**Restaurant Survey Application Documentation**

**Version:** 1.0  
**Author:** Ziyad Khaled, Manar Mesbah, Rawan Walid, Mohamed Hany, Alaa Ragab

**1. Introduction & Overview**

The Restaurant Survey application is a Flutter-based mobile app designed to collect feedback from restaurant customers about their dining experience. It allows customers to select meals, rate them, and provide textual comments. The application also features a separate section for managers/administrators to view aggregated feedback and a verification step to access this admin dashboard.

**Key Objectives:**

* Allow customers to easily provide meal-specific feedback.
* Collect customer contact information (optional).
* Provide a secure dashboard for managers to view feedback trends.
* Utilize modern technologies for a smooth user experience and efficient data management.

**2. Core Features**

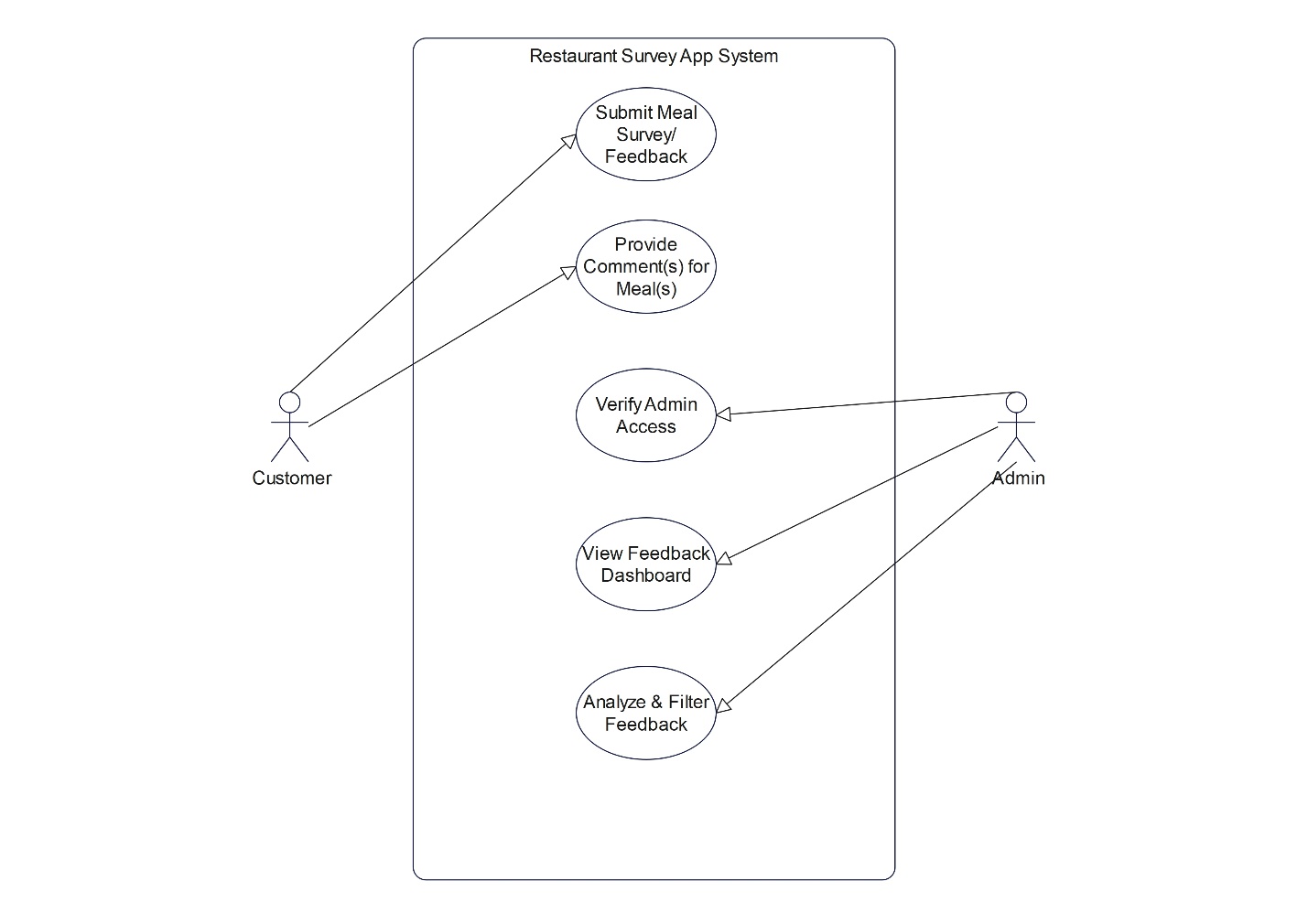
* **User Roles:**
  + **Customer:** Can browse a menu, select meals, rate them, provide comments, and optionally submit their name and phone number.
  + **Manager/Admin:** Can access an admin dashboard to view meal ratings and comments after passing a verification step.
* **Meal Selection & Feedback:**
  + Customers can select multiple meals from a predefined list.
  + Each selected meal can be rated (1-5 stars, half-star capable).
  + Textual feedback can be provided for each meal.
* **Admin Dashboard:**
  + Displays aggregated meal ratings and comments.
  + Features filtering options (Best, Worst, Average, All meals).
* **Verification:**
  + A simple numeric code verification for accessing the admin dashboard.
* **Data Persistence:**
  + Feedback data (meal name, rating, comment, customer name, customer number) is stored in a Supabase backend.
* **User Interface:**
  + Consistent UI theme with gradient backgrounds and Material Design components.
  + Reactive UI updates using GetX.

