Melbourne Travel App (Flutter): Location Discovery, Itinerary Builder, Saved Places, Tickets & Profile

1. Project Overview and Summary

This project showcases a complete end-to-end **mobile/web travel discovery application** built using **Flutter (Dart)**. The goal is to simulate a real-world tourism and travel experience where users can **explore Melbourne destinations**, view details with google maps and weather forecast, **save favorite destinations**, and generate a **nearby itinerary (Food and Shopping)** around a selected attraction.

The app focuses on making trip planning easier by solving common travel pain points:

- Visitors don't know *what to do nearby* once they pick a landmark.
- People want a **quick plan** without searching multiple applications or websites.
- Users want to **save places** and return later.
- Itineraries should open directly in **Google Maps**.

Core App Capabilities

The app is designed around these core user actions:

- Browse recommended and nearby places
- View tourist spot details (photo, map, weather, distance)
- Build an itinerary of nearby food and shopping options (auto fetched)
- Open places/routes directly in Google Maps
- Save/remove favorites
- View favorites and clear saved data
- Tickets and profile modules included for full app navigation experience

1.1 Problem Statement

Planning a short trip is often fragmented across multiple apps: searching places, estimating distance, checking weather, building a simple route, and saving options for later. The problem is to provide a single, simple workflow that turns 'I want to visit a place' into an actionable plan (nearby food, shopping and map links) with minimal effort.

1.2 Project Goals

- Provide a clean onboarding experience (Welcome screen) and a simple 4-tab navigation shell.
- Show curated Melbourne places with category filtering, recommendation cards, and nearby cards sorted by distance.
- Offer place details with image and map, weather summary, and a one-tap itinerary builder. Generate itinerary suggestions (Food and Shopping) using OpenStreetMap Overpass API and allow opening locations in Google Maps.

- Allow users to save/unsave favourites and review them later.
- Provide lightweight Tickets and Profile pages to complete the app flow.

1.3 Key Outcomes

- Navigation duplication and bottom padding issues were resolved by consolidating navigation into HomeShell only.
- State handling for favourites was unified so Home, Details, and Saved pages stay consistent.
- Itinerary reliability improved by adding refresh and moving Futures into initState (fixes intermittent blank tabs).
- Google Maps opening on Flutter Web was made reliable using webOnlyWindowName: '_blank'.

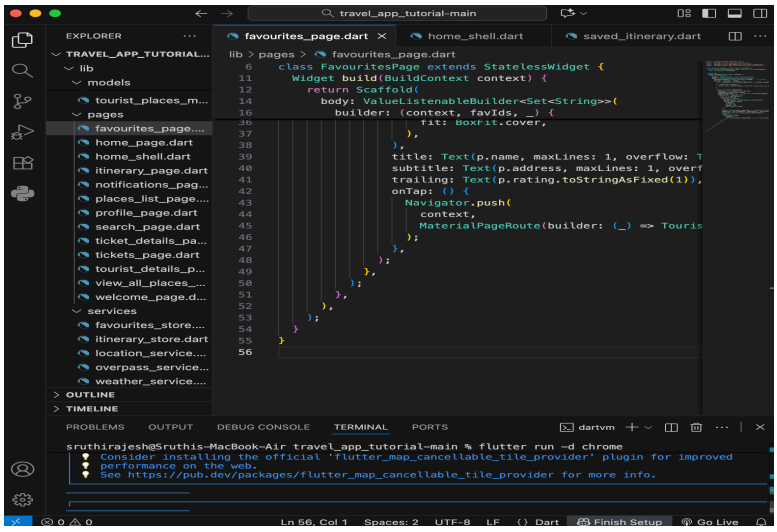
2. System Overview

2.1 Tech Stack

- Frontend: Flutter
- Maps: flutter_map (OpenStreetMap tiles)
- Location: geolocator and geocoding (current position + label)
- Weather: HTTP-based weather service abstraction (WeatherService)
- POI search for itinerary: Overpass API (OpenStreetMap) via OverpassService
- External links: url_launcher (open Google Maps routes/locations)

2.2 High-Level Architecture

The app follows a page-based architecture. HomeShell owns the bottom navigation and switches pages using an IndexedStack. Data (Place list) is loaded once and passed down to child pages (Home, Search, Saved).



