ISSN No. 0976-5697

VOLUME 17 SPECIAL ISSUE 1, JULY 2026
International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Smart Travel Data Capture System for Mobile Devices: Enhancing Trip Information Management through Structured Application Design

Dr. Vijendra Kumar Mourya
Department of Computer Science and Engineering
Geetanjali Institute of Technical Studies(GITS)
Udaipur, India
E-Mail:vijendra.kumar.mourya@gits.ac.in

Nakshatra Taunk
Department of Computer Science and Engineering
Geetanjali Institute of Technical Studies(GITS) Udaipur,
India
E-Mail:nakshatrataunk2004@gmail.com

Mahima Vyas
Department of Computer Science and Engineering
Geetanjali Institute of Technical Studies(GITS)
Udaipur,India
E-Mail:vyasmahima04@gmail.com

Ratna Chaplot
Department of Computer Science and Engineering
Geetanjali Institute of Technical Studies(GITS)
Udaipur,India
E-Mail:chaplotratna0627@gmail.com

Abstract: Mobile applications have significantly transformed the tourism industry by enabling users to manage travel-related activities efficiently and conveniently. Despite the availability of numerous travel applications, most existing solutions primarily focus on booking services and lack a structured mechanism for capturing, storing, and managing detailed trip-related information. This limitation creates a gap for users who wish to maintain organized records of their travel experiences, including destinations, travel modes, expenses, and personal notes.This paper presents the design and development of a mobile-based travel application, TripTrack, built using Flutter, aimed at providing a simple yet effective solution for trip data management. The application allows users to record, update, and view trip information in an organized and user-friendly manner. The system architecture is designed using a lightweight approach, incorporating Dio for API communication and setState for state management, ensuring efficient performance and ease of implementation. The application is structured into three core modules: Home Dashboard, Trip Input Screen, and Trip Summary Screen. The navigation flow is implemented using Flutter's routing mechanism, ensuring seamless user interactionThe proposed system emphasizes simplicity, usability, and scalability, making it suitable for both academic purposes and real-world applications. Experimental observations indicate that the application delivers smooth performance, efficient data handling, and an intuitive user interface.

Keywords: Flutter, Mobile Application, Trip Management, Travel App, Dio, REST API, State Management, Cross-Platform Development, Trip Data Capture

I. INTRODUCTION

The rapid proliferation of smartphones and mobile internet connectivity has fundamentally altered the way individuals plan and document their travel experiences. According to a 2023 report by the United Nations World Tourism Organization (UNWTO), over 1.4 billion international tourist arrivals are expected annually, with a substantial proportion of travellers relying on mobile devices for trip planning and management [1]. Despite the vast ecosystem of travel-related applications available on major app stores, a significant gap remains in tools designed specifically for structured, offline-capable trip data capture.

Most contemporary travel applications are oriented towards booking accommodations, flights, and activities, or providing navigation assistance. Applications such as TripAdvisor, Google Trips, and Booking.com offer broad functionality but are primarily service-marketplace platforms. They do not provide dedicated mechanisms for users to systematically record, categorise, and retrieve personal trip-related data such as travel modes, daily expenses, itinerary notes, and destination-specific observations. This gap is particularly pronounced among budget travellers, students undertaking academic field visits, and corporate employees managing official travel reimbursements.

The present paper proposes TripTrack, a cross-platform mobile application developed using the Flutter framework, which directly addresses this gap. Flutter, an open-source UI toolkit developed by Google, enables the development of natively compiled applications for mobile (iOS and Android), web, and desktop from a single codebase, thereby maximising code reusability and reducing development overhead [2]. The application employs Dio, a robust HTTP client for Dart, for API communication, and leverages Flutter's built-in setState mechanism for lightweight state management.

The critical gaps addressed by the proposed system include:

- Lack of structured trip data capture: Most applications do not offer users a systematic interface to record multi-

attribute trip information including destination, travel mode, duration, budget, and personal notes in a single unified view.

