

Digitalizing Local Taxis: A Code-Level Case Study of a Web-Based Booking System

Afzal Ahmed Shaikh under the guidance of Miss. Maya Nair

Department of Computer Science, SIES College of Arts, Science & Commerce (Autonomous) Mumbai, India

Email: Afzalfmcs2526107@ascs.sies.edu.in

Email: mayan@sies.edu.in

Abstract:

Local taxi services may depend on phone calls, street-hailing and manual records, making it difficult to maintain a shared view of vehicle availability and booking status. This paper presents a code-level case study of an educational PHP and MySQL cab-booking prototype considered as a starting point for digitalizing local taxi operations. The study examines the supplied source files, SQL schema and main passenger, driver and administrator workflows. The system uses a browser interface, PHP request handlers and a MySQL database with six principal tables. The reviewed code supports passenger registration, cab search, pickup and drop-off entry, booking history, driver-side status updates and administrative maintenance of cabs and categories. The review also identifies important limits: no live GPS tracking, automated fare calculation or online payment is implemented, and no user or performance study is supplied. Security inspection finds inconsistent use of parameterized queries and MD5-based password handling. The contribution is a transparent artifact analysis and improvement roadmap, not a claim of a novel dispatch algorithm or proven service impact.

Keywords — local taxi services, digitalization, web application, PHP, MySQL, booking management, case study.

I. INTRODUCTION

Local taxi services provide short-distance transport in cities and towns, but the booking process can remain fragmented across phone calls, street-hailing, personal contacts and paper records. When vehicle availability, pickup details and trip status are maintained separately, customers may have difficulty confirming a ride and operators may find it harder to review previous requests. A small web application can centralize these records without requiring the full feature set of a large ride-hailing platform.

Digital ride services have changed how passengers discover and request vehicles, but their transport effects are mixed. Rayle et al. compared ridesourcing, taxi and transit use in San Francisco and reported differences in use patterns and waiting experiences, while noting that many ridesourcing journeys replaced modes other than taxis [2]. A broader review by Tirachini found that ride-hailing can improve access and convenience, but its effects on congestion, public transport and total vehicle travel are context dependent [3]. These findings caution against equating digital booking with automatically better urban mobility.

This paper examines a PHP/MySQL cab-booking prototype as a bounded case study for local taxi digitalization. The reviewed artifact is based on an educational codebase publicly described by CodeAstro [1], rather than a new dispatch algorithm developed in this study. The paper contribution is a code-level mapping of its roles, workflows and database, together with a focused review of implementation gaps that must be addressed before real-world use.

The study asks three questions: (1) which passenger, driver and administrator workflows are represented in the code; (2) how are vehicle, customer and booking records organized in the database; and (3) which functional and security limitations constrain the prototype as a local service platform? The study does not claim measured improvements in waiting time, income, safety or user satisfaction.

The contribution is practical and bounded. It documents the link between the visible interface, PHP action handlers and stored records; summarizes the main entities in the database; and identifies a short list of implementation risks that should be addressed before field use. It does not propose a new dispatch algorithm or attempt to compare taxi services statistically. This distinction is important because a working web form demonstrates a software capability, but does not on its own demonstrate faster trips, lower costs or improved passenger trust.

II. RELATED WORK

A. Digital Booking and User Experience

Web-based booking can make vehicle options and booking records easier to access, but interface-level convenience is only one part of service quality. Rayle et al. used intercept surveys and comparative trip data to study ridesourcing alongside taxis and transit [2]. Their findings are relevant to booking-system design because they show that mode choice and service experience depend on more than the existence of an app. The present case is narrower: it reviews a booking and administration workflow and does not collect user surveys or compare travel modes.

B. Mobility and Local Operations

Tirachini's review describes both possible benefits of app-based matching and potential negative effects, including additional motorized travel and competition with public transport in some settings [3]. The International Transport Forum also discusses how shared-use mobility changes curb access, pickup and drop-off activity, and city management needs [4]. These studies motivate a cautious approach: a local booking portal should first improve the reliability of records and coordination, while larger claims about congestion or sustainability require separate field evidence.