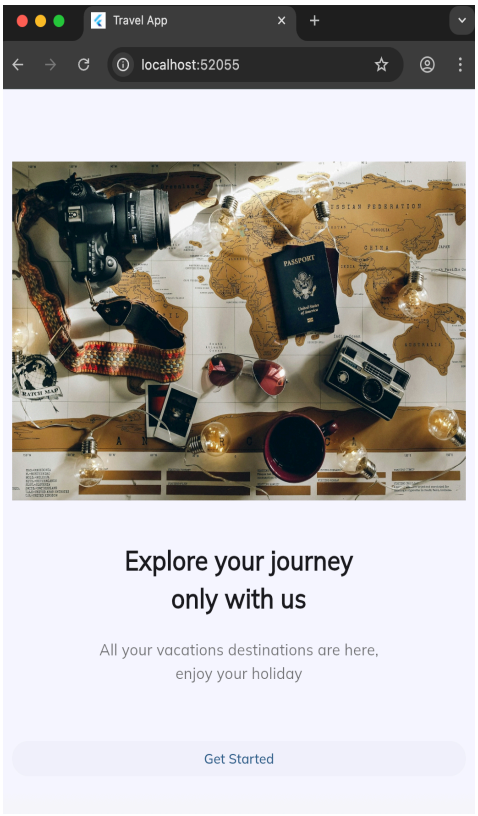
2.3 Data Model

- Place: id, name, category, image, address, lat, lng, rating.
- Poi (Point of Interest): id, name, type, lat, lng, address (from Overpass).
- Ticket: title/subtitle/date label/stops; TicketStop: name, lat, lng.

3. User Journey

1. Welcome → user taps Get Started.
2. Home tab → browse recommended + nearby, filter by category.
3. Tap a place → Tourist Details (image + map + weather + distance).
4. Tap “Build itinerary for this place” → Itinerary tabbed view (Food / Shopping) + map pins + list.
5. Tap a POI → opens Google Maps in a new tab (web) or external app (mobile).
6. Tap heart icon → save/unsave favourites; view in Saved tab.
7. Tickets tab → view saved tickets (itineraries) and open Ticket details.
8. Profile tab → basic profile settings + counts; manage saved items.

