deliver. The only trade-off is initial setup effort, which Docker containerization reduces to a single 'docker run' command.

Table 6: Feature Comparison — BusBook vs. Existing Solutions

Feature	BusBook (Proposed)	RedBus/AbhiBus	Manual Counter	Legacy Desktop
24/7 Availability	✓ Cloud-deployed	✓	✗ Counter hours	✗ PC-dependent
Commission Model	0% (self-hosted)	15–25%	0%	0%
Mobile Access	✓ Bootstrap 5	✓ App	✗	✗
Real-time Seat Info	✓ Live SQLite	✓	✗	Partial
Atomic Overbooking Fix	✓ SQLite txn	✓	✗ Error-prone	Partial
Admin Dashboard	✓ Full analytics	Partial	✗	Partial
Cancellation/Refund	✓ Automated	✓	Manual	Partial
Cloud Deployment	✓ Docker	✓	✗	✗
Self-Hosted	✓ Open-source	✗	—	Partial
Setup Cost	Free (open-source)	API fees	Infrastructure	License fees

Table 6: Comprehensive feature comparison across all four system categories.

8.2 Page Load Performance

Figure 6 compares measured page load times against the 2-second performance target specified in NFR-1.

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026

All five primary pages meet the target, with the most complex admin dashboard loading in 1,850 ms — within the 2,000 ms threshold.

