Academic Year 2021/2022

Politecnico di Milano

Design and Implementation of Mobile Applications

Design Document

Lyla Demange, Nils Mittelhockamp, Marion Henriot

Professor Luciano Baresi

January 14, 2026

Contents

1. Introduction

1.1 Project Overview

1.1.1 Project description

DillyDaily is a mobile application for recipe management, meal planning and prepping. It allows users to:

- Discover recipes
- Plan their weekly meals
- Manage their shopping list
- Customize their food preferences in terms of allergies, diets and their available equipment
- Create and modify their own recipes

The application, called **DillyDaily**, is a mobile recipe management and meal planning solution designed to simplify weekly meal organization and grocery shopping. The application addresses common challenges faced by individuals who want to organize their meals efficiently while respecting dietary restrictions and available cooking resources.

DillyDaily operates as a standalone mobile application with local data storage, eliminating the need for user authentication while maintaining personalized preferences directly on the device. The application communicates with an external server exclusively for retrieving the recipe database, ensuring users always have access to an up-to-date collection of culinary options.

Key characteristics of the system include:

- **Comprehensive recipe database**: A curated collection of recipes, continuously expanding, stored on an external server and accessible through RESTful communication.
- **Intelligent meal planning**: A 7-day meal planning system with intuitive drag-and-drop functionality, allowing users to organize up to two dishes per day.
- **Automated grocery list generation**: The system automatically compiles ingredients from planned meals, calculates approximate costs, and organizes items by category for efficient shopping.
- **Personalized cooking profiles:** Users define their dietary restrictions, available kitchen equipment, and cooking patience level to receive tailored recipe recommendations.

- **Step-by-step cooking mode**: An interactive cooking assistant that guides users through recipe preparation with sequential step navigation.
- **Custom recipe creation:** Users can create, edit, and personalize recipes, including the ability to modify existing recipes with custom photos and notes.

1.1.2 Targeted Audience

The application targets individuals seeking better meal (and shopping) organization, people with dietary restrictions requiring careful ingredient filtering, and others looking for inspiration adapted to their constraints and preferences.

1.1.4 Platform

- **Framework:** Flutter
- **Langage:** Dart
- **Availability:** Android, iOS, Web

1.2 Features :

- The user can add a recipe they like to their meal plan
- When wanting to cook the user simply has to go to the recipe on their meal plan and click on “Let’s cook” to get the cooking instructions

1.2.1 Set cooking profile and preferences

Users can customize their culinary experience by defining multiple preference dimensions:

Dietary Restrictions: The system supports comprehensive dietary filtering through two mechanisms:

- Four pre-configured dietary profiles: Vegan, Vegetarian, Gluten-free, and Halal
- Custom ingredient exclusions: Users can search for any ingredient in the database and mark it as either an allergen (☠️) or a disliked ingredient (🤮)

Available Equipment: Users specify which of four essential cooking appliances they have access to:

- Oven
- Mixer
- Microwave
- Fryer

Recipes requiring unavailable equipment are automatically filtered from search results unless explicitly disabled by the user.

Cooking Patience Level: A unique five-level energy scale ranging from "might not eat" to "love elaborate meals" allows the system to filter recipes by preparation time, particularly beneficial for students and busy individuals with limited time for meal preparation.

These preferences are stored locally on the device and automatically applied to filter the recipe catalog, ensuring users only see recipes compatible with their constraints.

1.2.2 Browse and discover recipes

The Explore page provides multiple discovery mechanisms:

Search Functionality: Users can search recipes through three criteria:

- Recipe name
- Ingredient content
- Energy level (preparation time)

Automatic Filtering: The system applies three layers of automatic filtering based on user profile:

- Dietary restrictions and ingredient exclusions
- Available kitchen equipment
- Cooking patience level

Users can selectively disable these automatic filters to explore recipes outside their usual constraints.

Organization and Display:

- Favorite recipes appear in a carousel at the top of the page for quick access
- Remaining recipes are displayed in a scrollable grid
- When search criteria are active, all matching recipes are displayed regardless of their favorite status

Recipe Preview: Tapping any recipe opens a detailed preview showing preparation time, and estimated cost before adding it to the meal plan.

1.2.3 Plan weekly meals

The Meal Plan page implements an intuitive drag-and-drop interface for organizing the week's meals:

Timeline Structure:

- 7-day fixed planning window
- The current day appears first, followed by the remaining days of the week
- Each day contains 2 meal slots

Interaction Model:

- Selected recipes appear in a carousel at the top of the page
- Users drag recipes from the carousel to desired meal slots
- Recipes can also be dragged between slots within the timeline
- The same recipe can be added to multiple slots through repeated drag-and-drop operations
- Dragging a recipe outside the timeline removes it from that specific slot
- Removing a recipe from the meal plan carousel removes it from all timeline slots

Recipe Management:

- Tapping a recipe in the meal plan opens a dedicated preview
- From this preview, users can launch the "Let's Cook!" mode
- Users can also add the recipe to the grocery list with quantity control

Grocery List Integration:

Each recipe must be individually added to the grocery list through its preview. The system intelligently suggests quantities:

- If the recipe is not in the timeline: defaults to 1x
- If the recipe appears in n slots: suggests n×
- Users can adjust quantities using +/- interface controls
- Each recipe serving is calculated based on the number of people specified in user preferences (1 recipe = 1 meal × nb people)

This approach provides flexibility and transparency, allowing users to control exactly which recipes contribute to their shopping list.

1.2.4 Manage grocery list

The Groceries page provides comprehensive shopping list management:

Automatic List Generation: Ingredients are automatically compiled from the meal plan, with quantities aggregated when the same ingredient appears in multiple recipes.

Price Calculation: Each ingredient in the database includes unit pricing, allowing the system to calculate and display the approximate total cost of the shopping list through simple addition of ingredient prices and quantities.

Category Organization: Ingredients are organized into 10 shopping categories:

- Produce
- Dairy Products
- Meat & Deli
- Breads & Grains
- Breakfast
- Canned Goods
- Sauces & Condiments
- Snacks & Drinks
- Baking & Seasonings
- Other

Manual List Generation: Users build their grocery list by individually adding recipes from the meal plan. Each recipe preview in the meal plan includes:

- A quantity selector (+/- interface) with intelligent defaults:
  - 1× if the recipe is not scheduled in the timeline
  - n× if the recipe appears in n timeline slots
- Recipe quantities are automatically converted to ingredient quantities based on the number of people configured in user preferences

Ingredients from added recipes are automatically aggregated when the same ingredient appears in multiple selected recipes.

1.2.5 Cook with step-by-step guidance

The "Let's Cook" mode provides an interactive cooking experience:

Step-by-Step Navigation:

- One preparation step is displayed at a time
- Users navigate forward with the right arrow
- Backward navigation is supported for reviewing previous steps
- A progress indicator shows advancement through the recipe

Interface Design: The mode focuses user attention on the current step while maintaining context awareness through the progress indicator, eliminating the need to scroll through the entire recipe during preparation.

Note: The current version does not include integrated timers or step completion marking.

1.2.6 Create and customize recipes

Users can contribute to their personal recipe collection through a dedicated creation page:

Required Fields:

- Recipe name
- Description

Optional Fields:

- Photo upload (from gallery or camera)
- Ingredients list with quantities
- Preparation steps
- External website reference
- Equipment requirements
- Dish type categorization
- Estimated preparation time

Recipe Editing: All recipes in the database, including pre-loaded recipes, can be edited. The creation page is pre-populated with existing recipe data, allowing users to:

- Modify any field
- Add personal photos
- Include additional notes or variations
- Save as a personalized version

Custom and edited recipes are stored locally and seamlessly integrated in the Explore page. They can be browsed from the Write page.

1.2.7 Manage application settings

Users can configure application preferences including:

- Language selection (currently English only)
- Filter preferences (enable/disable automatic filtering)
- Data management (clear cache, reset preferences)

1.3 Purpose

The purpose of this document is to provide a comprehensive technical description of the DillyDaily application, developed as part of the **Design and Implementation of Mobile Applications (DIMA)** course at Politecnico di Milano during Academic Year 2025/2026.

This document provides a detailed explanation of the system architecture, design patterns, and technical decisions made during development. It also describes the development process, technologies employed, and integration strategies used to build the application.