3. Solution Approach

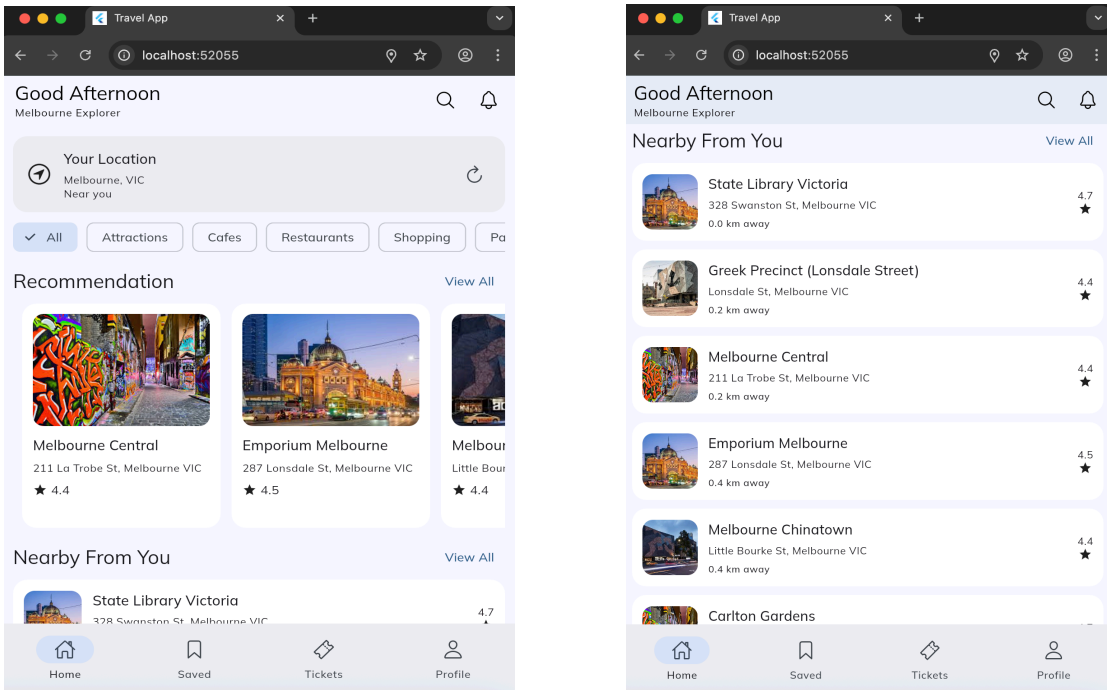


Welcome page (Tap Get started to Begin)

A) Location-based sorting and discovery (Home)

- The Home page loads places and sorts them based on the user's location (or fallback location).
- Users can filter places using category chips (Attractions, Cafes, Restaurants, Shopping, Parks).

Outcome: Users instantly see the most relevant places near them instead of scrolling random lists.



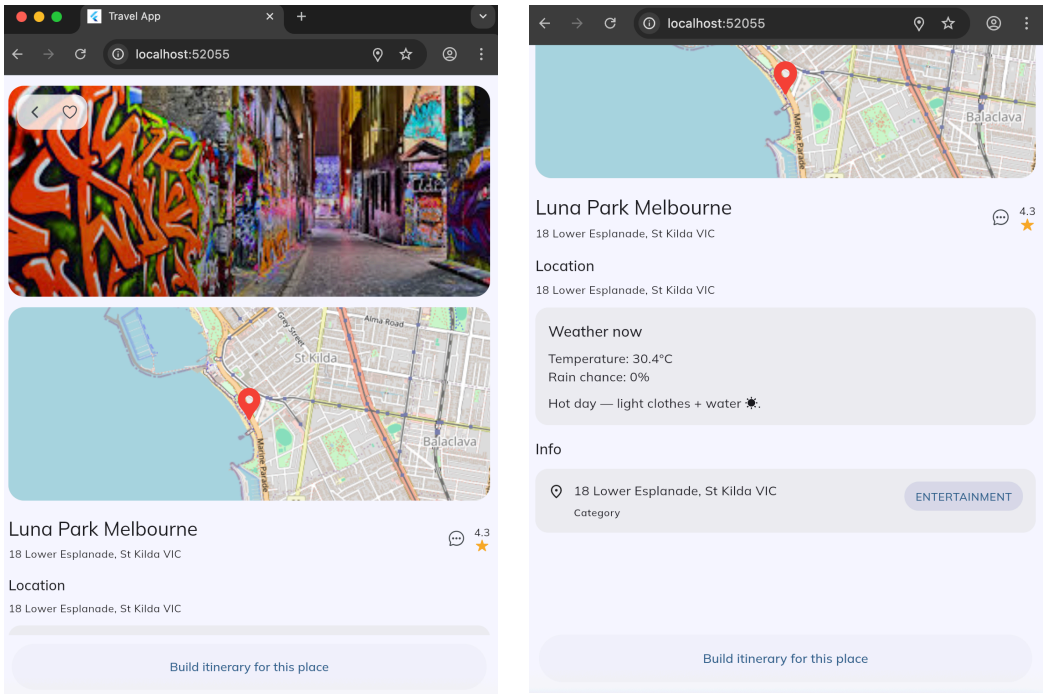
Home Page with categories, recommendations and nearby places

B) Tourist details with map, weather and favourites

When a user taps a place:

- A details page show:
  - Hero image (place photo)
  - Embedded map preview
  - Weather pulled from an API service (temperature with rain chance)
  - Distance component (how far it is)
  - Favourite toggle (save/remove)

Outcome: Users get "decision-making info" (weather, distance and location) instantly



