AMOV City Guide — Technical Report

Flutter mobile application - City Guide

Afonso David – 2023132572

Luccas Galvan – 2022100410

Octavio Marote – 2020111324

4/Jan/2026

1. Overview

AMOV City Guide is a Flutter application that presents points of interest (POIs) organized by categories, shows the current weather for the selected city, and allows users to mark POIs as favorites. The implementation focuses on: (1) user-friendly navigation, (2) correct loading of POIs from a JSON asset, (3) weather acquisition via Open-Meteo, (4) local persistence of favorites using SharedPreferences, and (5) adapting the POI detail layout to portrait and landscape orientations.

2. Architecture and technical decisions

To keep the codebase maintainable and aligned with the assignment requirements, responsibilities are separated into models, services/repositories, and UI screens. External I/O is isolated from widget code, and data is normalized at the boundaries so that UI components can assume consistent types.

Main layers:

- Models: Weather and Poi represent normalized data used by the UI.
- Repository: PoiRepository reads the JSON asset, normalizes image paths, and provides filtered queries.
- Services: WeatherService performs HTTP requests and maps weather codes; FavoritesService persists favorite POI IDs.
- Screens: HomeScreen, CategoriesScreen, PoiListScreen, PoiDetailScreen, FavoritesScreen handle navigation and presentation.
- Widgets: PoiCard provides consistent list item UI.

Key rationale:

- Async operations stay outside the UI whenever possible (services/repository), keeping widgets simple and testable.
- Favorites are persisted as POI IDs (strings), not entire objects, enabling quick lookups and easy reconstruction from JSON.
- Category support accepts both a single category field and a categories list, preventing failures when the dataset format changes.
- Navigation uses a consistent pattern: BottomNavigationBar for top-level sections and Navigator.push for drill-down.

3. Interface and interaction with the user

The Home screen is the main container of the application. It exposes two top-level sections—Categories and Favorites—via a BottomNavigationBar. Weather is displayed at the top of the Home content so it remains visible regardless of the selected section. Lists use Card + ListTile patterns for clear tap targets and consistent spacing.

User feedback and states:

- Loading states are shown with CircularProgressIndicator while futures are pending.
- Errors are surfaced as simple text messages in the corresponding screen widgets.

- Favorite changes are confirmed with a SnackBar in the POI detail screen.

4. Weather state acquisition (Open-Meteo)

Weather is fetched asynchronously from Open-Meteo using an HTTP GET request built from AppConstants (base URL, latitude, longitude). The Home screen stores the weather request as a Future and renders it with a FutureBuilder, providing a clean loading/error/success flow.

Flow summary:

- HomeScreen.initState initializes _weatherFuture by calling WeatherService().fetchCurrentWeather().
- HomeScreen._refreshWeather creates a new future inside setState to trigger a refetch.
- WeatherService.fetchCurrentWeather builds a Uri with query parameters and parses the response into a Weather model.
- Weather.fromOpenMeteo normalizes JSON values into safe numeric fields used by the UI.
- WeatherService.describeWeatherCode maps numeric codes to short descriptions for display.

5. Correct loading of POIs from the JSON file

POIs are stored in a local asset JSON file. PoiRepository loads and decodes the asset using rootBundle.loadString, supporting either a top-level list of POIs or a JSON object containing a 'pois' list. During parsing, Poi.fromJson normalizes fields and category formats. Image paths are normalized using PoiRepository.toAssetImagePath so the UI can reliably load assets with Image.asset.

Normalization decisions:

- Categories: accept either 'category' (string) or 'categories' (list) and always store List<String> categories.
- Average price: parse as double with safe fallback (0.0).
- Image path: accept 'images/x.jpg', 'assets/images/x.jpg', or 'x.jpg' and normalize to an assets/ path.

6. Consistent navigation between options

Navigation is intentionally split into two levels: (1) top-level tab switching (Categories / Favorites) handled by BottomNavigationBar, and (2) drill-down navigation using Navigator.push for category -> POI list -> POI detail. Favorites also navigates to POI details and triggers a refresh when returning to ensure the list remains accurate.

Navigation points:

- HomeScreen uses BottomNavigationBar to switch between CategoriesScreen and FavoritesScreen without pushing routes.
- CategoriesScreen opens PoiListScreen(category) with Navigator.of(context).push(MaterialPageRoute(...)).