1.4 Definitions, Acronyms, Abbreviations

1.4.1 Definitions

- **Android**: A mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets.
- **iOS**: A mobile operating system created and developed by Apple Inc. exclusively for its hardware
- **Flutter**: An open-source UI software development kit created by Google, used to develop cross-platform applications for Android, iOS, Linux, Mac, Windows, and Web from a single codebase.
- **Dart**: A programming language designed for client development, such as for the web and mobile apps. It is developed by Google and can also be used to build server and desktop applications.
- **Framework**: An abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

- **Local Storage**: Data persistence mechanism that stores information directly on the user's device rather than on remote servers.
- **RESTful**: A software architectural style that defines a set of constraints to be used for creating Web services, emphasizing scalability, statelessness, and uniform interfaces.

1.4.2 Acronyms

- **API**: Application Programming Interface - defines interactions between software intermediaries.
- **HTTP**: Hypertext Transfer Protocol - foundation of data communication for the web.
- **JSON**: JavaScript Object Notation - data interchange format.
- **REST**: Representational State Transfer - architectural style for distributed systems.
- **SDK**: Software Development Kit - provides tools, libraries, and documentation for developing applications.
- **UI**: User Interface - the space where interactions between humans and machines occur.

1.4.3 Glossary

- **Recipe**: A culinary preparation defined by its name, description, dish type, ingredients list, preparation steps, required gear and photo.
- **Dish Type:** Dish category (main course, dessert…), referenced in the Recipe object.
- **Gear:** Kitchen equipment (oven, stove, microwave, fryer)
- **Ingredient**: A food item stored in the database with properties including name, unit, unit price, category, and potential allergen information. Referenced in a recipe.
- **Meal Plan**: A list of selected recipes. They can be organized in a timeline across 7 days with up to 2 meal slots per day. The recipes in the Meal Plan can be added to the Grocery List and launched in the 'Let's Cook' mode.
- **Grocery List**: An automatically generated shopping list compiled from meal plan ingredients, organized by category with calculated total cost.

- **Cooking Profile/Personalization**: User-defined preferences including dietary restrictions, available equipment, and cooking patience level.
- **Energy Level:** A five-point scale representing user's available time and willingness for meal preparation, from "might not eat" to "love elaborate meals".

1.5 Reference Documents and Websites

- Flutter Documentation: https://docs.flutter.dev/
- Dart Documentation: https://dart.dev/guides
- Vercel Documentation: https://vercel.com/docs
- Microsoft Clarity Documentation: https://clarity.microsoft.com/
- JSON Specification: https://www.json.org/

2. General Architecture of the project

2.1 Overview

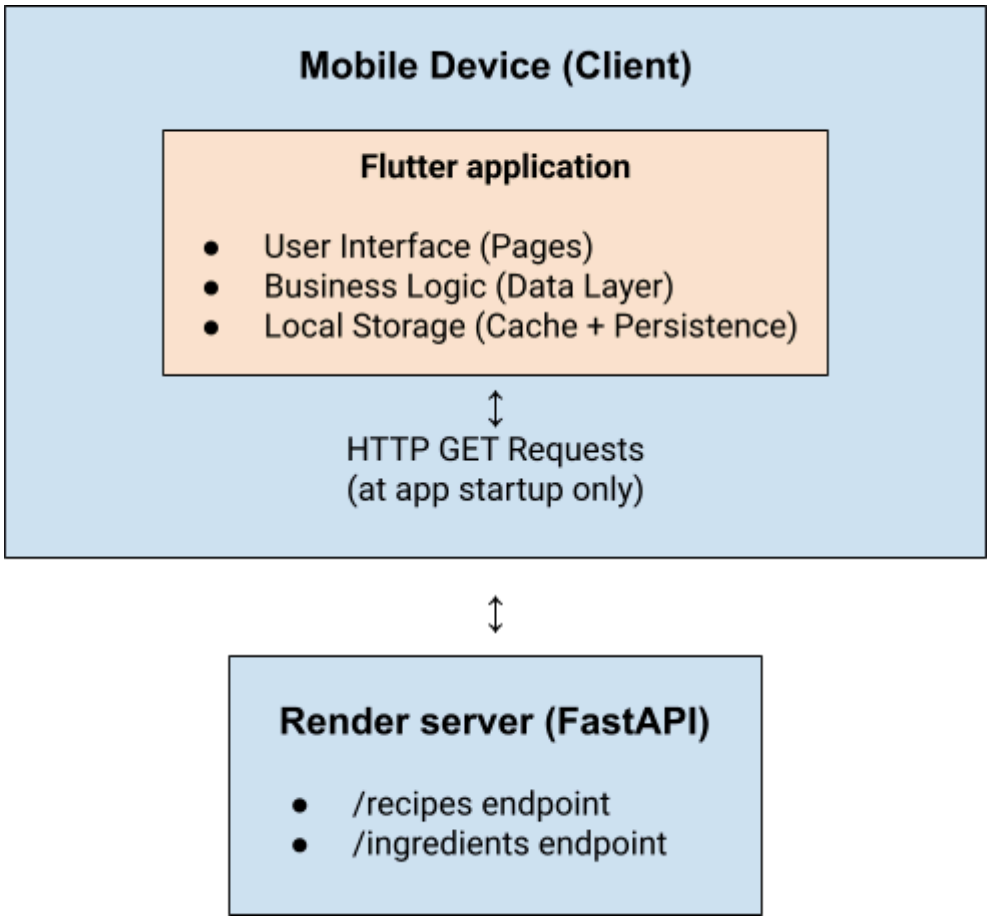
DillyDaily follows a simplified client-server architecture optimized for mobile deployment with offline-first capabilities. The system is designed to minimize server dependency while maintaining access to a centralized, continuously updated recipe database.

The architecture consists of two primary physical components:

- **Client Tier:** A Flutter mobile application running on Android or iOS devices that handles all user interactions, business logic, data processing, and local storage. The client is designed to operate independently with cached data, ensuring full functionality even without network connectivity.
- **Server Tier:** A lightweight FastAPI server hosted on Render (https://fastapi-example-da0l.onrender.com) that serves as a centralized repository for the recipe and ingredient databases, exposing two RESTful endpoints:
    - /recipes - Returns the complete recipe collection in JSON format
    - /ingredients - Returns the ingredient database in JSON format

This architecture prioritizes offline-first functionality and user autonomy, ensuring that users can access their personalized content, favorites, custom recipes, and meal plans regardless of network availability.

2.1.1 System Communication Flow



2.1.2 Data Retrieval Strategy

The application implements a startup synchronization pattern:

1. **Initial Load**: Upon application launch, the client issues HTTP GET requests to both server endpoints using Dart's http library:

```
await http.get(Uri.parse('https://fastapi-example-da0l.onrender.com/recipes'));
await http.get(Uri.parse('https://fastapi-example-da0l.onrender.com/ingredients'));
```

2. **Caching:** Retrieved data is immediately cached in memory for the duration of the session, eliminating the need for subsequent network requests.