**3. Technical Architecture**

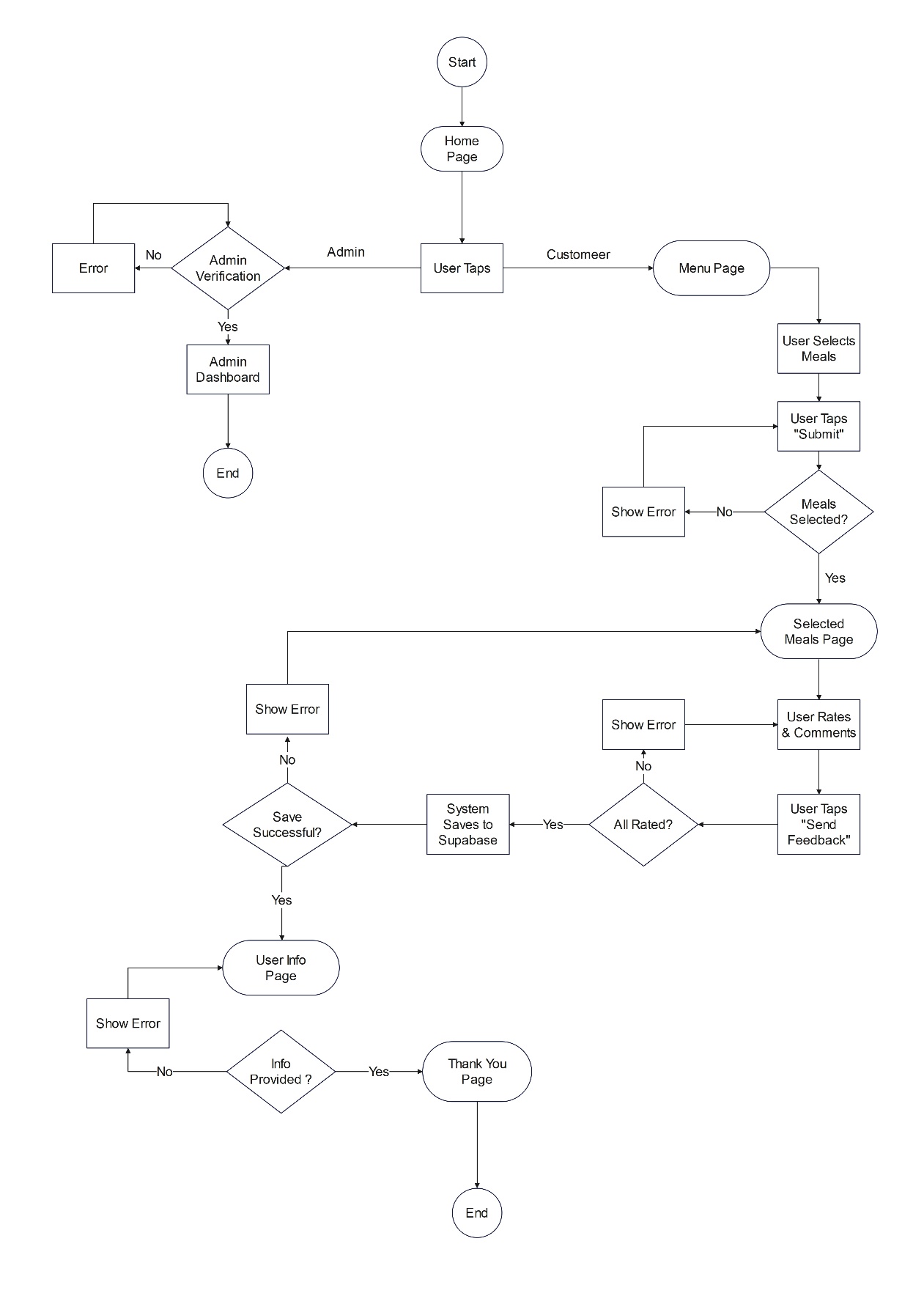
* **Framework:** Flutter
* **Language:** Dart
* **State Management:** GetX (for reactive state management, dependency injection, and navigation).
* **Backend & Database:** Supabase
  + **Database Table:** 'Mero' table stores feedback (meal name, rating, feedback text, customer name, customer number, timestamps).
* **Key Packages Used:**
  + flutter/material.dart: Core Flutter UI framework.
  + get/get.dart: For state management, navigation, and utilities.
  + supabase\_flutter/supabase\_flutter.dart: For Supabase integration.
  + flutter\_rating\_bar/flutter\_rating\_bar.dart: For star rating UI.
  + readmore/readmore.dart: For expandable text in the admin dashboard comments.
* **Project Structure:** (Based on provided files)
  + lib/
    - main.dart: App entry point, Supabase initialization, root widget.
    - home\_page.dart: Initial screen with Customer/Manager role selection.
    - menu\_page.dart: Customer screen to select meals.
    - menucontroller.dart: GetX controller for managing meal list and selections.
    - mealclass.dart: Data class for Meal (name, image).
    - selected\_meals\_page.dart: Customer screen to rate and comment on selected meals.
    - model.dart: Data class Model for feedback entries (for Supabase).
    - user\_information.dart: Customer screen to enter name and phone.
    - ThankYouPage.dart: Final screen for customers after submission.
    - AdmineVerivicationPage.dart (Typo, suggested: AdminVerificationPage.dart): Manager login screen.
    - admine\_page.dart (Typo, suggested: AdminPage.dart): Manager dashboard.
    - manar.dart: Appears to be a development/testing file for Supabase interactions.
  + assets/: Contains images for meals (e.g., assets/burger.jpg).