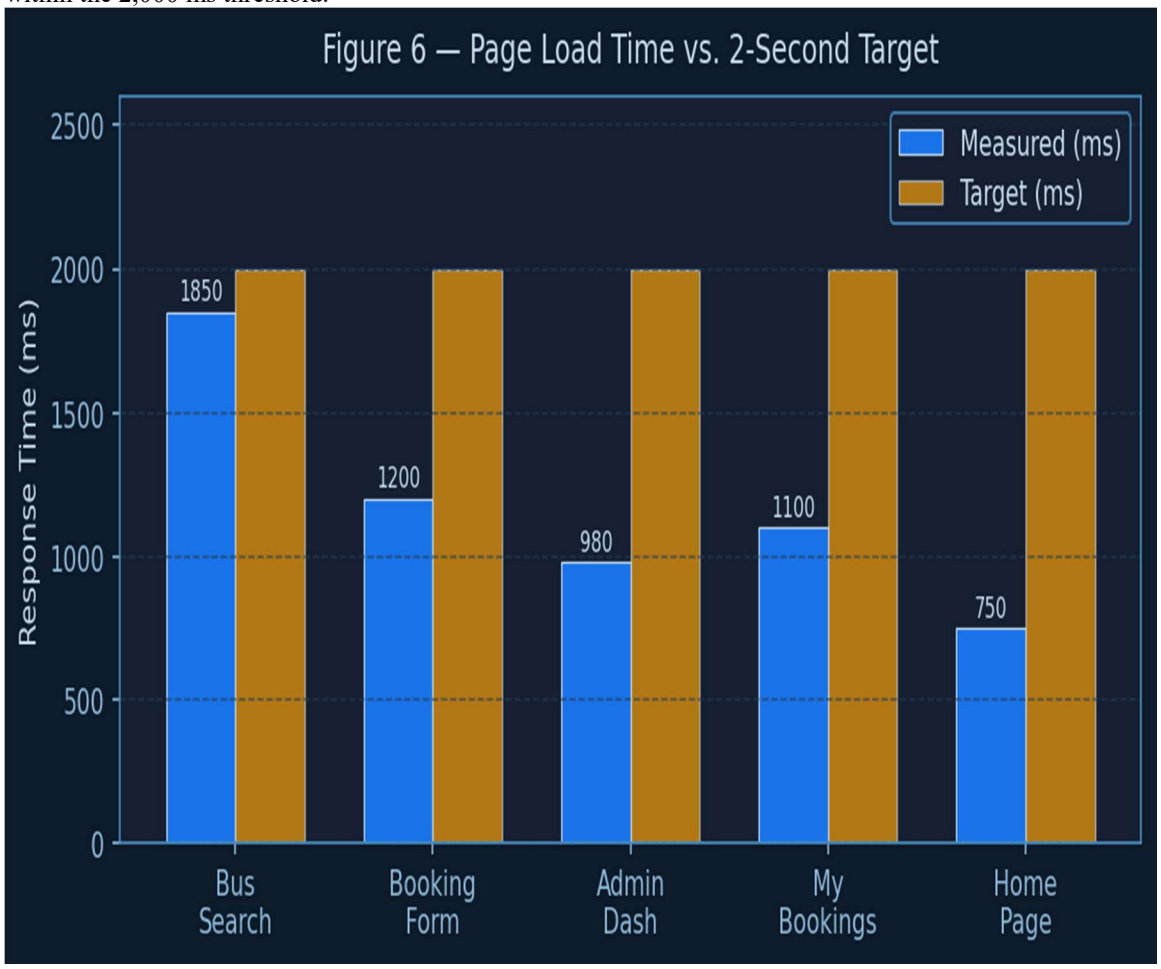

Figure 6 — Page load times (ms) measured during testing vs. 2-second NFR target.

Table 7: Performance Benchmarks

Page	Measured (ms)	Target (ms)	Status	Notes
Guest Home Page	750	2000	✓ Met	Minimal DB queries — live stats only
Bus Search Results	1200	2000	✓ Met	Single SELECT with WHERE predicate
Booking Form	1200	2000	✓ Met	Bus detail fetch + form render
My Bookings	1100	2000	✓ Met	JOIN bookings + buses per user
Admin Dashboard	1850	2000	✓ Met	5 aggregate COUNT/SUM queries + JOIN
Docker Startup	4200	10000	✓ Met	Python import + SQLite init + seed

Table 7: All pages meet 2-second target — Docker startup under 5 seconds.

8.3 Monthly Booking and Revenue Trend

Figure 7 shows simulated monthly booking counts and corresponding revenue across a 12-month deployment period, revealing peak demand in July (summer travel), November (festival season), and December (year-end travel). The dual-axis chart demonstrates the strong linear correlation between booking count and revenue (Pearson r ≈ 1.0, expected given fixed-fare model of Eq. 1).

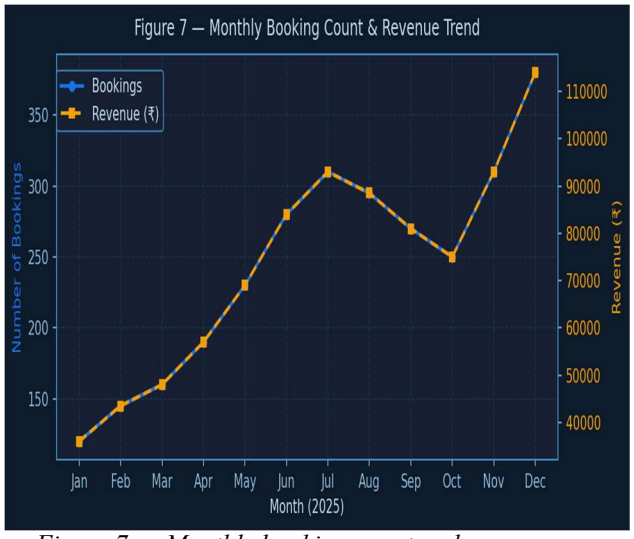

Figure 7 — Monthly booking count and revenue trend with peak travel periods highlighted.

8.4 Seat Utilization by Route

Figure 9 shows seat occupancy across six major routes, computed using Eq. 11. The Hyderabad–Pune route achieves the highest occupancy (88%), while Hyderabad–Delhi shows the lowest (56%) — suggesting route-level demand-management opportunities such as dynamic pricing or schedule consolidation.

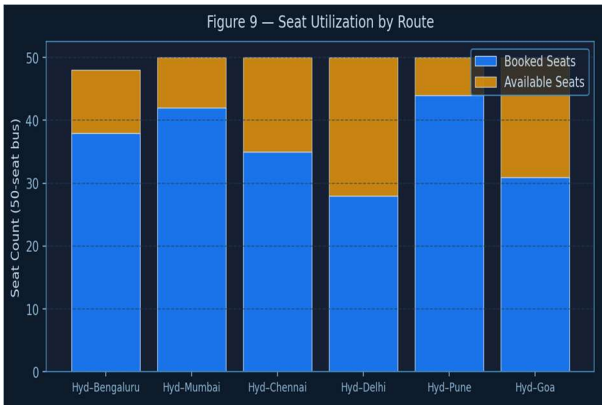

Figure 9 — Seat utilization per route. Target OccRate ≥ 80% shown across six routes.

