

MYGLOOCO (V.1.0)

Software Authors

1. Joseph Muema - Frontend / Backend Developer (Group Lead)
2. Jayden Mathenge - Backend Developer
3. Joseph Njenga - Frontend Developer / UI Designer
4. Kelly Kasina - Database Engineer
5. Terry Mutheu - Frontend Developer / UI Designer

i. PROBLEM ANALYSIS AND IDEA VALIDATION

Problem Statement

The Rising Prevalence of Diabetes

Diabetes is a chronic disease that affects millions of people worldwide. Its prevalence is on the rise, and it is estimated that by 2045, nearly 700 million people will be living with diabetes. Despite the availability of treatments, many individuals lack the necessary knowledge or tools to manage their condition effectively.

Currently, individuals living with diabetes are facing a myriad of challenges, most of which are related to their current help systems. Some rely on manual record keeping for their blood sugar level which has proven to be cumbersome, and others have little to no information on dietary planning, as well as exercise planning, which hinders their overall quality of life.

Pain points

1. **Cumbersome:** manual recording/tracking of blood sugar levels is often cumbersome, posing a challenge to those aiming to analyze/take notes on their blood sugar levels
2. **Inefficiency:** manual tracking poses a great challenge to overall efficiency in analysis of one's blood sugar records. Without dedicated blood sugar log books, it becomes difficult to keep track of one's blood sugar level data which makes it harder to analyze their data/whether they are achieving their goals. It also makes the doctor's area of interpretation more difficult as data may not be organized in an understandable format.
3. **Absence of dietary data/information:** Whilst Diabetics do not require a specialized diet, healthy eating is recommended, and that requires proper knowledge of nutritional information. However, deciding on a proper dietary plan is often difficult while on their own, as they have to factor in the aforementioned nutritional information such as the glycemic index (a measure used to determine how much a food can affect your blood sugar levels), and their fat/Calorie consumption, as they are at greater risk of developing heart diseases.
4. **Lack of a proper exercise plan:** Choosing an exercise plan tuned to one's goals is often difficult, which could create difficulty for a patient aiming to reach/maintain their targeted blood sugar level goals

5. **Medication Management:** Diabetes often requires multiple medications, including insulin injections or oral medications. Managing these medications can be complex, especially if there are multiple doses or types of medication. A mobile app could help users track their medication schedules, set reminders for doses, and even provide information about potential drug interactions.
6. **Community Support:** Living with diabetes can be emotionally challenging, and having a supportive community can make a significant difference. A mobile app could facilitate connections between users, allowing them to share experiences, tips, and encouragement. This social support aspect can be particularly beneficial for individuals who may feel isolated in managing their condition.

Significance of pain points

Current systems targeted at helping diabetics live a healthy and normal life are mostly cumbersome/inefficient, which poses a great challenge to their overall quality of life. Addressing these pain points should improve their overall quality of life, easing the overall burden of an unwanted disease.

The Need for a Personalized Approach

Given these challenges, there is a clear need for a simple, efficient, and personalized system to aid individuals in managing their diabetes.

Such a system should provide users with the knowledge they need to understand their condition, as well as tools for tracking their blood sugar levels, medications, diet, and physical activity. It should also offer personalized suggestions and recommendations based on the user's unique needs and circumstances.

The Solution

The Myglooco app aims to provide users with an easy and efficient way to track their blood sugar readings, tuned to their desired scales (mmol/L (millimoles per liter) or mg/dL (milligrams per deciliter)), with the ability to set and observe their targeted blood sugar level in comparison to their current levels, as well as provide users with visualization of data through charts from recorded values. It respects and aims to provide for individuals with all currently documented types of diabetes and aims to provide dietary recommendations and information, alongside exercise recommendations to the individuals using the app, set to their desired specifications. It also contains information to further help individuals with diabetes

learn more about diabetes, and provides resources to help with testing and medication. For analytical purposes, it may be used in future to generate reports which can be passed onto doctors for further recommendations and assessments.