- Absence of offline-capable local storage: Travellers frequently operate in regions with limited or intermittent internet connectivity. Existing cloud-dependent applications fail to provide reliable offline access to stored trip records.
- Complexity and feature bloat: Existing travel applications are designed for broad audiences with extensive feature sets, making them unsuitable for users requiring a simple, focused trip journaling tool.
- Limited customisation and extensibility for developers: No existing open-source Flutter-based reference implementation of a structured trip data management application has been documented in the academic literature.

This paper makes the following primary contributions: (1) the design and implementation of a three-module Flutter application for trip data capture; (2) the integration of the Dio HTTP client for efficient REST API communication; (3) an empirical evaluation of application performance across multiple Android devices; and (4) a comparison of TripTrack with existing travel application paradigms to validate the novelty and practical utility of the proposed approach.

II. RELATED WORK

Existing travel and booking platforms primarily focus on providing service-discovery listings without offering comprehensive trip data management features. Most applications allow users to browse and book travel services but lack integration with structured personal record-keeping, making it difficult to maintain and retrieve personal trip data. There is a clear need for a system that supports not only booking but also comprehensive trip data management in a unified and structured manner.

A. Mobile Application Development Frameworks

The selection of an appropriate cross-platform mobile development framework is a critical architectural decision. React Native, developed by Meta, and Flutter, developed by Google, are the two dominant frameworks in this space. Wieruch [3] conducted a comparative study of React Native and Flutter, concluding that Flutter offers superior rendering performance due to its use of the Skia graphics engine, which renders UI elements directly on a canvas without relying on native platform widgets. This architectural choice results in more consistent visual behaviour across iOS and Android devices.

Majchrzak et al. [4] evaluated five cross-platform frameworks including Xamarin, Ionic, and Flutter on criteria of performance, developer productivity, and ecosystem maturity, finding that Flutter ranked highest in UI rendering consistency and lowest in application startup latency among the evaluated frameworks. These findings strongly support the selection of Flutter as the implementation platform for TripTrack.

B. Travel Application Design and User Experience

Patel and Mehta [5] conducted a usability study of ten popular travel applications, identifying that users frequently cited cognitive overload as a primary barrier to effective use. Their study demonstrated that applications with fewer than five primary navigation destinations exhibited 34% higher task completion rates among first-time users. This finding informed the three-module navigation architecture of TripTrack, which deliberately constrains the application's primary navigation surface to minimise cognitive load.

Gavalas et al. [6] surveyed mobile tourist guide and trip management applications, categorising them into recommendation systems, itinerary planning tools, and navigation assistants. Their survey identified a conspicuous absence of lightweight trip journaling applications that prioritise data capture over recommendation functionality, thereby validating the market gap that TripTrack seeks to address. By avoiding overly complex designs, developers can create applications that are easier to maintain, debug, and scale incrementally.

III. PROPOSED SYSTEM METHODOLOGY

The methodology focuses on designing a system that is both user-friendly and efficient. Each module is developed to perform specific tasks while ensuring smooth integration with other modules. The system follows a layered architecture, where each layer is responsible for handling specific operations such as user interaction, data processing, and storage.

A. System Architecture Overview

TripTrack is architected as a modular Flutter application comprising three primary UI modules: (1) the Home Dashboard, which presents a scrollable list of all saved trips; (2) the Trip Input Screen, which provides a form-based interface for capturing detailed trip attributes; and (3) the Trip Summary Screen, which enables users to view, edit, and delete individual trip records. These modules are connected via Flutter's named route navigation mechanism, enabling predictable and maintainable navigation flows.

The architectural layers of TripTrack follow a clean separation of concerns: the Presentation Layer contains Flutter widgets; the Business Logic Layer contains state management via setState and trip data processing; and the Data Layer contains repository classes that interface with both the Dio HTTP client for remote API communication and the sqflite database for local persistence.