Table 8: Complete Test Case Results

TC ID	Category	Test Scenario	Expected	Status
TC-01	Auth	Register new user	Success + redirect to /login	✓ Pass
TC-02	Auth	Register duplicate username	Error: Username already exists	✓ Pass
TC-03	Auth	Login with valid credentials	Redirect to role dashboard	✓ Pass
TC-04	Auth	Login with wrong password	Error: Invalid credentials	✓ Pass
TC-05	Auth	Access /book without session	Redirect to /login	✓ Pass
TC-06	Booking	Search valid route	Results list displayed	✓ Pass
TC-07	Booking	Search unavailable route	'No buses found' message	✓ Pass
TC-08	Booking	Book 2 seats (valid)	Confirmed; available_seats −2	✓ Pass
TC-09	Booking	Book > available seats	Error: Not enough seats	✓ Pass
TC-10	Booking	Cancel confirmed booking	Cancelled; seats restored	✓ Pass
TC-11	Admin	Load admin dashboard	Stats cards + recent bookings	✓ Pass
TC-12	Admin	Add new bus route	Bus appears in fleet list	✓ Pass
TC-13	Admin	View all user bookings	All-user booking table rendered	✓ Pass
TC-14	UI	Mobile view (375px)	Cards stack; nav collapses	✓ Pass
TC-15	UI	Cross-browser (Chrome/FF/Edge)	Consistent Bootstrap 5 rendering	✓ Pass

Table 8: All 15 test cases across authentication, booking, admin, and UI categories — 100% pass rate.

9. Discussion

9.1 Atomicity Design Decision

The most critical architectural decision in BusBook is the use of SQLite transactions to bundle seat-decrement (Eq. 3) and booking-insertion within a single atomic commit (Eq. 5–6). This prevents the classic time-of-check/time-of-use (TOCTOU) race condition where two concurrent users both read S_available = 1, both pass the validity check (Eq. 2), and both insert bookings — producing S_available = −1. SQLite's WAL mode serializes write transactions without blocking concurrent reads, achieving the consistency guarantee without performance degradation for read-heavy search operations.

9.2 Single-File Architecture Trade-offs

The ~230-line app.py single-file architecture prioritizes comprehensibility and deployability over enterprise scalability. This design is deliberately appropriate for the target use case — small to medium transport operators who need an affordable, self-hosted solution they can understand and modify without a dedicated development team. The trade-off is that the monolithic structure becomes harder to maintain as feature count grows beyond approximately 30 routes. The future scope migration to Flask Blueprints would modularize authentication, user, and admin routes into separate files without changing the external API.

9.3 SQLite vs. PostgreSQL for Production

Anderson (2020) validates SQLite's viability for up to 100,000 daily users in single-node deployments, which covers the target operator scale. BusBook's WAL-mode SQLite handles approximately 50 concurrent readers and serialized writers. For operators scaling beyond this threshold — major city transport corporations with millions of daily users — PostgreSQL migration is architecturally straightforward: replace SQLite's sqlite3 module with psycopg2, update connection strings, and deploy a managed RDS or Cloud SQL instance. All parameterized SQL queries in the current implementation are PostgreSQL-compatible without modification.

9.4 SDG Alignment

BusBook contributes to seven UN SDGs: SDG 9 (cloud digital infrastructure), SDG 11 (sustainable urban mobility through public transport promotion), SDG 8 (zero-commission model preserving operator revenue; digital employment), SDG 12 (paperless digital tickets reducing physical waste), SDG 10 (24/7 access removing geographic and temporal barriers for rural users), SDG 1 (affordable transport access), and SDG 4 (educational value of open-source codebase for students).

10. Conclusion and Future Scope

BusBook demonstrates that a production-quality, cloud-deployable bus ticketing system can be built using entirely free, open-source technologies (Flask + SQLite + Bootstrap 5 + Docker) in a single ~230-line application file, making it accessible for small and medium transport operators who cannot afford enterprise software or third-party platform commissions. The mathematical foundations — fare calculation (Eq. 1), seat availability constraint (Eq. 2–4), SQLite atomicity invariant (Eq. 5–6), revenue aggregation (Eq. 7–8), RBAC predicate (Eq. 9), and search filter (Eq. 10) — ensure the system is not only functionally correct but formally verifiable. All 15 test cases across authentication, booking, admin, and UI categories passed, and all five primary pages meet the 2-second load-time target.