C. Research Gap and Scope

The prototype should be understood as a lightweight booking and management tool, not as a complete mobility platform. It does not implement live vehicle coordinates, map-based matching, dynamic pricing or integrated payments. Its relevance lies in showing how basic booking records and role-specific pages can be organized, and in identifying additional work needed to build a safer and empirically evaluated service. Unlike a transport-impact study, this work focuses on the software artifact and the integrity of its implementation.

The resulting research gap is therefore not the absence of commercial ride-hailing products. It is the need to understand what a small, locally run booking portal actually supports and where its implementation would need strengthening. For a local operator, reliable records and simple booking coordination may be a reasonable first step, but any claim about service quality, equitable access or environmental impact would require a separate evaluation involving users and operational data.

III. CASE STUDY AND METHODOLOGY

A. Artifact and Inspection Procedure

The case artifact is the supplied CabBooking-PHP source archive and its SQL dump, cbsphp.sql. The implementation uses PHP pages and classes, a MySQL database, browser-side JavaScript, and an Apache-based local deployment model. The analysis followed four steps: (1) inventory the main pages and role-specific folders; (2) trace forms to PHP action handlers; (3) inspect table definitions, identifiers and preloaded records in the SQL dump; and (4) review query construction and password-handling patterns for security concerns.

The review was conducted as static code and schema inspection. No interviews, survey responses, GPS traces, transaction logs from a live service, or controlled performance measurements were supplied. Functional statements therefore describe behavior visible in the source and database structure; they should not be read as evidence that every workflow has passed runtime testing on all supported PHP and MySQL versions. The reviewed project is treated as an educational prototype, and the existing source provenance is acknowledged in [1].

To keep the review traceable, observations were grouped into three categories: pages and forms visible in the source, database tables and bundled seed records, and implementation patterns related to correctness or security. A capability was described as represented only when a relevant page, handler or schema field could be identified in the

supplied files. A missing feature means it was not found in the reviewed artifact; it does not mean the feature could never be added. The row counts below are counts from the included SQL dump, not observations of a live service.

B. Technology Stack and Local Execution

The client interface is built with HTML and CSS and uses JavaScript, jQuery, Bootstrap-related components and AJAX for selected interactions. PHP page handlers and classes process requests. The database wrapper uses MySQLi to connect to the database named cbsphp. In a local demonstration environment, Apache serves the PHP application while MySQL stores persistent records; XAMPP provides a convenient local setup for these services. The architecture does not require a separate mobile application.

A typical request begins with an event in the browser, such as submitting a booking form. A page-level JavaScript handler can prevent a full-page form submission and send the form values to a PHP endpoint. The PHP layer then checks the active session, processes the action and accesses MySQL through the application classes. A response is returned to the browser, where the interface can show a message or redirect to another page. This request pattern separates presentation from data operations, but important decisions must still be enforced on the server.

IV. SYSTEM ARCHITECTURE AND DATA MODEL

A. Application Architecture

The application follows a conventional browser-server-database arrangement. The browser renders HTML and CSS and loads JavaScript libraries for interface interactions. PHP pages and classes process submitted forms and access the database through the MySQLi interface. Configuration values identify the database server, account and database name. The reviewed DBConnection class creates a MySQLi connection from those configured values.

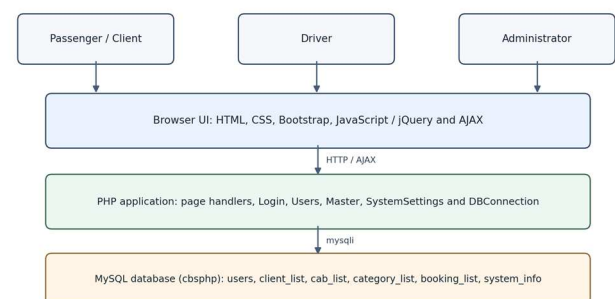


Fig. 1. Logical architecture of the reviewed cab-booking application.

Fig. 1 summarizes the logical flow. Passengers, drivers and administrators use role-specific pages in the same web application. Browser-side scripts can submit forms with AJAX to PHP handlers; PHP performs the requested database operation and returns a response. This separation keeps interface events, server-side processing and persistent records conceptually distinct. Browser-side validation alone, however, cannot replace server-side authorization or validation.