**4. System Diagrams**

**4.1. Use Case Diagram**



* **Description:** This diagram illustrates the primary interactions between the users (Customer, Manager) and the core functionalities of the Restaurant Survey application.

**4.2. User Flow Diagram (Customer Feedback Journey)**

* **Description:** This diagram visually maps the step-by-step journey a customer takes to submit feedback, from selecting meals to receiving a thank you message, including key decision points and error paths.

**5. Application Flow (User Journeys)**

**5.1. Customer Flow**

1. **Launch App (main.dart):** Initializes Supabase and starts MyApp.
2. **Home Page (HomePage):**
   * User selects "Customer".
   * CustomMenuController is initialized/reset.
   * Navigates to MenuPage.
3. **Menu Page (MenuPage):**
   * Displays a list of meals (from CustomMenuController) with images and checkboxes.
   * User selects desired meals. Selections are managed by CustomMenuController.
   * User clicks "Submit".
   * If no meals selected, a snackbar error is shown (user returns to MenuPage to select).
   * Otherwise, navigates to SelectedMealsPage, passing selected meal names as arguments.
4. **Selected Meals Page (SelectedMealsPage):**
   * Displays each selected meal with an image, name, star rating input, and comment field.
   * User provides ratings and comments.
   * **Local Validation:** A keyword-based check (isFeedbackConsistent) is performed to see if comments align with ratings. If inconsistent, an error snackbar is shown (user stays on SelectedMealsPage to revise).
   * User clicks "Send Feedback".
   * If any meal is unrated, a snackbar error is shown (user stays on SelectedMealsPage to complete).
   * Otherwise, feedback (for meals with rating > 0 or non-empty comment) is attempted to be saved to the 'Mero' table in Supabase.
   * If saving to Supabase fails, an error snackbar is shown (user may stay on SelectedMealsPage or an error state is handled).
   * On successful save, the app navigates to UserInfoPage, passing a list of insertedIds (IDs of the feedback records in Supabase).
5. **User Information Page (UserInfoPage):**
   * Prompts the user to enter their name and phone number.
   * User clicks "إرسال" (Send).
   * If fields are invalid or saving info fails, an error snackbar is shown (user stays on UserInfoPage to correct/retry).
   * If fields are valid and info is saved (updating Supabase records), the app navigates to ThankYouPage (using Get.off), passing the user's name.
6. **Thank You Page (ThankYouPage):**
   * Displays a thank you message personalized with the user's name.
   * "Back to Home" button navigates the user back to HomePage, clearing the feedback flow from the navigation stack.

**5.2. Manager/Admin Flow**

1. **Launch App (main.dart):** Initializes Supabase and starts MyApp.
2. **Home Page (HomePage):**
   * User selects "Manager".
   * Navigates to AdminVerificationPage.
3. **Admin Verification Page (AdminVerificationPage):**
   * Prompts for a verification code.
   * **Current Logic:** If the code is a number between 1 and 10 (inclusive), verification passes.
   * On successful verification, navigates to AdminPage.
   * Otherwise, an error dialog is shown (user stays on AdminVerificationPage to retry).
4. **Admin Page (AdminPage):**
   * Displays "📊 Admin Dashboard".
   * Fetches all meal feedback data from the 'Mero' table in Supabase.
   * Aggregates comments per meal and calculates average ratings.
   * Displays a list of meals with their average ratings and comments (using ReadMoreText for long comments).
   * Provides filter buttons ("أفضل الوجبات" - Best, "أضعف الوجبات" - Worst, "المتوسطة" - Average, "الكل" - All) to sort/filter the displayed meals based on ratings.
   * (User session ends, or navigates away/logs out - implicitly ends here for now).

