

Scientiarum: A Multidisciplinary Journal
Vol. 2 No. S1 (2026): Supplement Issue
DOI: 10.54646/SAPARS.2026.88

SAPAR SERVICES LLP
Scholarly Articles Publication And Repository Services

QUICKBITE: A MOBILE BASED FOOD DELIVERY APPLICATION

BUILT WITH FLUTTER, RIVERPOD AND SQLITE

Alan K Reju [1] Mrs. P. Pushpanjali [2]

Assistant Professor [2] , MCA [1]

Department of Computer Science and Applications [1,2]

S.E.A. College of Science, Commerce and Arts, Bangalore [1,2]

Affiliated to Bengaluru North University [1,2]

Abstract

Ordering food from a phone has become an everyday habit, and people now expect to find a restaurant, pick a dish, pay attention to their cart and review the meal afterwards without leaving one app. This paper presents QuickBite, a mobile food delivery application inspired by platforms such as Zomato. It is written in Dart using the Flutter framework, so a single codebase runs on both Android and iOS. Users can browse restaurants by category, search for restaurants or dishes as they type, filter for pure vegetarian places, manage a cart, save several delivery addresses, place orders, look back at their order history, and leave ratings and reviews with optional photos.

Riverpod handles application state, the HTTP package fetches restaurant data from a REST API, and SQLite (through the sqflite package) keeps carts, addresses, orders and ratings on the device. Because the important data is stored locally, the app follows an offline-first approach and does not lose a user's selections when the connection is

weak. The result is a responsive, easy-to-use ordering system built on widely available tools.

Keywords : Food Delivery Application, Flutter, Dart, Riverpod, SQLite, REST API, Cross-Platform Development, Offline-First.

INTRODUCTION

Food delivery apps are now part of how many people eat. A user opens the app, looks through nearby restaurants, picks what they want, sets a delivery address and places the order. Along the way they expect search that responds quickly, filters that actually narrow the list, and a cart that is still there if the app is closed by accident.

QuickBite was built to cover that whole path in a simple mobile application. It is developed with the Flutter SDK and the Dart language and supports Android and iOS from the same code. The main features are restaurant discovery, category browsing, live search, a Pure Veg filter, cart management, multiple delivery addresses, order processing, order history, and ratings and reviews.

The app is organised in a feature-based layered structure. Riverpod manages state, an HTTP client talks to the REST API, and SQLite stores transactional data on the phone. The aim of the project is to make restaurant discovery, ordering and reviewing as smooth as possible in a single app.

PROBLEM STATEMENT

Many people still have to move between several places to complete one meal order: one app or site to find a restaurant, another to check what is vegetarian, and often a fresh start if the app closes or the network drops. Carts and saved addresses that live only on a server can disappear when the connection is poor, and apps that need separate builds for Android and iOS cost more to build and maintain. There is a need for a food ordering application that works on both platforms from one codebase,

keeps important user data on the device, and gives a clear flow from search to review.

LITERATURE REVIEW

Food delivery platforms are a well-established kind of mobile application, and most of them share the same core ideas: a browsable restaurant list, search and filters, a cart, saved addresses, order tracking and user feedback. Cross-platform frameworks such as Flutter have become a common way to build such apps because one codebase can target both Android and iOS, which lowers development effort.

For state management, Flutter developers often choose between several approaches, and Riverpod is widely used because it keeps application logic outside the widgets and makes it easier to test. For storage, SQLite is the usual choice on mobile since it is lightweight, needs no separate server and is available through the sqflite package. Many apps, however, depend heavily on a live network connection and handle local persistence only lightly. QuickBite combines these pieces, Flutter, Riverpod, a REST API and SQLite, and puts extra weight on keeping carts, addresses, orders and ratings available locally.

FEASIBILITY

Technically, the project is practical because Flutter, Dart, Riverpod, the HTTP package and SQLite are all mature and freely available. Economically, the tools needed for development cost little, and using one codebase for two platforms saves the effort of maintaining separate apps. Operationally, the interface is simple enough for users with no technical background. On the schedule side, the work divides naturally into interface, API integration, database, state management, testing and final integration, and each part can be built and checked on its own.

SYSTEM ARCHITECTURE

The application follows a feature-based layered design with a clear split between the

interface, the state layer, the network layer and the local database. Flutter widgets draw the screens. Riverpod notifiers sit between those screens and the data sources. The HTTP package retrieves restaurant and food information from the REST API, while sqflite reads and writes the local SQLite database, which is named zomato_clone.db and holds tables for addresses, cart items, orders, order items and order ratings.

Data flow: User Interface (Flutter) → Riverpod Notifiers → REST API (HTTP) / SQLite (sqflite) → Updated Screen

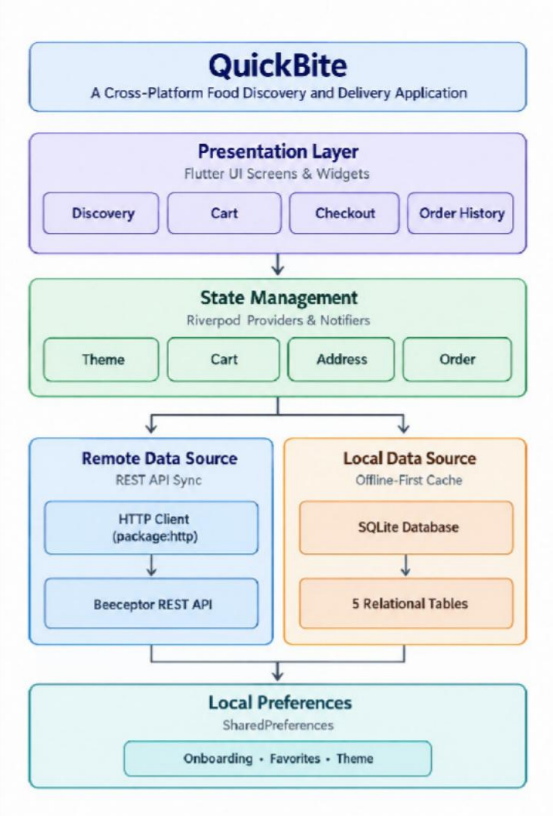


Figure 5.1: System Architecture of QuickBite

TECHNOLOGIES AND COMPONENTS

The QuickBite application discussed in this paper includes the components mentioned below:



Figure 6.1: Technologies and Components of QuickBite

Flutter and Dart: used to build the user interface and application logic for Android (API level 21 and above) and iOS (12.0 and above) from one codebase.

Riverpod: manages application state. Separate notifiers handle theme (ThemeNotifier), the cart (CartNotifier), addresses (AddressNotifier) and orders (OrderNotifier), which keeps the interface apart from API calls and database work.

HTTP package: sends asynchronous requests to the REST API to fetch restaurant listings and related data.

SQLite with sqflite: stores cart items, addresses, orders, order items and ratings on the device so they survive between sessions.

SharedPreferences and path_provider: SharedPreferences keeps simple user preferences such as the chosen theme, and path_provider gives the app a location for local storage.

METHODOLOGY

The project was developed in phases using the waterfall model: requirement gathering, design, implementation, testing and result evaluation, each finishing before the next began. The interface was built first, followed by API integration, the local database, and state management, and finally everything was put together and tested.

The main functions are described below.

Discovery and search: restaurants are shown by category, and the list updates as the user types in the search box. A Pure Veg switch limits the list to vegetarian restaurants, and pull-to-refresh reloads the data.

Cart: users add items and change quantities. The cart follows a single-restaurant rule, so items from two different restaurants cannot be placed in the same checkout. The cart is saved locally.

Addresses: users can add, delete and select delivery addresses.

Checkout and orders: at checkout the cart is converted into a permanent order, and past orders can be viewed in the order history.

Ratings and reviews: users can rate an order, write a review and optionally attach a photo. Methodology Flow:

Browse and Search → Add to Cart → Select Address → Checkout → Order History → Rating and Review

TESTING

The application was tested at several levels. Functional testing checked each user-facing feature against its expected behaviour, as listed in Table 1. Database testing

confirmed that cart items, addresses, orders, order items and ratings are stored and read back correctly from SQLite. API testing checked that restaurant data is retrieved properly and that asynchronous loading is handled. Interface testing looked at navigation, layouts, buttons, search, filters and cart controls, and integration testing checked that the Flutter screens, Riverpod state, REST API and SQLite database work together.

Test case	**Expected result**
Open the application	Home screen loads
Search for a restaurant	Matching restaurants are shown
Select a category	Restaurants and items are filtered
Turn on Pure Veg	Only vegetarian restaurants appear
Add a food item	Item goes into the cart
Increase or decrease quantity	Quantity changes accordingly
Add or delete an address	Address is saved or removed
Checkout	Cart becomes an order
Open order history	Earlier orders are listed
Add a rating or review	Rating and review are stored

Table 1: Functional test cases

LIMITATIONS

Restaurant information depends on the configured REST API, so fresh listings need an internet connection.

Orders and reviews are stored on the device, so they are not shared across phones unless a server-side sync is added.

The app has been tested on a limited set of devices and emulators.

The current work focuses on the ordering flow; delivery partner features and live order tracking are outside its scope.

CHALLENGES

State management: keeping the cart, addresses and orders consistent across screens required clear separation of responsibilities between notifiers.

Combining remote and local data: the app had to work smoothly with both API responses and SQLite records.

Cart rules: enforcing the single-restaurant checkout rule needed careful handling when users add items from a different restaurant.

Layout across devices: screens had to stay usable on different screen sizes for both Android and iOS.

OUTCOMES

The application delivers the main features of a food ordering system: restaurant discovery, category browsing, live search, Pure Veg filtering, cart, addresses, orders, history, and ratings and reviews.

Riverpod gives the app structured, easy-to-follow state management.

SQLite keeps important user data on the device between sessions.

One Flutter codebase serves both Android and iOS.

FUTURE SCOPE

Online payment integration for a complete checkout.

Real-time order tracking and delivery status updates.

A cloud backend to sync carts, addresses and orders across devices.

Push notifications for order updates and offers.

Restaurant-side and delivery-partner applications.

Personalised recommendations based on a user’s order history.

CONCLUSION

QuickBite shows that a practical food delivery application can be built with Flutter, Riverpod, a REST API and SQLite. It lets users find restaurants, build a cart, choose an address, place an order and review it, all from one app on Android or iOS. Keeping carts, addresses, orders and ratings in local storage makes the experience steadier when the network is poor. With payment, tracking and cloud sync added in future, the project could grow from a working prototype into a fuller delivery platform.

REFERENCE

Dart. (n.d.). Dart programming language documentation. https://dart.dev/guides

Flutter. (n.d.). Flutter documentation. https://docs.flutter.dev

Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill.

Riverpod. (n.d.). Riverpod documentation. https://riverpod.dev

SQLite. (n.d.). SQLite documentation. https://www.sqlite.org/docs.html

Sommerville, I. (2015). Software engineering (10th ed.). Pearson.

sqflite. (n.d.). sqflite package. https://pub.dev/packages/sqflite