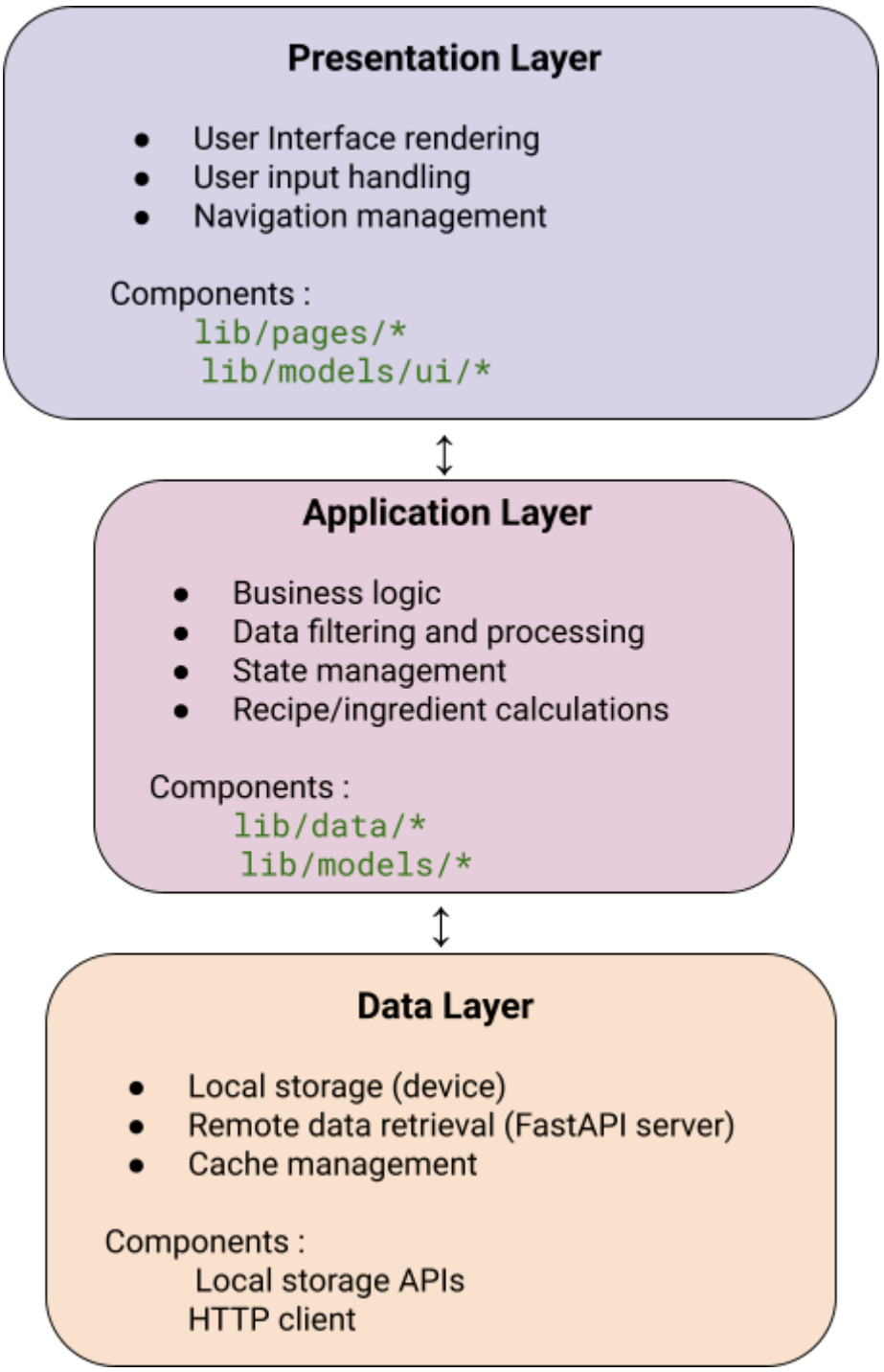
3. **Persistent Local Storage**: Critical data is persisted to device storage to enable offline functionality:
    - Favorite recipes: All recipes marked as favorites by the user
    - Custom recipes: Recipes created or edited by the user
    - Meal plan recipes: All recipes currently in the meal plan
    - User preferences: Cooking profile, dietary restrictions, equipment availability

This hybrid approach ensures that users always have access to their personalized content while benefiting from centralized recipe database updates.

2.2 Architectural Style and Patterns

2.2.1 Three-Layer Architecture

DillyDaily implements a classic three-layer architecture, with clear separation of concerns between presentation, business logic, and data management:



2.2.1.1 Presentation Layer

Role: Manages all aspects of user interface rendering and interaction handling. This layer is responsible for displaying data to users and capturing their input.

Components:

- `lib/pages/` : Contains the main application screens
  - `Explore/explore_page.dart:` Recipe discovery and browsing
  - `MealPlan/meal_plan_page.dart`: Weekly meal planning interface
  - `Groceries/groceries_page.dart`: Shopping list management
  - `Write/write_page.dart:` Recipe creation and editing
  - `Account/account_page.dart`: User preferences and settings
- `lib/models/ui/`: UI-specific components and widgets
  - `category_title_bloc.dart`: Category header components
  - Other reusable UI elements

Characteristics:

- All pages are implemented as `StatefulWidget` classes
- Each page manages its own state and that of its children
- Child components communicate state changes to parent pages through callback functions
- Uses Flutter's built-in `setState()` mechanism for state updates

2.2.1.2 Application Layer

Role: Implements all business logic, data processing, and filtering operations. This layer acts as an intermediary between raw data and the presentation layer.

Components:

- `lib/data/`: Business logic implementations
  - `recipes.dart`: Recipe filtering, search, and retrieval logic
  - `ingredients.dart`: Ingredient management and aggregation
  - `personalisation.dart:` User preference application and filtering
- `lib/models/`: Data structure definitions
  - `Recipe.dart`: Recipe model with ingredients, steps, equipment
  - `Ingredient.dart`: Ingredient model with quantities and units
  - `Personalization/`: User profile and preferences models

        - `my_recipes.dart`: Custom recipe management
        - User profile and dietary restriction models

Key Functions:

- Filtering recipes based on dietary restrictions
- Filtering recipes based on available equipment
- Filtering recipes based on cooking patience level
- Calculating grocery list totals from meal plan
- Aggregating ingredients across multiple recipes
- Managing recipe favorites and custom recipes

2.2.1.3 Data Layer

Role: Handles all data persistence, retrieval, and caching operations. This layer abstracts the source of data from upper layers.

Components:

- **Remote Data Source**: FastAPI server on Render
    - Provides centralized recipe database
    - Provides ingredient database with pricing
    - Accessed via HTTP GET requests at startup
- **Local Data Source:** Device storage
    - Caches full recipe and ingredient databases
    - Persists user-specific data (favorites, custom recipes, meal plans)
    - Stores user preferences and cooking profile
- **HTTP Client**: Dart's `http` library for network communication

2.2.2 RESTful Architecture

The server component follows REST principles, providing stateless endpoints that serve JSON data:

Endpoint Design:

- `GET /recipes`: Returns array of all recipes in JSON format
- `GET /ingredients`: Returns array of all ingredients with metadata

RESTful Characteristics:

- **Stateless**: Server maintains no session information about clients
- **Cacheable:** Responses are designed to be cached by clients
- **Uniform Interface:** Consistent JSON structure across endpoints
- **Client-Server Separation:** Clear separation between data provider (server) and consumer (mobile app)

The FastAPI framework was chosen for its:

- High performance for serving JSON responses
- Minimal configuration requirements
- Native Python support

2.2.3 State Management Pattern

DillyDaily employs a **lifted state pattern** using Flutter's native `setState()` mechanism:

Pattern Description:

- Each page is a `StatefulWidget` that owns and manages its state
- Child widgets are typically `StatelessWidget` instances
- When child widgets need to trigger state changes, they invoke callback functions defined in the parent page
- The parent page updates its state using `setState()`, triggering a rebuild of affected widgets

Advantages of This Approach:

- Simplicity: No external state management library required
- Predictability: State changes are explicit and localized
- Performance: Only affected subtrees are rebuilt
- Maintainability: Clear data flow from parent to children

This pattern is sufficient for DillyDaily's scope, where state is primarily page-specific and does not require complex cross-page data sharing.

2.3 Component Architecture

2.3.1 Data Models

The application defines several core data structures that represent the domain model:

2.3.1.1 Recipe Model (`lib/models/Recipe.dart`):

```
class Recipe {
   String id;                          // Unique identifier
   String name;                        // Recipe name
   String image;                       // Image path/URL
   String personalized;                // Customization flag
   List<Step> steps;                   // Preparation steps
   Map<String, double> ingredients;    // Ingredient name → quantity
   List<String> necessaryGear;         // Required equipment
   String summary;                     // Brief description
   int servings;                       // Number of servings
   String recipeLink;                  // External reference URL
   List<String> dishTypes;             // Categories (main, dessert, etc.)
 }
```