PROJECT ABSTRACT

Effective diabetes management is critical for maintaining long-term health and preventing complications, yet it can be overwhelming for individuals to monitor and manage the various aspects of their condition. Our personalized diabetes management mobile application is designed to simplify this process by integrating a suite of tools that address the core needs of diabetes care in one platform.

Key features of the app include comprehensive blood sugar logging and visualization, allowing users to track their glucose levels over time and identify trends that can inform their daily management decisions. Diet planning and meal logging features enable users to monitor their nutritional intake, with options to log meals, track carbohydrate counts, and receive personalized dietary recommendations.

The app also includes a medication management system with reminders, ensuring users take their prescribed medications on time and maintain their treatment regimen. Physical activity monitoring allows users to log exercise routines, helping them to balance activity levels with their dietary and medication plans. Additionally, the app offers a wealth of educational resources, providing users with information on diabetes management, healthy lifestyle choices, and strategies for preventing complications.

One of the app's standout features is its ability to generate detailed reports that compile blood sugar readings, diet, exercise, and medication data. These reports can be easily shared with healthcare providers, offering a comprehensive overview of the user's diabetes management and facilitating more informed medical consultations.

By bringing together these essential tools, our mobile application empowers users to take control of their diabetes care, making it easier to manage their condition on a day-to-day basis. The app aims to improve adherence to treatment plans, enhance user knowledge, and ultimately lead to better health outcomes. This holistic approach to diabetes management not only supports users in their current health journey but also lays the foundation for a healthier future.

ii. ANALYSIS

Analysis of Pre-Existing Solutions

- BG Monitor (Free, Android)
- One Drop (Free, iOS app)
- mySugr (Free, web plus iOS and/or Android app)

Market Analysis

The Myglooco app is targeted at individuals with Diabetes, as well as prediabetics, as both have to take immense care of their blood sugar, and the reports generated from the app can prove very useful in medical analysis.

It can also be used by family, friends and caregivers with the intention of helping family affected (for example, individuals too old/too weak to physically do collection themselves can receive aid from family using the app)

Besides the aforementioned parties, the app can see use by doctors, in form of health assessment through reports, or redirection of users to hospitals in case of emergency/no change in health assessment, or when one is unsure of whether they have diabetes or not (in which case it is heavily recommended to visit a doctor)

Health and Wellness Enthusiasts could also use the app, even if they don't have diabetes themselves. They could use it to track their diet, monitor their physical activity levels, and access educational resources on nutrition and fitness.

Market Research Findings

This section summarizes the key findings from our market research activities, including customer interviews, surveys, competitor analysis, and industry trends.

1. Customer Interviews:

Struggles:

Participants expressed frustration with the burden of self-monitoring and keeping track of various data points.

Many felt their current treatment plans were static and didn't adapt to their daily lives.

Limited communication with healthcare providers made it difficult to adjust medications or address concerns promptly.

Alternative Solutions:

Some participants relied on paper logs or spreadsheets to track blood sugar and other metrics.

Others used basic mobile apps for blood sugar logging but lacked advanced features such as dietary logging.

A few mentioned participating in online diabetes forums for support and information sharing.

Valued Features:

Participants emphasized the importance of user-friendly interfaces and clear data visualization tools.

They expressed interest in personalized insights and recommendations based on their specific data.

Integration with wearables and seamless data collection were highly desirable features.

Secure data storage and privacy were critical considerations for all participants.

2. Surveys:

The survey results supported the interview findings, highlighting the widespread desire for a more integrated and data-driven approach to diabetes management.

A significant portion of respondents indicated a willingness to pay for a subscription-based service offering personalized health assessments and actionable insights.

Survey data also helped quantify the target market size and identify key demographics most interested in the solution.

[Survey results are linked here](#), and the actual survey, [here](#)

3. Competitor Analysis:

Several existing competitors offer mobile apps for diabetes management.

Strengths of competitors include established user bases, brand recognition, and basic features like blood sugar logging and medication reminders.

