BLOOMIFY - REPORT

CEP

Subject: MOBILE APPLICATION DEVELOPMENT

Instructor: Ma’am Mariam Memon

Team Members:

Maryam Qureshi (22SW078)

Table of Contents

Bloomify Application Development Report

1. Real-World Problem Identification

The primary problem Bloomify addresses is the difficulty novice and casual users face in identifying suitable house plants and tracking their care. Many existing plant databases or commerce sites are either overwhelming, focused purely on sales, or lack a centralized, easy-to-use system for personal curation.

Key Issues for Users:

- Discovery: Difficulty filtering large plant libraries based on simple criteria (e.g., "Indoor" vs. "Outdoor").
- Access to Information: Care details (light, watering) are often scattered or overly technical.
- Retention: No easy way to save and quickly reference preferred plants or potential purchases (Favorites).

Bloomify provides a clean, mobile-first experience to browse, filter, and save plants, solving these core retention and information overload problems.

2. Proposed Solution

The proposed solution is a cross-platform mobile application built with Flutter, structured around a clear separation of concerns using the Provider state management pattern.

Core Solution Components:

- Plant Catalog: Loads plant data from a local JSON asset (house_plants.json), parsed into the Plant data model.
- Filtering & Search: Implemented using the PlantProvider to enable real-time filtering by category (Indoor/Outdoor) and common name search without reloading the base data.
- User Persistence: Uses Hive for local, high-speed storage of user authentication and favorite plant lists.
- Intuitive UI: Features a main navigation shell (MainShell) with dedicated screens for the plant catalog (HomeScreen) and saved items (FavoritesScreen).

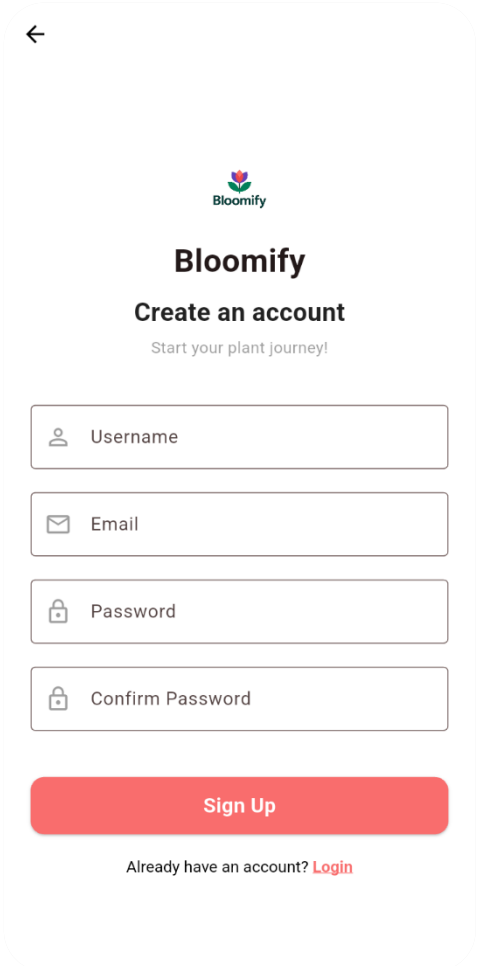
3. Responsive User Interfaces

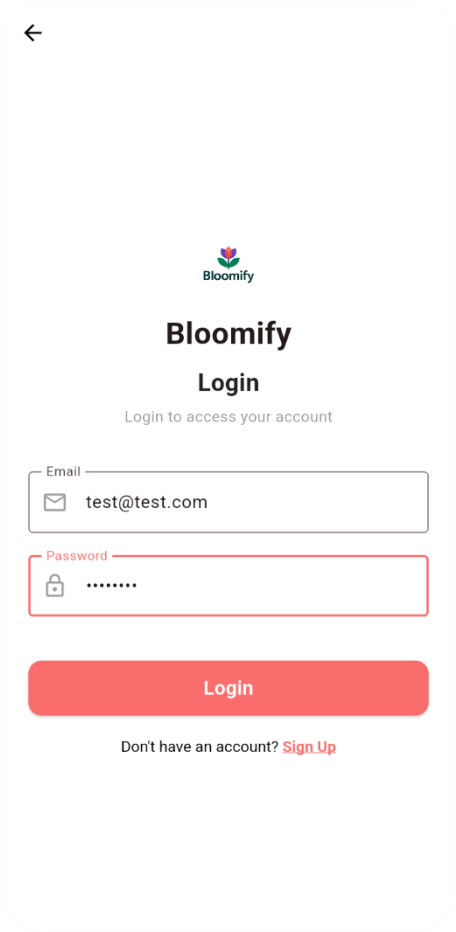
The Flutter framework ensures that the UI is responsive by design. The application utilizes GridView.builder and flexible widgets (like Expanded and Column/Row layouts) to adapt gracefully across different screen sizes, orientations, and platforms (mobile/desktop simulators).