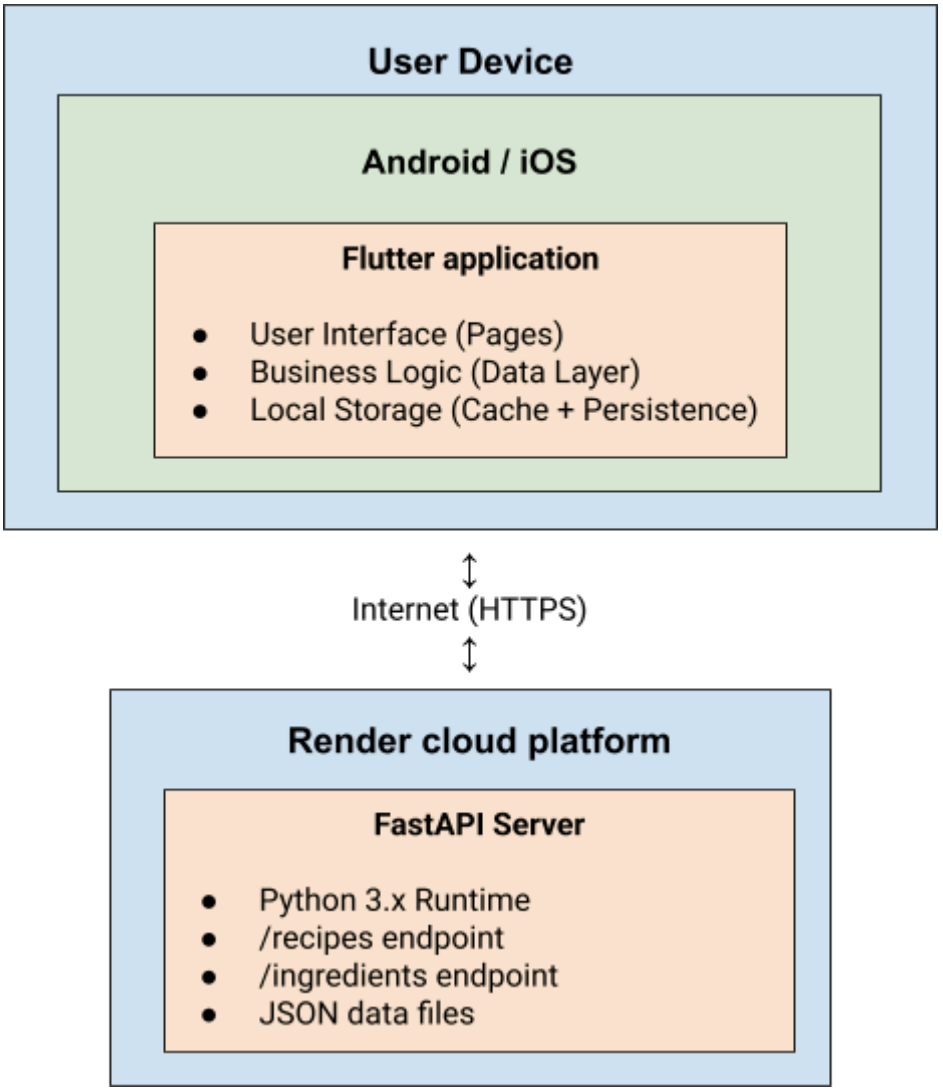
2.3.1.2 Step Model

```
class Step {
   String description;              // Step instructions
   Duration? duration;              // Optional timing
   StepType type;                   // "preparation" or "cooking"
   List<Ingredient> ingredients;    // Ingredients used in this step
}
```

2.3.1.3 UserProfile Model

```
class UserProfile {                  // Number of people to cook for
   int _defaultPersonNumber;         // Cooking patience level (1-5)
   double _patience;                 // IDs of favorite recipes
   List<String> _favoriteRecipes;    // 7 days × 2 slots meal plan
   List<List<String>> _weekMeals;    // Available equipment
   List<String> _kitchenGear;        // Custom recipes dictionary
   MyRecipes _myRecipes;             // Manual grocery items
   PersonalizedGroceries
_coursesPersonnelles;
}
```

2.3.2 Physical Deployment



1 - Client Device:

- Platform: Android (Google Play) or iOS (App Store)
- Runtime: Flutter SDK with Dart VM
- Storage: Device local storage for cached data and user preferences
- Distribution: Cross-platform single codebase

2 - Application Server:

- Platform: Render cloud hosting (free tier)
- Framework: FastAPI (Python)
- Endpoints: RESTful JSON API
- Storage: JSON files served as static data

2.4 Other Design Decisions

2.4.1 Absence of User Authentication

Decision: DillyDaily does not implement user authentication or account management in the current version.

Rationale:

- **Development Timeline**: Limited time and team experience with secure authentication implementation
- **Functional Sufficiency**: The current prototype operates effectively without user accounts, as all personalization is device-local
- **Reduced Complexity**: Avoiding authentication simplifies the architecture, eliminating the need for user databases, session management, and security infrastructure

Current Implications:

- All user data is stored locally on the device
- No cloud backup of preferences or custom recipes
- No cross-device synchronization
- Enhanced privacy: user data never leaves the device

Future Considerations: Authentication would be the **first priority** for future development, as it would enable:

- User-to-user connections and social features
- Recipe sharing between users
- Cloud backup and synchronization
- Multi-device access to meal plans and preferences

2.4.2 Offline-First Architecture with Local Storage

Decision: Implement local-first data storage with server synchronization only at startup.

Rationale:

- **Offline Functionality**: Users can access favorites, custom recipes, and meal plans without internet connectivity
- **User Autonomy**: The application functions independently of server availability
- **Privacy-First Approach**: Sensitive user preferences and dietary restrictions remain on the device

- **Performance**: Eliminates network latency during app usage after initial load

Implementation Strategy:

- All user-generated content persists locally
- Server provides read-only recipe database
- Cache invalidation occurs only at app restart

Trade-offs:

- No cross-device synchronization (acceptable for current scope)
- Users must restart app to receive recipe database updates
- Increased local storage usage (mitigated by selective persistence)

2.4.3 Technology Selection

2.4.3.1 Flutter and Dart

Decision: Build the mobile application using Flutter framework with Dart programming language.

Rationale:

- **Cross-Platform Development**: Single codebase deploys to both Android and iOS without platform-specific code
- **Performance**: Compiled to native code for both platforms
- **UI Consistency**: Material Design widgets provide consistent experience across platforms
- **Development Origin**: The project originated as a team member's personal exploration of Flutter, which organically evolved into a full academic project

Implementation Notes:

- No platform-specific widgets were used, ensuring full compatibility with both Android and iOS
- Material Design provides a modern, familiar interface paradigm
- Hot reload functionality significantly accelerated development iteration

2.4.3.2 Render for Server Hosting

Decision: Host the FastAPI server on Render's free tier.

Rationale:

- **Cost**: Free tier sufficient for academic project and prototype phase
- **Simplicity**: Minimal configuration required for Python/FastAPI deployment
- **Reliability**: Adequate uptime for a recipe database server with infrequent queries

Limitations Acknowledgments:

- Free tier may have cold start delays
- Limited to simple JSON serving (no complex database queries)
- Sufficient for current scale (22 recipes) but would require upgrade for production

2.4.3.3 JSON Data Format

Decision: Store recipe and ingredient data in JSON format rather than a traditional relational database.

Rationale:

- **Development Agility**: JSON structure easily modified during iterative development
- **Schema Flexibility**: Adding new fields or restructuring data requires no migration scripts
- **Deployment Simplicity**: No database server configuration or maintenance required
- **Rapid Iteration**: Changes to data structure never broke existing code
- **Manageable Scale**: Current data volume (22 recipes, complete ingredient list) easily handled in memory

Example Benefit: When the team decided to add the `personalized` field to recipes or expand the `Step` model to include ingredient references, these changes required only updating the JSON schema without cascading code changes.

Future Migration Path: For production deployment with user authentication and recipe sharing, the data layer would migrate to **SQL database** to support:

- User-generated recipe storage at scale
- Recipe sharing and permissions
- Full-text search capabilities
- Relational queries (e.g., finding all recipes using a specific ingredient)
- Data integrity constraints

The current JSON-based approach serves as an effective prototyping foundation while maintaining a clear migration path to more robust data management.

2.4.3.4 Clarity Analytics Integration

Decision: Integrate Microsoft Clarity for user behavior analytics.

Implementation:

- Heatmap visualization of user interactions
- Session duration tracking
- Screen visit frequency analysis
- Dead tap detection (UI elements users attempt to interact with but are not interactive)

Purpose:

- Identify usability issues through actual user behavior
- Prioritize feature improvements based on usage patterns
- Detect UI elements causing confusion (dead taps)

Privacy Consideration: No personally identifiable information is collected; only anonymous interaction patterns are tracked.

3 User Interface Design

3.1 Introduction

This section presents the main screens of the DillyDaily mobile application and describes the user interaction flows for core functionalities. The interface follows Material Design principles with a custom color scheme based on a vibrant pink primary color (**#844c72**).

The application uses a bottom navigation bar with five main sections: Write, Explore, Meal Plan, Groceries, and Account. The Explore page serves as the landing page to immediately present users with recipe inspiration.

3.2 Mobile Application Interface

3.2.1 Application Launch

Upon launching, users see a splash screen featuring the custom DillyDaily logo during data loading. The application employs a zero-friction onboarding approach with no mandatory setup—users can immediately browse recipes with default settings and configure preferences later through the Account page.