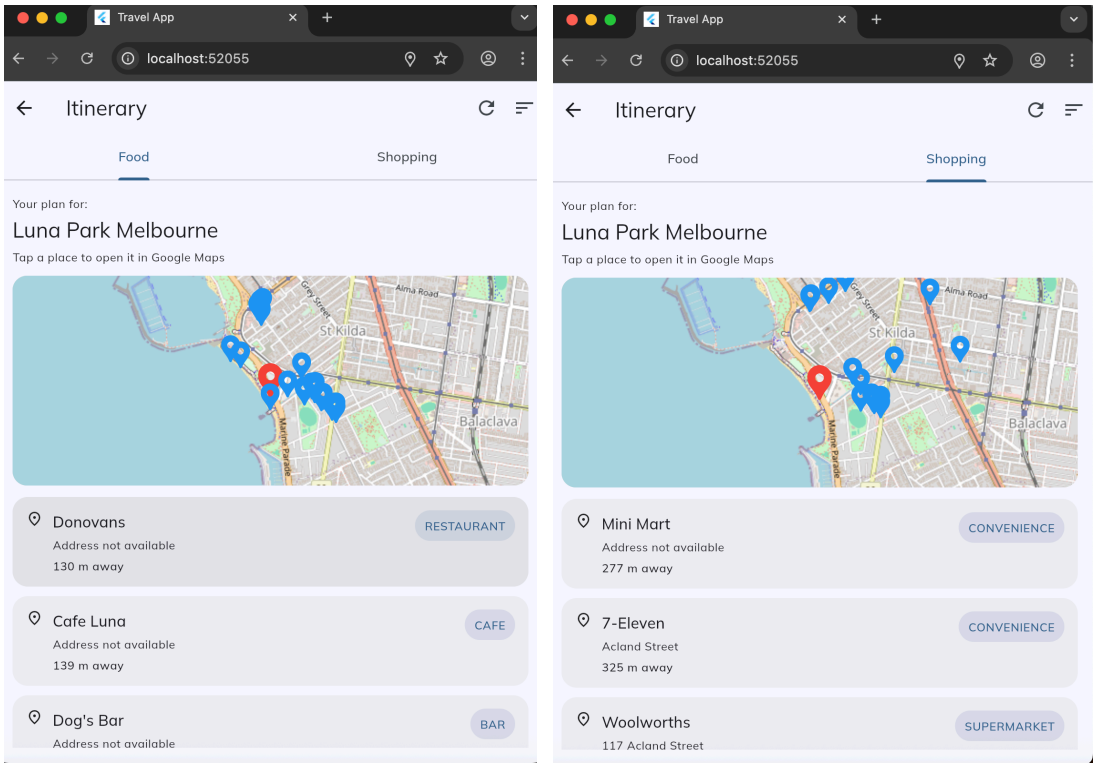
Tourist Details Page

Itinerary Builder (Food + Shopping near the selected place)

This is the main "planner" feature:

- User presses "Build itinerary for this place"
- Itinerary page fetches nearby points of interest (POIs) using Overpass (OpenStreetMap)
- Shows results in two tabs:
  - **Food**
  - **Shopping**
- Includes:
  - Sort mode (Closest / A–Z)
  - Refresh button (retry for Overpass failures)
  - Map pins + list of places
  - Tap any POI to open in Google Maps

Outcome: The app auto-generates a local plan around the selected destination.

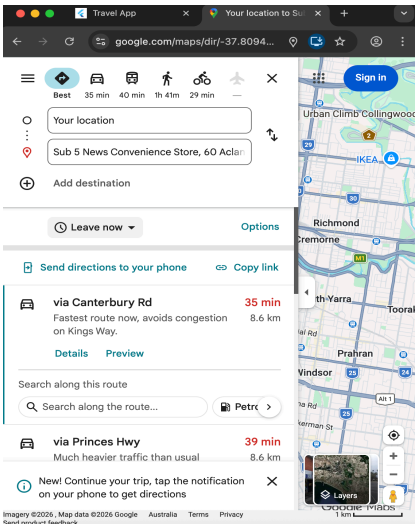


Itinerary Food and Shopping Tabs

D) Google Maps integration (Web + Mobile-safe)

- The app opens Google Maps using safe web support (webOnlyWindowName: '_blank')
- This ensures the link opens reliably in browser and avoids web launch issues.

Outcome: One-click navigation from the app into Google Maps.