B. Principal Database Tables

The SQL dump defines six principal tables. The users table stores system-user account information; client_list stores passenger profiles; cab_list records vehicle and driver details;

category_list stores vehicle categories; booking_list stores pickup and drop-off text, a client identifier, a cab identifier and a status; and system_info contains basic application settings. Table I summarizes their roles.

TABLE I. PRINCIPAL DATABASE TABLES

Table	Primary purpose
users	System-user and administrator account data.
client_list	Passenger profile, contact and account information.
cab_list	Cab registration, model and driver details.
category_list	Cab categories and descriptions.
booking_list	Booking reference, client/cab IDs, pickup/drop-off and status.
system_info	Basic application name, logo and settings.

The schema links each booking to a client through client_id and to a selected vehicle through cab_id; each cab is associated with a category through category_id. Booking status is represented by numeric values, as listed in Table II. These fields provide a compact record of the booking lifecycle, but consistent server-side validation and database constraints are needed to keep transitions and relationships reliable.

The data model is easy to understand because the booking is the central transaction: it refers to the requesting client and the selected cab, while the cab points to a category. That design can support basic booking history and status views. However, identifiers alone do not enforce valid relationships unless the database or application checks them. For instance, a booking should not reference a removed client or nonexistent cab, and status changes should follow permitted transitions rather than accepting arbitrary values from a request.

TABLE II. BOOKING STATUS VALUES IN THE REVIEWED CODE

Value	Meaning
0	Pending
1	Driver confirmed
2	Picked up
3	Dropped off
4	Cancelled

The examined dump defines primary keys and some indexes, but does not declare explicit foreign-key constraints for the client, cab and category references. Referential consistency therefore appears to depend on application behavior rather than database-enforced relationships. A production version should add appropriate foreign keys, index frequent lookups and define deletion rules so that removing a client or cab cannot leave inconsistent booking records.

V. IMPLEMENTED WORKFLOWS AND FINDINGS

A. Passenger Workflow

The client-facing pages provide registration and login, an available-cab listing, a booking form and a booking-history view. The availability query joins cab_list to category_list to display each cab's category. It excludes cab identifiers associated with bookings whose status is pending, driver-confirmed or picked up. A booking form collects pickup and drop-off locations and sends the chosen cab and location data to the save_booking action in classes/Master.php. The backend sets the client identifier from the active session when a new booking is created and stores a booking reference code.

A client can view booking details and cancel a booking that is still pending through the status-update handler. The booking lifecycle makes the current state visible to the user, but the application is not a real-time dispatch engine. The reviewed code does not show map-based pickup coordinates, live location updates or fare computation. The destination and pickup values are recorded as user-provided text rather than verified geographic coordinates.

B. Driver and Administrator Workflow

The driver area includes account pages and a booking view that can update trip status. The administrative area provides pages for managing cabs, categories, clients, system users and system settings. The Master class contains actions such as saving or deleting a cab, saving or deleting a category, creating or updating a booking, and updating booking status. These operations support a central record-keeping workflow; they do not by themselves guarantee that a cab is physically available or that a driver has reached the pickup location.

The separation between passenger, driver and administrator pages is useful for navigation and role-specific tasks, but role separation must be enforced by server-side checks. Hiding an admin link or using a separate page does not prevent a user from attempting to call an endpoint directly. A deployment review should therefore test direct URL access, booking ownership, and whether each role can modify only the records permitted by the intended workflow.

C. Data-Dump Observations and Implementation Defect

The bundled SQL dump contains demonstration records: 17 client_list rows, 14 cab_list rows, three category_list rows, 23 booking_list rows, three users rows and five system_info rows. These are seed records included with the project archive, not a representative sample of actual local taxi demand. They confirm that the principal tables contain examples, but cannot be used to estimate booking frequency, utilization, response time or customer outcomes.

The code review identified a booking-reference generation concern: the booking creation method checks cab_list.reg_code while searching for a booking-reference candidate, rather than checking booking_list.ref_code. This mismatch should be corrected and tested because booking references are expected to identify bookings consistently. Availability queries and status updates should also be exercised using concurrent and repeated requests before the application is used outside a demonstration environment.

The observations suggest that the next development step should be verification rather than adding many visible features. For example, two users could attempt to reserve the same cab at nearly the same time, or a client could attempt to change a booking after its status has moved beyond pending. The current code review does not establish how the application behaves under all such conditions; these are specific scenarios that should be tested and documented before any field pilot.