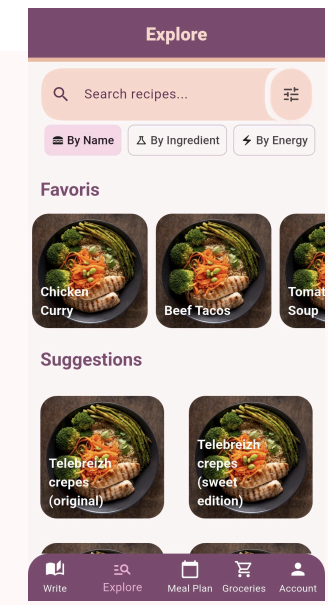
Splash Screen for Android (left) and for web applications (right)

3.2.2 Explore Page - Recipe Discovery

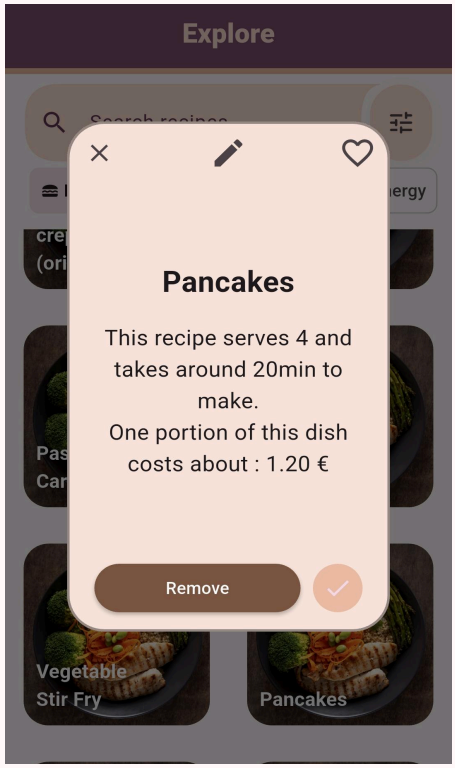
The Explore page provides recipe browsing with integrated search and filtering. A search bar at the top offers three filter modes (by name, by ingredient, by energy level). A settings icon reveals toggles for automatic filtering based on dietary restrictions, equipment, and cooking patience.

Favorite recipes appear in a horizontal carousel below the search controls. The remaining recipes display in a scrollable grid labeled "Suggestions."

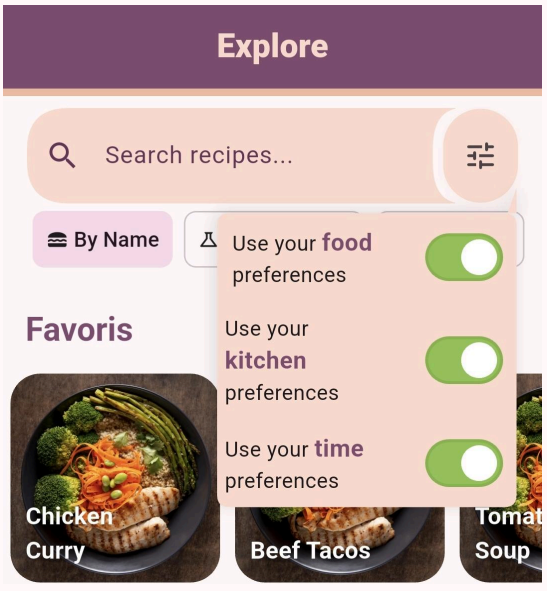
Tapping a recipe opens a modal preview showing the recipe name, servings, preparation time, estimated cost, and a large "Add" button to include it in the meal plan. The modal header includes close, edit, and favorite buttons.



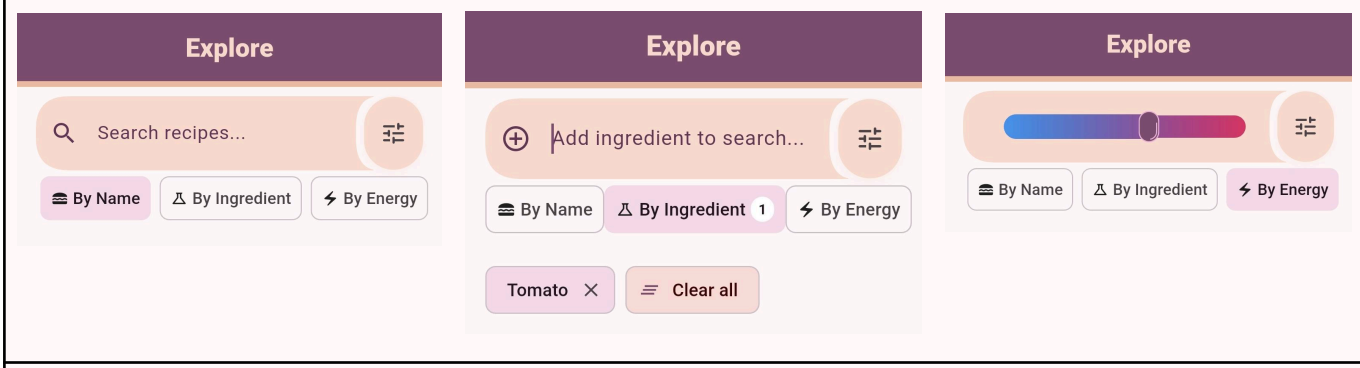
Screenshot of the full Explore page



Screenshot of Recipe Preview Modal

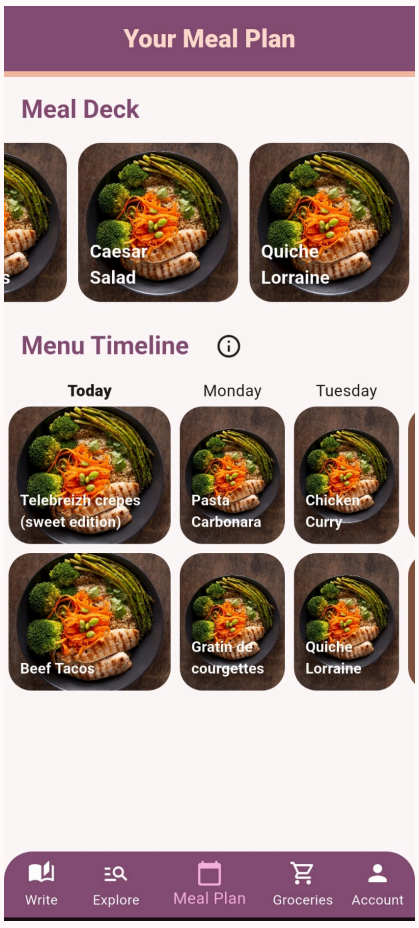


Screenshot of Automatic filter popover



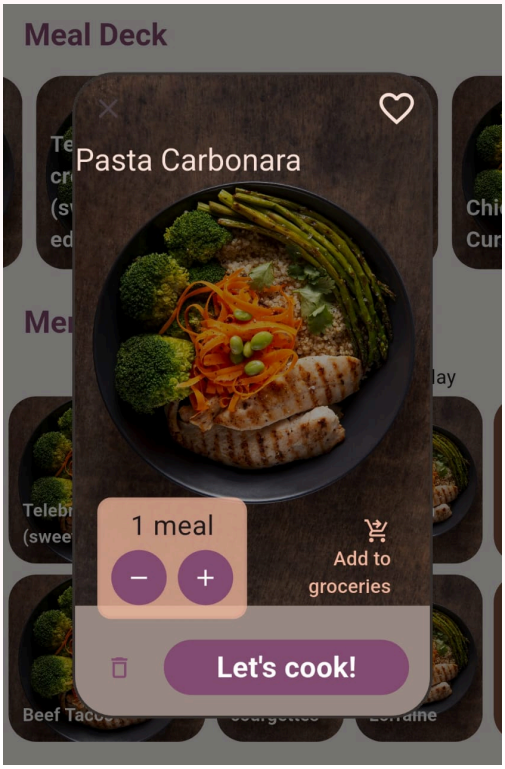
Screenshot of the search bar dynamic adaptation to the different selected filters

3.2.3 Meal Plan Page - Weekly Organization



Screenshot of full Meal Plan page

The Meal Plan page displays recipes in the meal plan as a horizontal carousel at the top. Below, a timeline shows 7 days with 2 meal slots each. The current day is labeled "Today" in bold with wider slots.
Users drag recipes from the carousel or existing slots to organize their week. An info button (ⓘ) provides drag-and-drop instructions via tooltip.