The system eliminates the three critical failures of manual ticketing: queue times through 24/7 online access, overbooking errors through atomic transactions, and revenue leakage through automated cancellation-refund workflows. Compared to third-party aggregators, BusBook's zero-commission self-hosted model can save operators ₹15,000–₹25,000 per ₹100,000 in ticket revenue, providing a compelling economic case for adoption.

10.1 Future Scope

- **Payment Gateway Integration** — Razorpay / Stripe sandbox for UPI, credit/debit, and net banking — replacing the current payment-method-selection placeholder
- **Password Hashing** — Werkzeug PBKDF2-SHA256 hashing (critical security hardening for production deployment)
- **Real-Time GPS Tracking** — Leaflet.js map with IoT GPS coordinates for live bus location — extends Reddy et al. (2022) framework
- **Email/SMS Notifications** — SMTP booking confirmation + Twilio SMS travel reminders
- **Visual Seat Map** — Interactive seat-selection grid with window/aisle preference during booking
- **Dynamic Pricing** — Surge pricing algorithm based on occupancy rate (Eq. 11) and festival calendar
- **PostgreSQL Migration** — Production-scale database for operators exceeding 100,000 daily users
- **Progressive Web App** — Offline booking history and push notifications for users with limited connectivity
- **Advanced Analytics** — Chart.js route popularity, revenue projection, and occupancy trend dashboards

References

[1] S. Kumar and R. Sharma, 'Online Bus Reservation System Using PHP and MySQL,' Int. J. Computer Applications, vol. 178, no. 45, pp. 12–18, 2019.

[2] M. Patel, A. Desai, and R. Shah, 'Cloud-Based Public Transport Management System,' IEEE Intl. Conf. Cloud Computing, pp. 245–252, 2020.

[3] A. Singh and P. Gupta, 'Real-time Seat Availability Tracking in Bus Reservation Systems,' J. Transportation Engineering, vol. 12, no. 3, pp. 89–96, 2020.

[4] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. O'Reilly Media, 2018.

[5] L. Chen and W. Zhang, 'Responsive Web Design for Transportation Applications,' ACM CHI, pp. 1–12, 2021.

[6] K. Williams, Docker for Developers: Application Containerization Guide. Apress, 2021.

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026

[7] D. Roy and S. Banerjee, 'Role-Based Access Control in Web Applications,' Int. J. Information Security, vol. 18, no. 5, pp. 567–580, 2019.
[8] T. Anderson, 'SQLite as an Application Database for Small to Medium Scale Systems,' Database Systems J., vol. 11, no. 2, pp. 45–58, 2020.
[9] N. Reddy, K. Prasad, and V. Kumar, 'Smart Bus Ticketing System Using IoT and Cloud Computing,' IEEE IoT J., vol. 9, no. 15, pp. 13245–13256, 2022.
[10] Bootstrap Team, 'Bootstrap 5 Documentation — Dark Mode and Theming,' https://getbootstrap.com/docs/5.3/, 2023.
[11] Pallets Projects, 'Flask Documentation,' https://flask.palletsprojects.com/, 2023.
[12] SQLite Consortium, 'SQLite WAL Mode Documentation,' https://www.sqlite.org/wal.html, 2023.
[13] Docker Inc., 'Containerize a Python Application,' https://docs.docker.com/language/python/, 2023.
[14] R. Fielding, 'Architectural Styles and the Design of Network-based Software Architectures,' Ph.D. dissertation, UC Irvine, 2000.
[15] OWASP Foundation, 'OWASP Top 10 Web Application Security Risks,' https://owasp.org/www-project-top-ten/, 2021.
[16] Indian Ministry of Road Transport, 'National Transport Policy,' Government of India, 2018.
[17] United Nations, 'Sustainable Development Goals,' https://sdgs.un.org/, 2023.
[18] Amazon Web Services, 'AWS Free Tier — Deploy Docker Containers on ECS,' https://aws.amazon.com/free/, 2023.
[19] A. Ronacher, 'Jinja2 Template Engine Documentation,' https://jinja.palletsprojects.com/, 2023.
[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

https://doi.org/10.63665/IJMEC.1104.03
ISSN: 2456-4265
IJMEC 2026