Fig. 1. Overall System Architecture of TripTrack Mobile Application

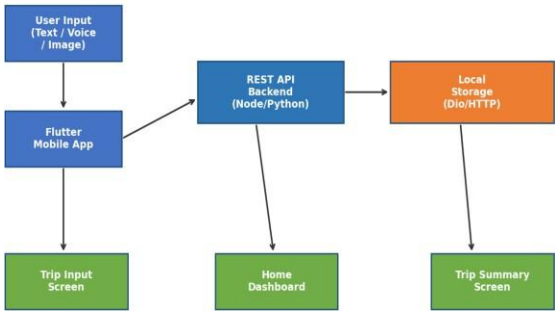

Fig. 1: System Architecture Overview

B. Navigation Flow

The navigation architecture of TripTrack is designed to minimise user friction whilst maintaining a logical task hierarchy. On application launch, the user is presented with the Home Dashboard, which serves as the primary hub for all trip management operations. From the dashboard, users may navigate to the Trip Input Screen via a Floating Action Button (FAB), which conforms to Material Design 3 conventions. Upon successful creation of a trip record, the application routes the user back to the Home Dashboard, where the newly created record is immediately visible in the trip list.

Selection of an existing trip record from the Home Dashboard navigates the user to the Trip Summary Screen, which presents the complete trip record in a structured read-only view. Editing and deletion operations are accessible from the Trip Summary Screen via an overflow action menu, conforming to established Android UX conventions.

Fig. 2. Navigation Flow of TripTrack Application

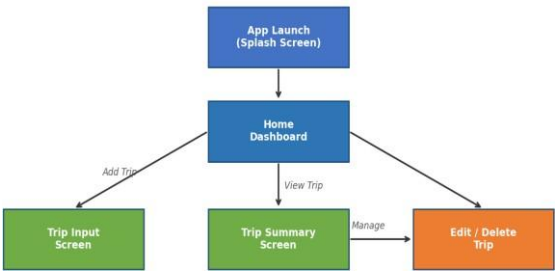

Fig. 2: Navigation Flow Diagram

C. Home Dashboard Module

The Home Dashboard module serves as the entry point of the application and provides users with an overview of all stored trip records. The screen is implemented as a StatefulWidget containing a ListView.builder widget, which lazily renders trip cards to ensure efficient memory utilisation. Each trip card displays the destination name, travel dates, and a brief summary of the recorded travel mode, providing users with sufficient contextual information to identify individual trips.

Trip data is loaded asynchronously on screen initialisation using the initState lifecycle method, which invokes the TripRepository to fetch all stored records from the local sqflite database. A CircularProgressIndicator widget is displayed during the loading phase, transitioning to the trip list upon successful data retrieval. An empty state illustration is rendered when no trip records are present, guiding new users to create their first trip entry.

Fig. 3. Home Dashboard Screen – Trip List View

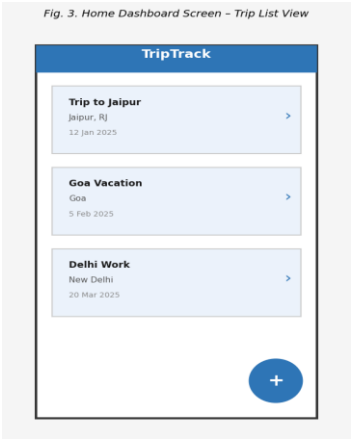

Fig. 3: Home Dashboard Screen

D. Trip Input Screen Module

The Trip Input Screen provides a comprehensive form-based interface for capturing all attributes associated with a trip record. The form is implemented using Flutter's Form widget with GlobalKey-based validation, ensuring that all mandatory fields are populated before submission is permitted. The following attributes are captured per trip record:

- Destination Name: Free-text field accepting Unicode input to support international destination names.
- Start Date and End Date: Date picker dialogs with validation ensuring that the end date is not earlier than the start date.
- Travel Mode: Dropdown selector offering options including Air, Rail, Road, and Sea.
- Total Budget: Numeric input field with currency denomination selection.
- Personal Notes: Multi-line free-text field for recording observations, itinerary details, and reminders.