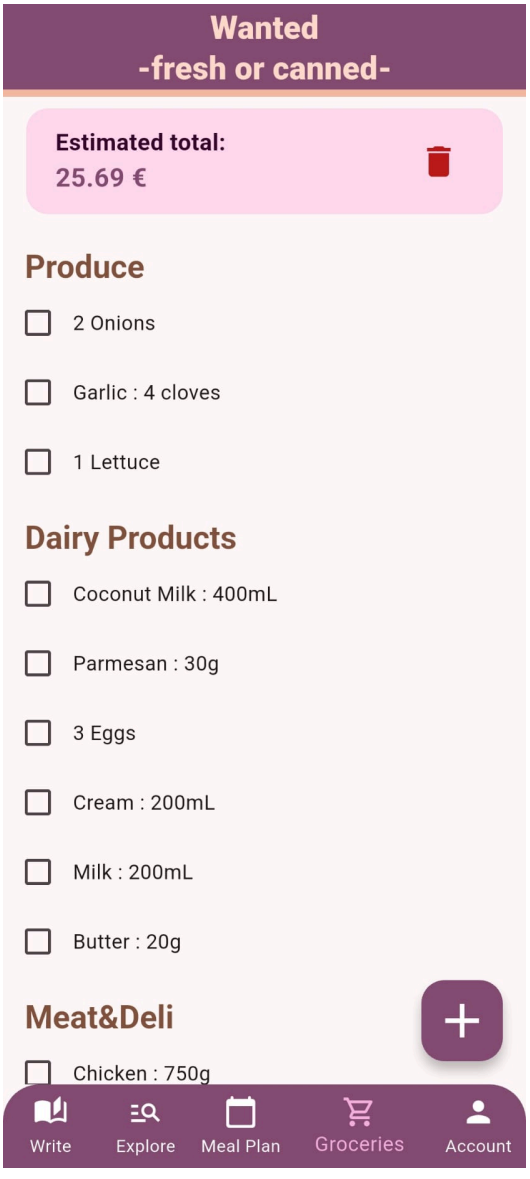
Screenshot of full Meal Plan page

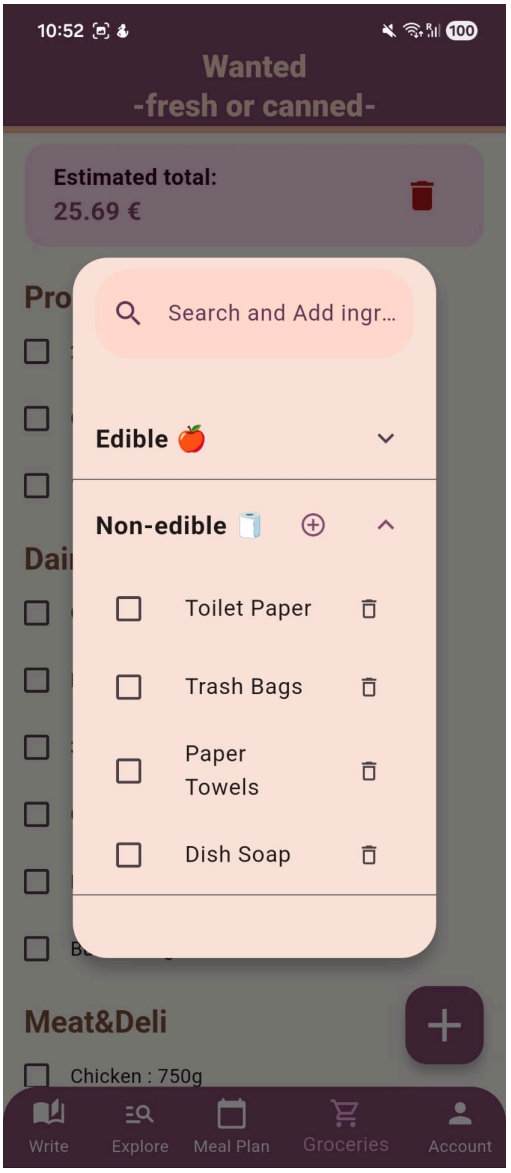
Tapping a recipe opens a specialized preview with "Let's Cook!" button, meal count display with +/− controls, and "Add to Groceries" button. A long-press delete button (🗑️) removes the recipe from the meal plan.

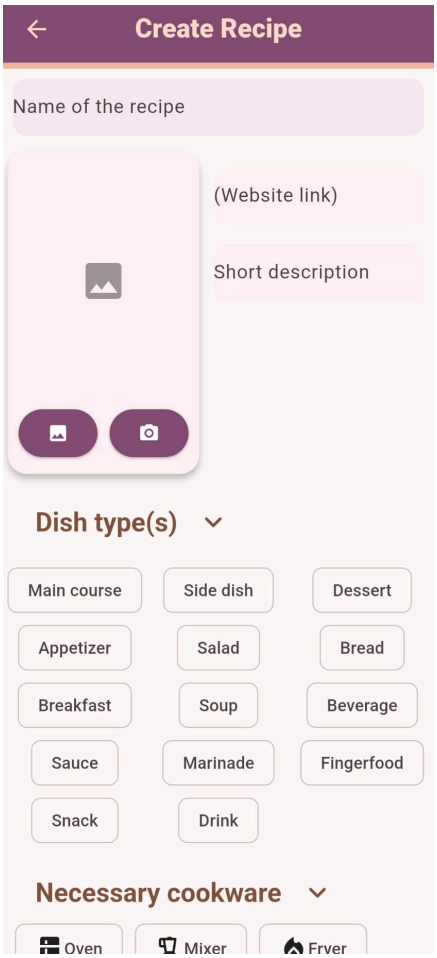
3.2.4 Groceries Page - Shopping List

The Groceries page displays ingredients grouped by category (Produce, Dairy, Meat, etc.) with checkboxes. The header shows the total cost and a clear button. Checked items appear struck-through and grayed but remain in place.

A floating + button opens a modal for adding items through three methods: ingredient search, selecting from the user's "Edible" list, or adding "Non-Edible" items.





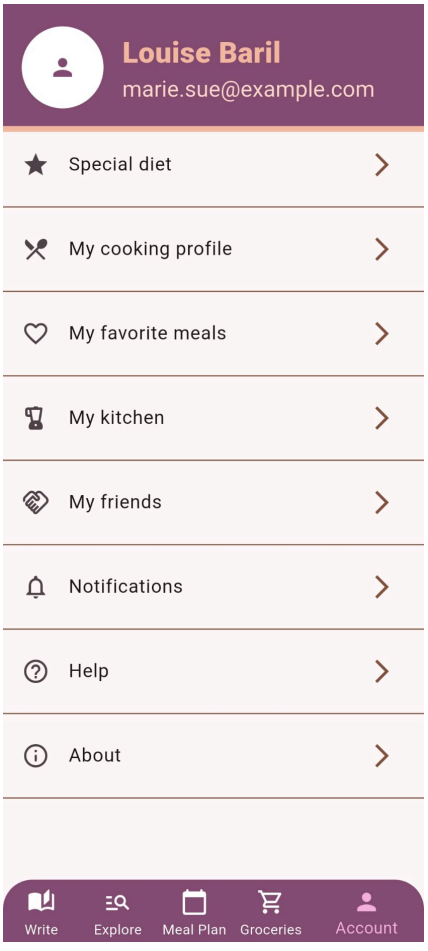


3.2.5 Write Page - Recipe Creation

The Write page presents a scrollable form for creating recipes. Users can add photos via gallery or camera, enter basic information (name, description, servings), select dish types and necessary cookware using ChoiceChips.

The "Add Ingredient" button opens autocomplete search, creating draggable list tiles with quantity inputs. The "Add Step" button opens a form for step type, description, and duration. Steps appear as color-coded, reorderable tiles.

When accessed from a recipe preview's edit button, the page title changes to "Edit Recipe" and pre-populates with existing data.