TABLE III. FUNCTIONAL COVERAGE AND CURRENT BOUNDARIES

Role / area	Observed support	Not established
Passenger	Register/login, list cabs, create/view booking, cancel pending request.	GPS pickup, fare calculation, online payment.
Driver	Access driver pages and	Live tracking or verified

Role / area	Observed support	Not established
	update booking/trip status.	arrival.
Administrator	Manage cab, category, client, user and settings records.	Audited production-grade operations.
System	PHP/MySQL records, role-oriented pages, AJAX interactions.	Load-tested performance or user-study results.

VI. SECURITY, PRIVACY AND LIMITATIONS

The prototype uses MySQLi, and the administrator login method includes a prepared statement. However, other methods construct SQL statements by inserting request values into query strings, including booking and status-update paths. Parameterized queries are a standard way to keep input data separate from SQL command structure [5], [6]. All user-controlled values should be validated and handled through prepared statements; output should also be escaped when rendered in HTML.

The inspected authentication code uses MD5 to transform passwords before storing or checking them. MD5 is not suitable for modern password storage. OWASP recommends slow, adaptive password-hashing algorithms such as Argon2id, bcrypt or PBKDF2, and PHP provides `password_hash()` and `password_verify()` for this purpose [7]. A migration requires a planned compatibility path; simply replacing the hash function without updating verification could prevent existing accounts from signing in.

The database includes customer contact details and driver contact/address fields, so a deployed version would need access controls, a clear retention policy, encrypted transport, secure backups and protection against unauthorized viewing or export. The prototype also lacks live GPS tracking, map integration, automated fare calculation, online payment, notification delivery and measured usability or performance evaluation. These features are outside the reviewed implementation and should not be represented as existing capabilities.

The review also recommends checking server-side input validation, output encoding, session expiration, logout behavior, role checks and configuration permissions. These are review priorities rather than a claim that every listed weakness was demonstrated in a runtime exploit. The system should be tested with malformed values and unauthorized requests in a controlled local environment. Any deployment using real passenger information should minimize collected data, limit administrative access and use secure transport rather than the default local-development configuration.

VII. PROPOSED VERIFICATION CHECKLIST

A concise verification plan can turn the static review findings into testable requirements. Table IV lists representative checks and expected behavior. The table is a proposed plan only: none of these tests was run as part of the present code-level analysis, so no pass rate or measured reliability is claimed.

TABLE IV. PROPOSED FUNCTIONAL AND SECURITY CHECKS

Test area	Scenario	Expected behavior
Registration	Valid and incomplete client details.	Accept valid data; reject missing or invalid fields.
Authentication	Valid and invalid login credentials.	Authenticate valid users and reject invalid credentials.
Availability	Cab has pending/active booking.	Apply the documented status-based availability rule.

Test area	Scenario	Expected behavior
Booking	Submit pickup, drop-off and selected cab.	Store booking linked to the signed-in client and cab.
Status control	Cancel pending and non-pending bookings.	Allow only permitted status transitions.
Access control	Open another user's booking or protected URL.	Deny access not authorized for the current role.
SQL/input safety	Malformed input and SQL-like strings.	Treat input as data; do not alter query structure.
Concurrency	Two requests select the same cab together.	Prevent inconsistent overlapping active bookings.

VIII. DISCUSSION AND FUTURE WORK

The case illustrates a feasible baseline for digitalizing a small taxi operation: replace fragmented booking notes with a structured request, associate the request with a customer and vehicle, and provide separate interfaces for passenger, driver and administrator tasks. Its main value is operational organization and inspectable records, rather than automated optimization. The literature on app-based mobility indicates that convenience and broader transport outcomes are separate questions [2]–[4].

The first improvement cycle should prioritize security and data integrity: convert dynamic queries to prepared statements, migrate password storage, add database constraints, validate server-side permissions, and correct reference-code generation. Functional tests should cover registration, sign-in, availability filtering, booking creation, cancellation, driver status transitions, account updates and invalid submissions. Tests should include simultaneous booking attempts for the same cab and attempts by one user to access another user's booking. These are recommended tests, not completed experimental results.