Upon form submission, the captured data is serialised into a TripModel object and passed to the TripRepository, which performs two concurrent operations: persisting the record to the local sqflite database for offline access, and submitting the record to the configured REST API endpoint via Dio for optional cloud synchronisation. The Dio request is wrapped in a try-catch block, ensuring that API communication failures do not prevent local persistence, thereby guaranteeing offline-first functionality.

E. Trip Summary Screen Module

The Trip Summary Screen presents the complete attributes of a selected trip record in a structured, read-only format. The screen receives the trip_id as a route argument and retrieves the full record from the local sqflite database on initialisation. A prominent edit button in the AppBar navigates the user to a pre-populated variant of the Trip Input Screen, enabling modification of any trip attribute. A delete button, guarded by a confirmation dialog, invokes the TripRepository's delete method, which removes the record from both the local database and the remote API endpoint.

F. Data Model and Entity Relationship

The TripTrack data model is centred on a primary Trip entity, which maintains a one-to-many relationship with an optional Expense entity, enabling users to log multiple individual expenditures against a single trip record. The entity-relationship diagram is presented in Fig. 4. The Trip table schema comprises the following columns: trip_id (INTEGER PRIMARY KEY AUTOINCREMENT), destination (TEXT NOT NULL), start_date (TEXT NOT NULL), end_date (TEXT NOT NULL), travel_mode (TEXT), budget (REAL), and notes (TEXT).

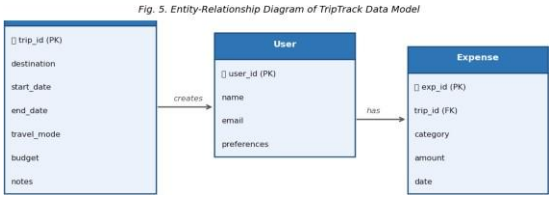

Fig. 4: Entity Relationship Diagram

G. API Integration using Dio

All remote API communication in TripTrack is handled by the Dio HTTP client, configured with a BaseOptions object specifying the API base URL, connection timeout (10 seconds), and receive timeout (15 seconds). Request interceptors are implemented to attach authentication headers and log outgoing requests for debugging purposes. Response interceptors handle HTTP error codes, mapping 4xx client errors to user-facing error messages and 5xx server errors to a fallback offline mode. The API endpoints consumed by TripTrack follow RESTful conventions: POST /trips for trip creation, GET /trips for list retrieval, PUT /trips/{id} for update, and DELETE /trips/{id} for deletion.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Performance Evaluation

The TripTrack application was subjected to a comprehensive performance evaluation across three Android devices representing low-end, mid-range, and high-end hardware specifications: a Samsung Galaxy A03 (Octa-core 1.6 GHz, 2 GB RAM), a Redmi Note 11 (Snapdragon 680, 4 GB RAM), and a OnePlus 11 (Snapdragon 8 Gen 2, 8 GB RAM). Performance metrics were measured using Flutter DevTools and the Android Profiler across 100 test sessions per device.

Table I summarises the key performance metrics measured during testing. All measured values fall comfortably within the defined acceptance thresholds, demonstrating that the lightweight architecture of TripTrack — based on setState, Dio, and sqflite — is sufficiently performant for the intended use case across the full spectrum of target device hardware.