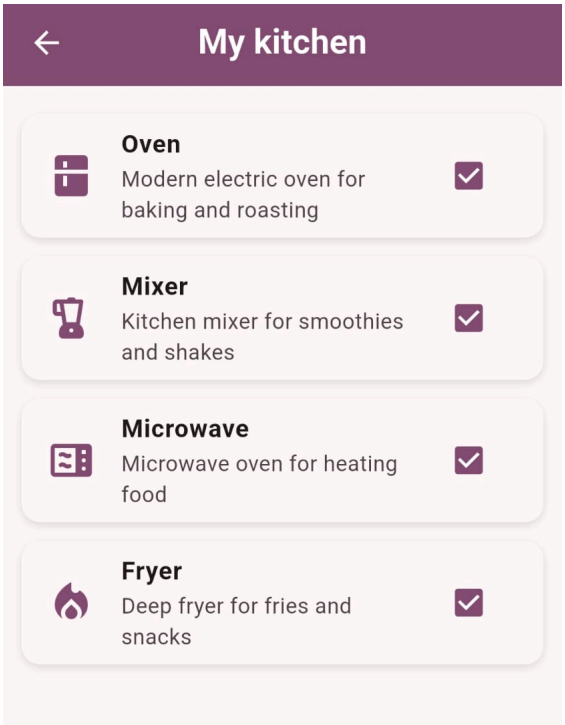


3.2.6 Account Page - Settings

The Account page lists sub-menus including Special Diet, My Cooking Profile, My Favorite Meals, My Kitchen, My Friends, Notifications, Help, and About.

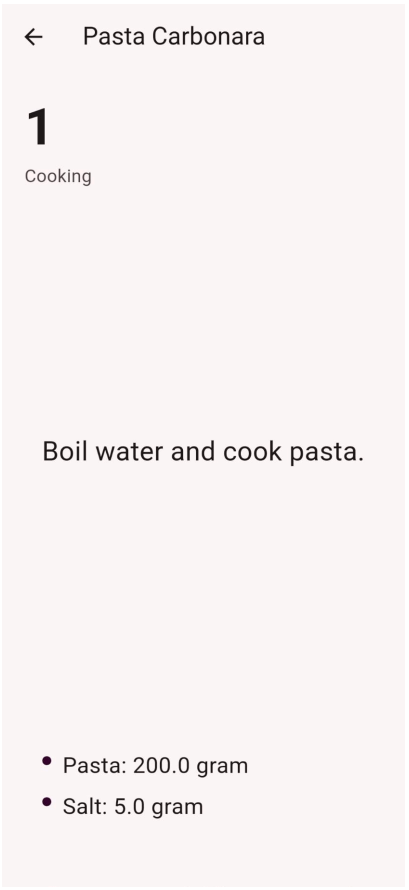
The Special Diet page provides autocomplete search for flagging ingredients as allergens (☠️) or dislikes (🤮), plus four curated diet profiles (Vegetarian, Vegan, Gluten-Free, Halal).

The Cooking Profile page sets the number of portions per meal and cooking patience level (5-level scale from "might not eat" to "love elaborate meals").



3.2.7 Let's Cook! Mode

Cooking mode displays one step at a time in fullscreen. The header shows step number and type. The center displays the description with optional duration and ingredients. Bottom navigation includes previous/next arrows and progress indicator (e.g., "3 / 8"). A back arrow in the top left exits the mode.



3.2.7. Examples of User flow

3.2.7.1 Discovery and Planning

1. User opens the explore page
2. User sets filters (equipment, allergies, time)
3. User browses through the pre-made recipes
4. User clicks on the recipe → Preview (time, price…)
5. User adds the recipe to meal plan (clicks on "Add")
6. User access meal plan (MealPlan page)
7. User can generate shopping list

8. User view list in Groceries
9. User can add other groceries to the list

3.2.7.2 Create Personal Recipe

1. User clicks "+" button (Write page)
2. User enters the name, ingredients needed, steps to prepare it (link to website)
3. User selects equipment and dish type
4. User validates the new recipe
5. The recipe saved locally
6. It is accessible from Explore with "My recipes" badge

4 Requirements

The following list contains the main requirements that the DillyDaily application satisfies, together with a brief explanation of them.

- **R1 :** The system allows users to discover and filter recipes according to multiple criteria including recipe name, ingredient content, energy level (preparation time), dietary restrictions, available equipment, and cooking patience level.
- **R2 :** The system allows users to create and edit custom recipes with optional photos, ingredient lists with quantities, preparation steps, equipment requirements, and external references. All pre-loaded recipes can be edited and personalized.
- **R3 :** The system allows users to organize a weekly meal plan with a 7-day timeline containing 2 meal slots per day. Recipes can be added, moved, and removed using drag-and-drop interactions.
- **R4**: The system automatically generates a grocery list from selected meal plan recipes, calculating ingredient quantities based on the number of servings configured in user preferences and computing the total estimated cost.
- **R5**: The system organizes the grocery list by ingredient categories (Produce, Dairy, Meat & Deli, etc.) and allows users to manually add custom edible and non-edible items. Items can be checked/unchecked for shopping progress tracking.
- **R6**: The system provides a cooking mode that displays recipe preparation steps one at a time with step-by-step navigation, including descriptions, optional durations, and required ingredients for each step.
- **R7**: The system allows users to configure a comprehensive cooking profile including dietary restrictions (with pre-configured profiles and custom ingredient exclusions), available kitchen equipment, cooking patience level, and default number of portions. This profile automatically filters incompatible recipes from search results.
- **R8**: The system stores user preferences, favorite recipes, custom recipes, and meal plans locally on the device, enabling full offline functionality. Recipe and ingredient databases are synchronized with the server only at application startup, ensuring the app operates independently of network connectivity.

5. Implementation, Integration and Testing

5.1 Introduction

This section describes the technologies, frameworks, and methodologies employed to implement, integrate, and test the DillyDaily application. It covers the server infrastructure, client application architecture, external services integration, and both automated and user-acceptance testing approaches.
The S2B (Software to Be) is divided into the following components:

- Application Server
- Client (mobile application)
- External Services
- Testing infrastructure

5.2 Application Server

5.2.1 Distribution Environment

The backend server is deployed on **Render's free tier** cloud platform, providing sufficient uptime and performance for serving static JSON data to the mobile application.

5.2.2 Framework and Programming Language

The server is implemented using **FastAPI**, a modern, high-performance Python web framework. FastAPI was chosen for its:

- Minimal configuration requirements
- Automatic OpenAPI documentation generation
- Native async support
- Excellent performance for serving JSON responses

5.3 Client

5.3.1 Distribution Environment

The client application is distributed as a mobile application targeting **Android** and **iOS** platforms. Due to resource constraints (lack of macOS development environment), the application has been fully tested on Android but not verified on iOS devices, though no platform-specific code was used.

Web Platform Support: Two team members developed using the web interface due to Android development environment limitations. The application is **visualizable on web** with limited functionality:

- Recipe browsing and discovery: Functional
- Cooking mode navigation: Functional
- Local storage (settings, favorites, edition, meal plan): Non-functional

The web version serves as a development and demonstration tool rather than a production target.

5.3.2 Framework and Programming Language

Flutter SDK: Version 3.38.5
Dart Language: Version 3.10.4

Flutter was selected for its **cross-platform development** capabilities, allowing a single codebase to target both Android and iOS without platform-specific implementations. The application uses exclusively **Material Design widgets**, ensuring compatibility across platforms.

5.3.3 Libraries

The application employs the following Dart/Flutter packages:

- **Network Communication** :
  - `http`: HTTP client for server communication
- **Local Storage**:
  - dart:io`: File system operations
  - `path_provider`: Device storage path resolution (uses `getApplicationDocumentsDirectory()`)
- **Image Handling**:
  - `image_picker`: Photo selection from gallery or camera
  - `image_cropper`: Image editing and cropping
- **User Interface**:
  - `flutter_emoji`: Emoji rendering for ingredient representation
  - Native Flutter widgets for drag-and-drop (`LongPressDraggable<String>`)
- **Analytics**:
  - `flutter_clarity`: Microsoft Clarity integration for user behavior tracking

5.3.4 Local Data Storage

DillyDaily implements a JSON-based local storage strategy using the device file system:

Storage Structure:

- **Favorite recipes**: Separate JSON file containing full recipe data
- **Meal plan recipes**: Separate JSON file containing recipes currently in the 7-day plan
- **Custom/edited recipes**: Separate JSON file for user-created and modified recipes
- **User preferences**: Single JSON file containing:
    - Meal plan timeline (7 days × 2 slots with recipe IDs)
    - Dietary restrictions and allergen flags
    - Available kitchen equipment
    - Cooking patience level
    - Default number of portions

Storage Location: Files are persisted in the application's documents directory provided by `path_provider`, ensuring data persistence across app sessions.

Synchronization Strategy: Local data takes precedence over server data for favorites, custom recipes, and meal plans, enabling offline functionality.

5.4 Database Server

DillyDaily does not employ a traditional database management system (DBMS). Instead, recipe and ingredient data are stored as **static JSON files** served by the FastAPI server. This approach is sufficient for the current prototype scale (22 recipes, complete ingredient catalog) and eliminates database configuration overhead.