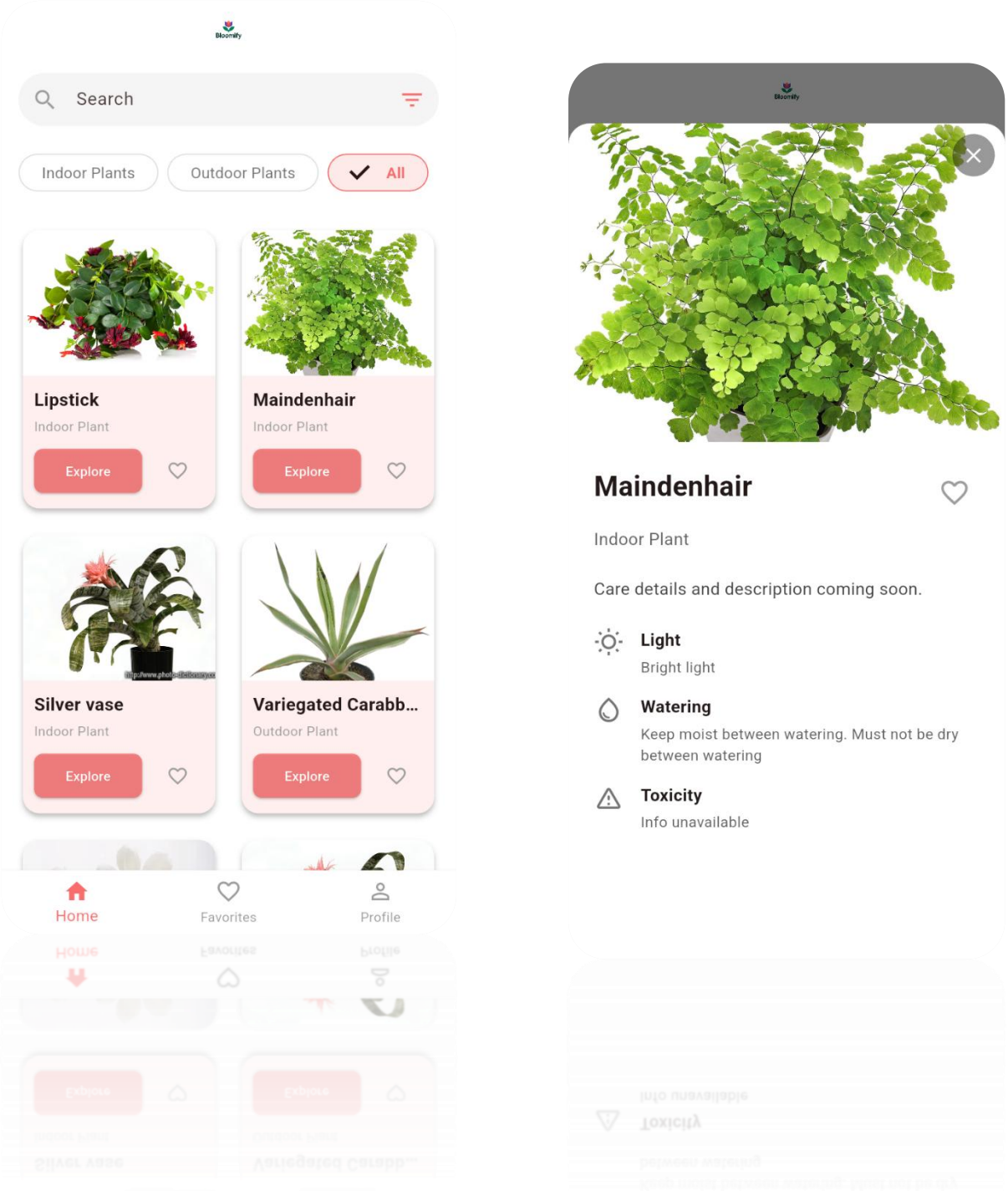
Key Responsive Elements:

- Grid Layout: The plant list in HomeScreen and FavoritesScreen uses GridView.builder with crossAxisCount: 2 (two columns), ensuring content is dense but legible on typical mobile screens.
- Detail Sheet: Plant details are shown using a DraggableScrollableSheet rather than a full-page navigation, allowing users to quickly peek at details while keeping the main catalog visible.

Screenshots:







4. Data Storage: Local Persistence with Hive

To satisfy the data storage requirement and create a useful, persistent application, a local data storage solution was chosen over a cloud database.

Function	Storage Mechanism	Justification
Plant Catalog Data (Read-Only)	Local JSON Asset (assets/house_plants.json)	Speed and Reliability. Since the plant data is static and pre-defined, storing it as a local asset ensures instantaneous loading without dependency on an external API or network connection.
Favorites & User Authentication	Hive Database (Local NoSQL Key-Value Store)	Performance and Simplicity. Hive is a lightweight, high-speed local database ideal for mobile applications. It handles simple key-value pairs (email: password for users, botanicalName: commonName for favorites) asynchronously, providing persistence without the overhead of complex relational databases or cloud sync.

5. Optimal State Management: provider

To manage the application's state and ensure the code is maintainable, we incorporated the provider package.

Justification for using Provider:

- Separation of Concerns: Provider allows a clean separation of our app's logic (Providers) from the display (UI Widgets), adhering to modern coding standards.
- Maintainability: Logic related to data fetching (PlantProvider) and persistence (FavoritesProvider) are cleanly encapsulated, making updates and debugging straightforward.
- Performance: Provider efficiently rebuilds only the specific widgets that need to change (e.g., updating a favorite icon when toggled), preventing unnecessary re-renders across the application.

6. APIs/Packages/Plug-ins Used

Package	Purpose	Justification
hive / hive_flutter	Local NoSQL persistence for user accounts and favorites.	Chosen for its superior speed compared to SQLite or Shared Preferences for non-relational mobile data, simplifying the persistence layer.
provider	State Management (ChangeNotifier)	Chosen for its simplicity and scalability in small to medium applications. It allows logic (data loading, filtering, favorites) to be separated from the UI, resulting in cleaner, testable, and maintainable code.

7. Issues and Bugs Encountered and Resolved during Development

Issue/Bug	Description	Resolution
Asset Loading Failure	Plant images were not loading, resulting in placeholders/error icons on the HomeScreen.	Two-Part Fix: 1. The pubspec.yaml was updated to include the assets/plant_images/ folder. 2. The Plant.fromJson logic was corrected to strip erroneous prefixes (e.g., assets/) and correctly construct the required file path (assets/plant_images/filename.ext).
Dependency Resolution Error	flutter pub get failed due to an error: "A dependency may only have one source."	Indentation Fix: The pubspec.yaml file had incorrect YAML indentation, placing provider: ^6.1.2 inside the flutter: dependency block. Corrected the indentation to resolve the dependency tree error.
State Management Overload	Initial code logic for loading, filtering, and favoriting was handled locally within StatefulWidgets, leading to tight coupling and potential bugs (e.g., favorite changes not updating the HomeScreen).	Provider Refactoring: Migrated all plant fetching/filtering logic to PlantProvider and all Hive interactions to FavoritesProvider. UI widgets now use Consumer or context.watch/read to react to state changes efficiently.
Asynchronous Context Warning	The Dart linter flagged use_build_context_synchronously warnings in async methods (_signUp in SignUpScreen).	Safety Check Implementation: Added the standard if (!mounted) return; check before any use of BuildContext (like navigation) following an await call to prevent potential errors if the widget is disposed.
Deprecated API Warning	The use of Color.withOpacity() triggered deprecated_member_use warnings across multiple files.	Style Upgrade: Replaced deprecated color.withOpacity(value) calls with the recommended color.withAlpha((value * 255).round()) method, improving color precision and code modernism.

8. GitHub Link

https://github.com/MaryamQureshii/bloomify