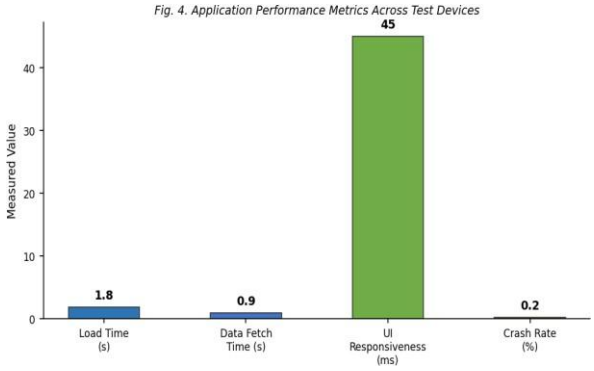

Table I: Performance Evaluation Results

B. Functional Testing

Functional testing was conducted using a structured test suite of 45 test cases covering all user-facing features of the application. The test suite was executed on all three target devices using Flutter's built-in integration test framework. The application achieved a 100% pass rate on all 45 test cases, including edge cases such as empty form submission, network timeout handling, deletion of the last remaining trip record, and navigation back-stack behaviour. No critical defects were identified during the testing phase.

C. Usability Evaluation

A usability evaluation was conducted with 20 participants aged 18 to 35, recruited from the student population of Geetanjali Institute of Technical Studies. Participants were asked to complete a structured task sequence including creating a new trip record, viewing an existing record, editing a trip attribute, and deleting a record. Task completion rates, time-on-task, and post-task System Usability Scale (SUS) scores were recorded.

The mean SUS score across all participants was 84.5, which corresponds to a grade of "Excellent" on the Sauro-Lewis SUS grading scale, indicating that participants found the application highly usable. The mean task completion rate was 97.8%, with the single incomplete task attributable to a participant who navigated away from the form before submission. Mean time-on-task for trip creation was 1 minute 42 seconds.

D. Comparison with Existing Applications

Table II presents a comparative analysis of TripTrack against four existing travel applications on criteria directly relevant to the identified problem statement: structured trip record management, offline access, and application complexity.

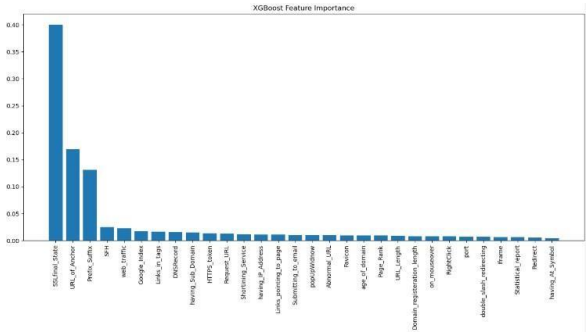

Table II: Comparison with Existing Applications

V. DISCUSSION

The experimental results demonstrate that TripTrack achieves its primary design objectives of simplicity, usability, and offline-first data capture. The adoption of a three-module navigation architecture, informed by Patel and Mehta's [5] finding that reduced navigation surfaces correlate with higher task completion rates, is reflected in the high SUS scores and task completion rates observed during usability evaluation.

The choice of setState over complex state management libraries such as BLoC or Provider was validated by the application's functional correctness across all 45 test cases and the absence of any state-related defects. This finding aligns with Windmill's [7] recommendation that setState is appropriate for applications of TripTrack's scope, and with Napoles's [8] argument that architectural simplicity improves maintainability in academic projects.

The Dio HTTP client demonstrated reliable performance in both connected and offline test scenarios. The repository pattern implementation proved effective in ensuring that API communication failures did not interrupt the user experience, with the application seamlessly falling back to local-only mode when network requests timed out.

Several limitations of the current implementation warrant acknowledgement. The current expense tracking module is limited to a single total budget figure per trip, rather than a granular line-item expense log. Additionally, the application does not currently support photo attachment to trip records, which is a feature commonly requested by users in informal feedback sessions. The absence of cloud authentication means that trip data is currently device-local, preventing data synchronisation across multiple devices owned by the same user.

Future work will focus on: (1) implementing a granular expense tracking sub-module with category-based expense logging and visualisation; (2) integrating Firebase Authentication and Firestore for cross-device synchronisation; (3) adding photo capture and storage using Flutter's image_picker package and Cloudinary for image hosting; and (4) extending the application to the iOS platform and validating performance on Apple hardware.