Future Migration: For production deployment with user authentication and recipe sharing, the data layer would migrate to a **relational database** (PostgreSQL or similar) to support user-generated content at scale.

5.5 External services

5.5.1 Microsoft Clarity Analytics

DillyDaily integrates **Microsoft Clarity** for qualitative user behavior analysis. Clarity provides:

- **Heatmaps**: Visualization of user tap patterns and interaction hotspots
- **Session recordings**: Playback of individual user sessions
- **Active time tracking**: Time spent per session
- **Page visit counts**: Screen popularity metrics
- **Dead tap detection**: Non-interactive elements users attempt to interact with

Privacy Consideration: No personally identifiable information is collected; only anonymous interaction patterns are tracked.

5.5.2 Firebase App Distribution

For user acceptance testing, the application was distributed via **Firebase App Distribution**, which provides:

- Simple link-based sharing to testers
- No requirement for App Store/Play Store publication
- Version management for test builds
- Tester feedback collection

This service enabled the team to conduct user testing without incurring publication costs or app store review delays.

5.6 Testing

To ensure the **reliability** and **correctness** of Dilly Daily, we adopted a two-level testing strategy combining **unit tests** and **widget tests**. This approach allows us to validate both the application logic and the user interface behavior.

5.6.1 Unit tests

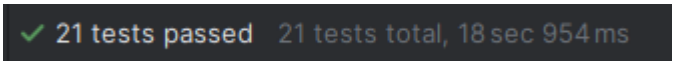
Unit tests are used to verify the correctness of core **business logic** and **data manipulation independently** from the user interface.
The test campaign includes only a limited number of unit tests, as the application does not rely heavily on abstract or complex logic, but rather focuses on user-driven interactions.

5.6.2 Widget tests

Widget tests are used to validate user interactions and UI flows and therefore make up the majority of the test campaign.
These tests aim to simulate **real user scenarios** and ensure that the interface and graphical components behave as intended. This approach is particularly useful to **prevent regressions** when adding new functionalities or modifying existing features.

5.6.3 Results and coverage

We have **21 stable tests** that test a specific logic or user scenario.



This is the coverage of our test campaign per folder:

File	Lines	Line Coverage
− All Files	1659/2927	56.7%
− lib	1659/2927	56.7%
+ data	7/9	77.8%
+ models	316/672	47.0%
− pages	1336/2246	59.5%
+ Explore	334/652	51.2%
+ Write	553/857	64.5%
+ MealPlan	276/455	60.7%
+ Account	173/282	61.3%

5.7 User Acceptance Testing

To complement the automated testing strategy, the team conducted user acceptance testing to validate that the application meets user expectations and provides an intuitive experience in real-world scenarios.

5.7.1 Test Participant Recruitment:

- 8 participants recruited from family and friends
- Participants had previously heard about the project
- Mix of cooking experience levels and technical proficiency

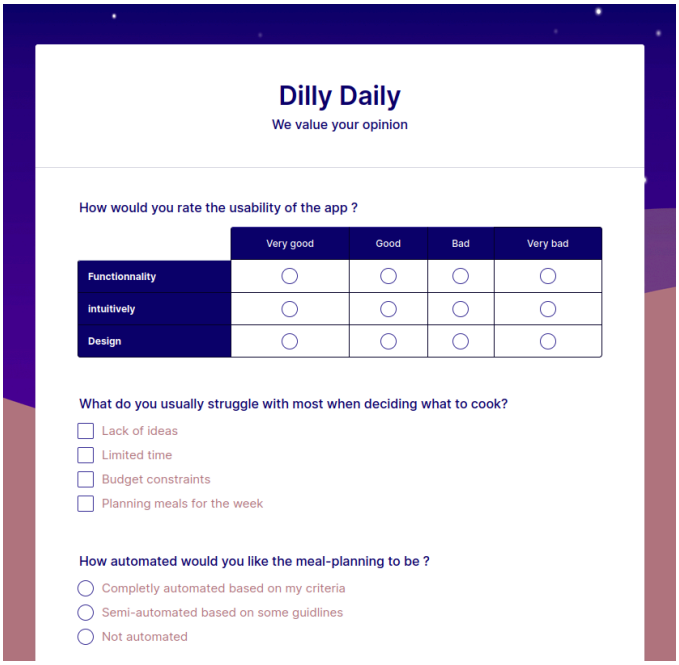
5.7.2 Testing Objectives:

- Gather general usability feedback
- Assess interest in proposed features
- Identify bugs and missing functionalities
- Evaluate intuitiveness of core interactions

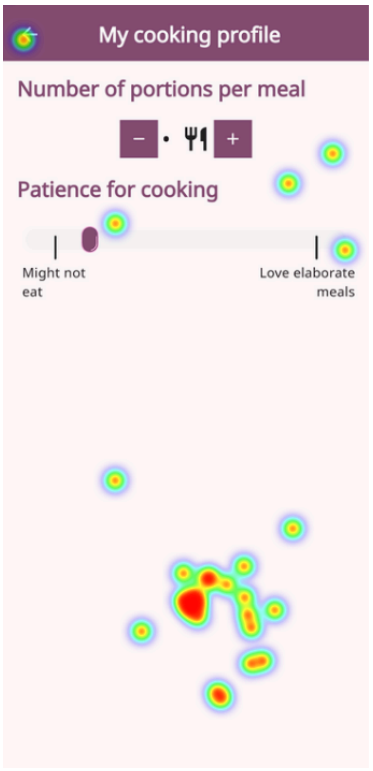
5.7.3 Methodology:

The testing campaign combined three data collection approaches:

1. **Structured Questionnaire**: Participants completed a survey with the questions focused on usability
2. **Microsoft Clarity Analytics**: Heatmaps and session recordings revealed where users experienced confusion or hesitation



3. **Direct Observation**: Team members observed some testing sessions to capture immediate reactions

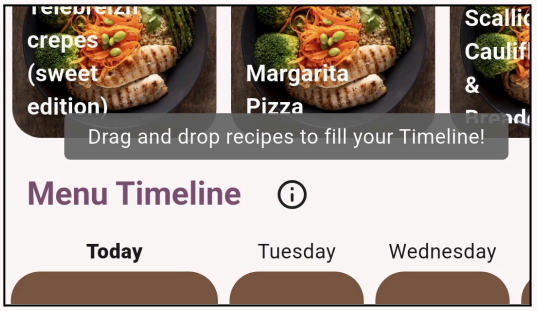


Key Findings:

- **Overall Enthusiasm**: Most testers were unfamiliar with meal planning applications and appreciated the "tailored cookbook" concept
- **Intuitiveness Gap**: The application was less intuitive than the team anticipated, particularly for:
  - Drag-and-drop meal planning (addressed with tooltip addition)
    - Navigation between meal plan and grocery list
    - Understanding automatic filtering behavior
- **Feature Requests**: Strong interest in **recipe sharing** functionality, validating the team's roadmap for future development with user authentication
- **Recipe Selection**: Positive reception of the curated recipe database, though users requested more variety in certain categories

Design Improvements Implemented: Based on user feedback and Clarity heatmaps, the team added:

- Info button (ⓘ) with tooltip explaining drag-and-drop mechanics on Meal Plan page
- Clarification of automatic filter toggles in settings



Screenshot of Meal Plan info tooltip

Integration of Quantitative and Qualitative Data: Clarity's heatmaps complemented questionnaire responses by revealing specific UI areas causing confusion. For example, session recordings showed users repeatedly attempting to tap empty meal slots in the timeline rather than dragging recipes, confirming the need for the info button (ⓘ) tooltip explaining the drag-and-drop interaction pattern.

5.8 Deployment

5.7.1 Distribution Strategy

Due to the **cost of app store publication** (€25 for Google Play, $99/year for App Store), the team opted not to publish DillyDaily on official stores for this academic project.

Alternative Distribution:

- **Firebase App Distribution**: Provided shareable links for test builds
- **APK direct distribution**: Android users could install via APK file
- **Web preview**: Demonstration version accessible via browser (limited functionality)

6 Final Notes

6.1 Effort Spent

Student	Backend	Mobile app	Testing	Documentation	Presentation	Total
Lyla Demange	2h	130h	0h	0h	1h	**132h**
Marion Henriot	0h	110h	0h	15h	2h	**127h**
Nils Mittelhockamp	0h	60h	50h	15h	4h	**129h**