After hardening, a small usability evaluation could ask representative passengers and drivers to complete predefined tasks. Measures may include task-completion rate, time on task, input errors and a standardized usability questionnaire. A separate load test could record response time under specified request levels. A field pilot would additionally require participant consent, safeguards for personal data and clear procedures for transport safety and service disputes. The current study reports none of these measurements; they are proposed directions for future work.

A staged roadmap is preferable to treating all feature requests as equally urgent. Phase one should correct security, reference generation and data-integrity problems. Phase two should add automated functional tests and verify role permissions and concurrent bookings. Phase three could evaluate usability and system responsiveness with volunteers using synthetic or consented data. Only after these steps should geolocation, notifications, fare handling or payment integration be considered. This order places reliability and privacy before feature expansion.

Local adoption also requires process design beyond code. An operator should define who confirms a request, who updates trip status, how cab details are kept current, how cancellations are handled and what message a passenger receives if a request cannot be completed. Pickup and drop-off fields currently accept text, which can be ambiguous when users enter informal landmarks or similar place names.

Clear instructions, server-side validation and explicit confirmation screens can reduce such ambiguity, while driver-facing status updates should provide feedback if a request fails. None of these operational conditions was evaluated in the present case study.

The evaluation protocol should distinguish software behavior from transportation outcomes. Functional test completion and response time can indicate whether the application behaves under stated conditions; only a controlled user study or field pilot can examine waiting times, trip completion, trust or operating cost. An initial pilot could define a completed booking in advance, use consented and minimized data, and document factors such as time of day and vehicle availability. Identifiable contact details and route histories should not be included in public results. This approach would prevent a technically successful prototype from being treated as proof of a broader mobility benefit.

For a reproducible evaluation, each test should record the setup, input values, expected result, actual result and any observed defect. Synthetic accounts and cab records can be used initially so that the test does not expose real passenger information. The same cab identifiers, booking statuses and user roles should be reused across repeated trials to make outcomes comparable. If a test fails, the failure should be categorized as an interface, server validation, authorization, database consistency or deployment issue rather than reported only as a generic error message.

Useful system-level measures include the proportion of test cases completed correctly, time taken to finish a common booking task, number of invalid submissions accepted, and response time at a stated request load. The evaluation should report the test environment and sample size, distinguish observations from expectations, and avoid generalizing results beyond the tested prototype. For user-based evaluation, participation should be voluntary and participants

should be told what data is recorded. Personal contact details, pickup locations and booking histories should not be retained longer than necessary for the study.

IX. CONCLUSION

This paper presented a code-level case study of a PHP/MySQL cab-booking prototype relevant to local taxi digitalization. The reviewed application organizes passenger, driver and administrator workflows and stores customer, cab, category, booking and system-setting data in six MySQL tables. It provides a starting point for structured booking records, but the available evidence does not establish improved waiting times, reliability, safety or commercial performance. The source review identifies concrete priorities before deployment, including parameterized SQL, modern password hashing, database-enforced relationships and proper functional testing. Future work should combine these improvements with user-based evaluation and, where appropriate, a carefully governed field pilot.

REFERENCES

- [1] L. Rayle, D. Dai, N. Chan, R. Cervero, and S. Shaheen, "Just a better taxi? A survey-based comparison of taxis, transit, and ridesourcing services in San Francisco," *Transport Policy*, vol. 45, pp. 168–178, Jan. 2016, doi: 10.1016/j.tranpol.2015.10.004.
- [2] A. Tirachini, "Ride-hailing, travel behaviour and sustainable mobility: An international review," *Transportation*, vol. 47, no. 4, pp. 2011–2047, Aug. 2020, doi: 10.1007/s11116-019-10070-2.
- [3] International Transport Forum, *The Shared-Use City: Managing the Curb*. Paris, France: OECD Publishing, 2018, doi: 10.1787/3af82772-en.
- [4] PHP Documentation Group, "mysqli::prepare," *PHP Manual*. [Online]. Available: <https://www.php.net/manual/en/mysqli.prepare.php>. Accessed: Oct. 13, 2026.
- [5] OWASP Foundation, "SQL Injection Prevention Cheat Sheet." [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. Accessed: Oct. 13, 2026.
- [6] OWASP Foundation, "Password Storage Cheat Sheet." [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. Accessed: Oct. 13, 2026.