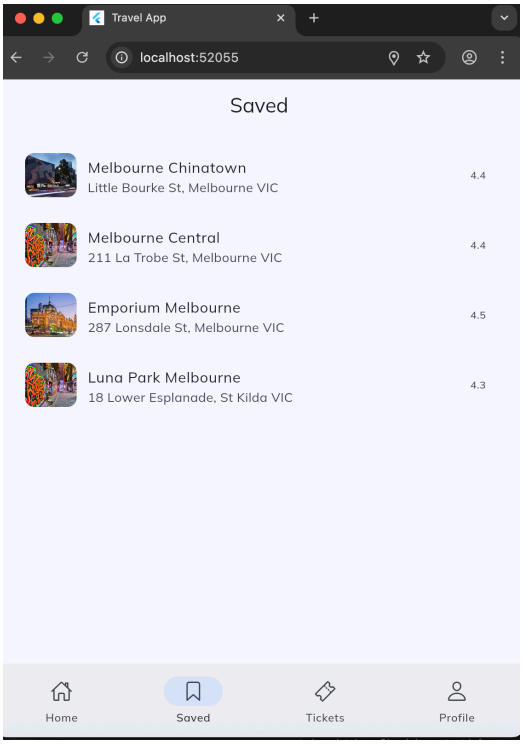


E) Saved Places (Favourites)

Users can save places and view them later:

- Favourite toggle on details page
- Saved page lists all favourited places
- Profile includes saved count + clear saved button

Outcome: Users don't lose their planning progress.



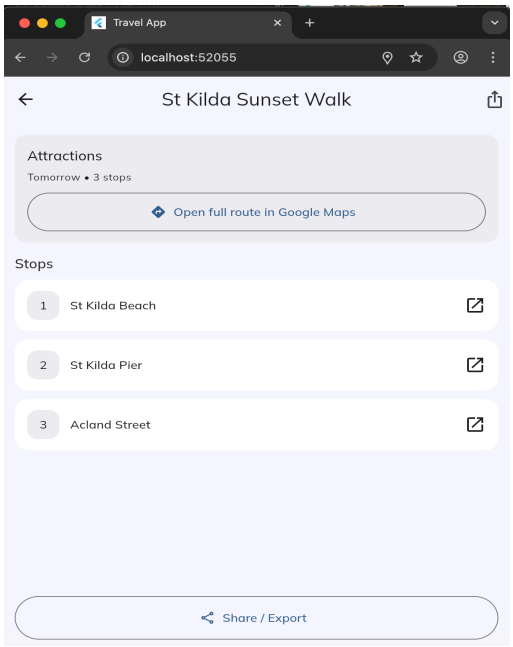
Saved Places

F) Tickets module (Trip plan storage concept)

Tickets acts like a trip plan record:

- Shows a list of "tickets" (itinerary plans)
- Each ticket opens a Ticket Details page
- Ticket details supports:
    - Open full route in Maps
    - Open each stop individually in Maps
    - Export/copy itinerary text (clipboard)

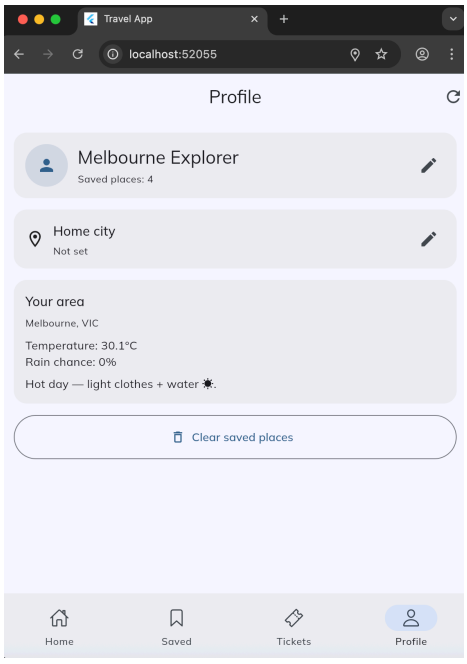
Outcome: Adds a "real travel-app feel" where itineraries can be saved and reused.



G) Profile Page (Non-hard-coded, editable)

- User can edit their display name
- Shows saved places count
- Includes “clear saved places” action (connected to favorites’ store)

Outcome: Adds personalization and demonstrates state and user settings.