**6. Detailed Module/Screen Descriptions**

* **main.dart:** App initialization, Supabase setup, root widget (MyApp using GetMaterialApp).
* **home\_page.dart:** Role selection screen, navigates to Customer or Manager flow.
* **menu\_page.dart:** UI for meal selection, interacts with CustomMenuController.
* **menucontroller.dart:** GetX controller managing meals list (hardcoded image paths) and selectedItems (observable list of booleans).
* **mealclass.dart:** Simple Meal data class (name, image).
* **selected\_meals\_page.dart:** UI for rating and commenting on meals, local feedback consistency check, Supabase submission.
* **model.dart:** Model data class for feedback, with fromJson and toJson for Supabase.
* **user\_information.dart:** UI for collecting user name/phone, updates Supabase records.
* **ThankYouPage.dart:** Final customer thank you screen, navigates back to home.
* **AdminVerificationPage.dart:** Manager verification screen with hardcoded password logic (1-10).
* **AdminPage.dart:** Manager dashboard, fetches and displays aggregated feedback from Supabase, includes filtering.
* **manar.dart:** Appears to be a development/testing widget for Supabase read/write operations, not part of the main user flow.

**7. Data Models**

* **Meal (mealclass.dart):**
  + name: String (Name of the meal)
  + image: String (Asset path for the meal image)
* **Model (model.dart):** Represents a feedback entry for Supabase.
  + id: int? (Auto-generated by Supabase)
  + createdAt: DateTime? (Timestamp, created\_at in JSON)
  + mealName: String (meal\_name in JSON)
  + rating: int
  + feedback: String
  + customerName: String? (customer\_name in JSON)
  + customerNumber: String? (customer\_number in JSON)
  + Includes fromJson, toJson, copyWith methods.
* **MealRating (in AdminPage.dart):** A local class used within the admin page to hold aggregated data.
  + name: String
  + averageRating: double
  + comments: List<String>

**8. External Services & Integrations**

* **Supabase:**
  + **URL:** https://kwombdxzzieovaxexnei.supabase.co
  + **Role:** Primary backend for data storage. The 'Mero' table stores all meal feedback and associated customer information.
  + **Operations:** insert, select, update.

**9. Potential Improvements & Areas for Attention**

1. **Admin Verification Code:**
   * The current verification code (1-10) is insecure.
   * **Recommendation:** Implement proper authentication for admins (e.g., Supabase Auth with specific admin roles) or store a hashed password securely.
2. **Error Handling & User Feedback:**
   * While snackbars are used, consider more comprehensive error handling (e.g., logging errors to a service, more specific error messages for different failure types).
   * Ensure loading indicators are present for all network operations (Supabase calls) to improve UX.
3. **Hardcoded Data:**
   * The meals list in CustomMenuController is hardcoded. Consider fetching this from Supabase to make the menu dynamically manageable.
   * Filter strings in AdminPage are hardcoded; could be dynamic if needed.
4. **Code Reusability & Organization:**
   * Some UI elements (e.g., gradient backgrounds, styled buttons) are repeated. Consider creating reusable custom widgets.
   * Correct typographical errors in filenames (e.g., AdmineVerivicationPage to AdminVerificationPage, admine\_page to AdminPage).
5. **State Management with GetX:**
   * The current use of GetX is effective. For larger features, explore GetX Services and Bindings for better organization.
6. **Testing:**
   * Implement unit tests for controllers and services.
   * Implement widget tests for UI components.
   * Consider integration tests for user flows.
7. **Null Safety and Robustness:**
   * Continue to ensure robust null checking and handling of potential null values, especially from API responses or user input.
8. **Admin Dashboard Performance:**
   * For a very large number of feedback entries, the current client-side aggregation in AdminPage might become slow. Consider server-side aggregation (e.g., using Supabase database views or functions) if performance becomes an issue.
9. **(Future Consideration) AI-Powered Comment Analysis:**
   * The openai\_service.dart file exists. If this feature is to be (re)integrated, ensure the API key is managed securely (e.g., via a backend function) and the service is properly called within the feedback submission flow. Consider latency and cost.

**10. Conclusion**

The Restaurant Survey application provides a solid foundation for collecting customer feedback and presenting it to management. It leverages Flutter for a cross-platform UI, GetX for efficient state management, and Supabase for a scalable backend. Addressing the highlighted security (admin verification) and usability (error handling, dynamic data) improvements will further enhance the application's robustness and professional quality.