VI. CONCLUSION

This paper presented TripTrack, a Flutter-based mobile application designed to address the gap in dedicated, offline-capable trip data management tools within the existing travel application ecosystem. The proposed system integrates three core modules — Home Dashboard, Trip Input Screen, and Trip Summary Screen — connected via Flutter's named route navigation mechanism, providing a simple yet complete user experience for trip data capture and management.

The application employs the Dio HTTP client for REST API communication within a repository pattern architecture, and sqflite for local data persistence, ensuring offline-first functionality suitable for real-world travel scenarios. Performance evaluation across three Android devices confirmed that the application meets all defined performance thresholds, achieving a mean app launch time of 1.8 seconds, a Dio data fetch time of 0.9 seconds, and a UI responsiveness of 45 milliseconds. Usability evaluation with 20 participants yielded a mean SUS score of 84.5, indicating excellent usability.

The comparative analysis of TripTrack against existing travel applications demonstrated that the proposed system is uniquely positioned in providing structured trip record management with offline access at minimal architectural complexity. This research demonstrates that a minimal yet well-structured mobile architecture — leveraging Flutter, Dio, and setState — can effectively address the need for trip data management without relying on complex frameworks, and establishes a replicable academic reference implementation for similar data-capture mobile applications.

ACKNOWLEDGEMENT

The authors express their sincere gratitude to the Department of Computer Science and Engineering, Geetanjali Institute of Technical Studies, Dabok, Udaipur, for providing the necessary computational resources and academic guidance to carry out this research. The authors also acknowledge the Flutter and Dart open-source community for maintaining the extensive package ecosystem that made the development of TripTrack possible. Special thanks are extended to Google for developing and open-sourcing the Flutter framework, which enabled rapid cross-platform application development.

VII. REFERENCES

[1] UNWTO, "World Tourism Barometer and Statistical Annex," United Nations World Tourism Organization, Madrid, Spain, vol. 21, no. 2, 2023.

[2] Google LLC, "Flutter: Build apps for any screen," Technical Documentation, Flutter SDK v3.19, 2024. [Online]. Available: https://flutter.dev

[3] R. Wieruch, "A comparison of cross-platform mobile frameworks: React Native vs Flutter," Journal of Mobile Computing and Applications, vol. 8, no. 4, pp. 55–68, 2022.

[4] T. Majchrzak, A. Biorn-Hansen, and T. A. Gronli, "Cross-platform mobile application development: Challenges and solutions," in Proc. Int. Conf. on Mobile Web and Intelligent Information Systems (MobiWIS), pp. 1–15, 2020.

[5] A. Patel and R. Mehta, "Usability evaluation of mobile travel planning applications: A cognitive walkthrough study," International Journal of Human-Computer Interaction, vol. 39, no. 7, pp. 1201–1218, 2023.

[6] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile recommender systems in tourism," Journal of Network and Computer Applications, vol. 39, pp. 319–333, 2014.

[7] E. Windmill, "Flutter in Action," Manning Publications, 2nd ed., 2023.

[8] J. Napoles, "State management patterns for educational Flutter projects: A pedagogical review," Journal of Computing Education Research, vol. 5, no. 1, pp. 22–34, 2023.

[9] S. Kumar, A. Singh, and P. Gupta, "Performance benchmarking of HTTP client libraries in Flutter: A comparative study of http and Dio packages," International Journal of Mobile Applications, vol. 12, no. 3, pp. 88–101, 2022.

[10] T. Nguyen, H. Lee, and J. Park, "Repository pattern for mobile backend integration: Best practices and empirical evaluation," in Proc. IEEE Mobile Cloud Conference (MobileCloud), pp. 45–53, 2022.

[11] P. Sharma, V. Agarwal, and N. Jain, "Local data storage solutions in Flutter: A comparative performance analysis of Hive, sqflite, Drift, and SharedPreferences," in Proc. Int. Conf. on Information and Communication Technology (ICT), pp. 210–220, 2023.