- PoiListScreen opens PoiDetailScreen(poi) with Navigator.of(context).push(MaterialPageRoute(...)).
- FavoritesScreen opens PoiDetailScreen(poi) and then calls _refresh() on return.

7. Persistence of favorites with SharedPreferences

Favorites are persisted locally using shared_preferences. The persisted value is a string list of POI IDs under the key 'favorite_poi_ids'. This approach is storage-efficient and decouples persistence from the POI object structure, allowing the app to reconstruct favorites by filtering the POIs loaded from JSON.

Persistence flow:

- FavoritesService.loadFavoriteIds reads the stored list and returns a Set<String> for efficient contains checks.
- FavoritesService.isFavorite is a convenience method used by PoiDetailScreen during initialization.
- FavoritesService.toggleFavorite adds/removes a POI ID and writes the updated list back to preferences.
- FavoritesScreen._load combines persisted IDs with PoiRepository.loadPois() to build the favorites list.
- PoiDetailScreen._toggleFav updates persistence and reflects the new state in the UI, showing a SnackBar.

8. Portrait and landscape support

Orientation support is implemented where it has the highest impact: the POI detail screen. The screen reads MediaQuery.of(context).orientation and switches layout accordingly. In landscape, a Row places the image and scrollable details side-by-side; in portrait, content is stacked vertically inside a SingleChildScrollView.

9. How the implementation reached the final result

The final design is a direct consequence of aiming for correctness first, then simplicity. The main guiding principles were: isolate I/O, normalize data early, and keep navigation patterns consistent. These principles reduced the number of widget states that had to be managed manually and helped ensure that every requirement could be demonstrated quickly during evaluation.

Notable decisions:

- Use FutureBuilder across screens to model async work (loading/error/data) without introducing extra state managers.
- Normalize POI categories and image paths at parsing/repository level, not in widgets.
- Persist favorites as IDs to avoid storing duplicate POI data and to keep SharedPreferences usage straightforward.
- Refresh favorites on return from the detail screen to prevent stale UI after an unfavorite action.

10. Key methods referenced in the implementation

The following methods are central to the required functionality. File locations are shown according to the imports used in the project (models/, services/, screens/, utils/).

Method	**File**	**Responsibility**	**How it works (summary)**
HomeScreen.initState()	screens/home_screen.dart	Bootstraps weather acquisition	Initializes _weatherFuture once with WeatherService().fetchCurrentWeather().
HomeScreen._refreshWeather()	screens/home_screen.dart	Manual refresh of weather	Calls setState and reassigns _weatherFuture, forcing FutureBuilder to refetch.
WeatherService.fetchCurrentWeather()	services/weather_service.dart	Fetch current weather	Builds Uri with lat/long + current_weather=true; performs HTTP GET; parses JSON to Weather.
Weather.fromOpenMeteo(...)	models/weather.dart	Normalize API JSON	Extracts current_weather fields and safely converts them to double/int with fallbacks.
PoiRepository.loadPois()	services/poi_repository.dart	Load POIs from asset JSON	Reads asset with rootBundle, supports list or {pois:[...]}, maps items to Poi.fromJson.
Poi.fromJson(...)	models/poi.dart	Normalize POI JSON	Accepts category or categories; parses average_price; stores categories as List<String>.
PoiRepository.loadCategories()	services/poi_repository.dart	Compute available categories	Builds a Set from all POI categories, sorts it, and returns a list.
PoiRepository.loadPoisByCategory(...)	services/poi_repository.dart	Filter POIs by category	Loads all POIs then returns only those where categories.contains(category).
PoiRepository.toAssetImagePath(...)	services/poi_repository.dart	Normalize image path	Converts relative paths into 'assets/images/...' consistently for Image.asset.
FavoritesService.toggleFavorite(...)	services/favorites_service.dart	Persist favorites	Reads ids, adds/removes poiId, writes back with setStringList, returns updated set.

FavoritesScreen._load()	screens/favorites_screen.dart	Build favorites list UI data	Loads favorite IDs then filters PoiRepository.loadPois() to those IDs.
PoiDetailScreen._toggleFav()	screens/poi_detail_screen.dart	Favorite button action	Disables button while saving, toggles persisted state, updates icon/label, shows SnackBar.
PoiDetailScreen.build()	screens/poi_detail_screen.dart	Orientation-aware layout	Checks MediaQuery orientation and chooses Row (landscape) vs Column (portrait) layout.