4. Pages and Design System Components (DSM)

DSM components are reusable UI building blocks used across multiple pages to ensure consistent design and faster development. Based on the implemented codebase, the project uses approximately 8-10 reusable DSM components/widgets (depending on whether you count small UI patterns like SectionHeader and TypeChip).

4.1 Pages Implemented

Page	Purpose	Key Features
WelcomePage	Onboarding	Illustration, tagline, Get Started button
HomeShell	App shell	Bottom navigation (Home/Saved/Tickets/Profile), IndexedStack
HomePage	Discovery	Location label, category chips, recommended carousel, nearby list
SearchPage	Find places	Search across allPlaces, open details
TouristDetailsPage	Place details	Image + map, weather card, distance, save toggle, itinerary CTA
ItineraryPage	Mini-plan builder	Food/Shopping tabs, map pins, list, refresh, open in Maps
FavouritesPage	Saved items	Saved list + open details
TicketsPage + TicketDetailsPage	Saved itineraries	Ticket cards, open full route, export/copy
ProfilePage	User settings/summary	Editable display name, saved count, (optional) quick actions
NotificationsPage (optional)	Placeholder	Basic notifications UI / future extension

4.2 DSM Components Used

- CustomIconButton: consistent icon-button styling in AppBar actions.
- LocationCard: shows current location label and refresh action.
- RecommendedPlaces: horizontal carousel card layout.
- NearbyPlaces: vertical list card layout with distance calculation.
- PlaceMap: reusable map widget (place-only and place+POI pins).
- Distance: distance display widget in TouristDetailsPage.
- SectionHeader (pattern): title and View All action reused across sections.
- TypeChip (pattern): category/type labels for POI cards.

5. Key Implementation Steps

5.1 Navigation & Page Wiring

- Created HomeShell as the single owner of bottom navigation (prevents duplicate navigation bars).
- Used IndexedStack to preserve page state when switching tabs (e.g., scroll position, loaded data).
- Passed the same allPlaces list into HomePage, SearchPage, and FavouritesPage to keep behaviour consistent.

5.2 Data Loading

- Implemented PlacesRepository to load a JSON dataset from assets (melbourne_places.json).
- Validated pubspec.yaml asset paths to ensure Flutter bundles images and JSON correctly.
- Standardised Place model usage so all pages consume the same structure.

5.3 Favourites State Management

FavouritesStore was used as a lightweight shared store so that saving a place in TouristDetailsPage immediately reflects in the Saved tab and in profile counts. The key requirement is that the store exposes a single source of truth (e.g., a Set<String> of IDs) and provides functions like toggle(id), isFavourite(id), and clear().

5.4 Itinerary Reliability Fixes

- Moved POI Futures into initState and added a _reload() method (prevents using widget in field initializers).
- Added a refresh icon in the AppBar to allow retry when Overpass temporarily fails.
- Improved Google Maps opening on Flutter Web using launchUrl(url, webOnlyWindowName: '_blank').
- Added sort modes (closest and A–Z) for better usability.

6. Usefulness and Benefits

- Faster trip planning: users can go from discovery to an itinerary in a few taps.
- Reduced app switching: details, weather, map preview, and navigation links are all in one flow.
- Reusable UI components enable quick expansion (add more cities, more categories, additional itinerary modes).
- A strong base for future features: authentication, persistent storage, user-generated itineraries, sharing links.

7. Testing and Verification

- Navigation tests: ensured each tab opens correctly and state persists when switching tabs.
- Asset tests: verified images and JSON load on Flutter Web and mobile emulator.
- Itinerary tests: verified Overpass fetch works and refresh re-runs requests.
- External link tests: verified Google Maps links open reliably on Flutter Web (new tab) and mobile (external app).

8. Conclusion

The completed Travel App provides a complete workflow for trip planning: users can explore destinations, view detailed information, save favourites, build a nearby itinerary, and open stops/routes in Google Maps. The codebase is structured for scalability using HomeShell navigation, service-based integration (location/weather/Overpass), and reusable UI components, which improves maintainability and supports future feature growth. With reliable error handling and refresh/retry options, the app demonstrates real-world readiness and can be extended into a fully production-ready travel assistant with saved itineraries, user profiles, and smarter recommendations.