Weaknesses identified were limited data analysis capabilities, lack of personalization, and poor user interface design in some cases.

Pricing models varied from freemium with limited features to premium subscriptions with more advanced functionalities.

4. Industry Trends:

The market for digital health solutions is rapidly growing, driven by increasing smartphone penetration and user adoption of wearable health trackers.

There is a growing emphasis on preventative care and personalized medicine approaches in chronic disease management.

Regulatory considerations related to data privacy and security need to be addressed to ensure user trust.

Emerging technologies like artificial intelligence have the potential to further personalize and enhance diabetes management tools.

Overall Insights:

Our market research reveals a clear demand for a more comprehensive and personalized approach to diabetes management. Patients are looking for solutions that empower them to actively manage their condition and collaborate effectively with their healthcare providers. By addressing the identified gaps in the market and leveraging valuable industry trends, our proposed software has the potential to significantly improve diabetes management for a large and growing patient population.

Refinement

Value Proposition:

Empower patients with diabetes to take control of their health through personalized digital assessments and actionable insights, leading to improved management and collaboration with healthcare providers.

This statement highlights the key benefits for both patients (empowerment, control, improved management) and healthcare providers (collaboration).

Minimum Viable Product (MVP):

Core Features:

- User-friendly interface for self-monitoring blood sugar levels, dietary intake, and physical activity.
- Food search and review system.
- Exercise logging and recommendation systems.
- Educational facilities related to living with diabetes.
- Secure data storage and privacy compliant with relevant regulations.

Focus:

- Localizing the app's features to make it more relatable to users in our region. (Kenya, EA)
- Validating user interest and understanding their needs through early feedback.
- Testing the core functionalities and data analysis capabilities.
- Gathering user feedback on the usability and value proposition of the MVP.

Business Model:

After careful analysis, the following business model was decided upon:

1. Freemium Model:

Offer a free tier with basic features like blood sugar tracking and limited data insights.

Provide a premium subscription tier with advanced functionalities such as:

- Deeper data analysis with personalized recommendations
- Integration with additional wearables and health apps
- Live chat support with healthcare professionals

This model allows users to try the core service for free and upgrade for more advanced features, generating revenue through subscriptions. Alternatively, the following options could be considered:

2. Partnerships: Collaborate with healthcare providers, diabetes educators, or pharmaceutical companies who can offer your service to their patients for a fee.

3. Data Insight: With proper consent and anonymized data, explore offering aggregated data insights to research institutions or pharmaceutical companies for a fee (ensure user privacy is protected).

REQUIREMENTS ANALYSIS**Functional requirements**

- The system provides new users with a brief tutorial on their first-time launch, on how to use the app.

- The system accepts/collects the user's data (only for analysis purposes), such as, Diabetes type, insulin therapy method, target glucose ranges, etc. during initial set-up
- The system provides its users with a visual representation/summary of vital information, such as graphs of blood sugar readings against time.
- The system should allow/prompts users to enter information related to their health (as well as for analysis purposes), such as glucose levels, medication used, details of any exercise/activity or meal records (for calorie estimations).
- The system, if prompted, provides users with a food search system, detailing the food's data (Calories, Carbs, fat, etc.). The users should also be able to record the searched food as a meal if they desire to do so.
- The system keeps logs of user entered data (such as food logs or water/drink intake), be it for analysis or report generation.
- The system stores user's private data in a safe and secure environment, ensuring that it is only accessible to the users themselves.
- The system provides its users with educational support on diabetes and any terms related to it.

Non-Functional requirements

- **Usability** – the application is to offer a user-friendly and intuitive interface for smoother and better interactions its users.
- **Security** – the user's private data is kept safe through robust security measures beginning with mechanisms such as password authentication.
- **Performance** – the application is to work smoothly and efficiently without lag.
- **Maintainability** – the application is designed in a way that makes it easy to update and maintain. This includes using coding standards and having a clear documentation.
- **Reliability** – the application is reliable and not prone to frequent crashes or errors.

iii. **DESIGN AND IMPLEMENTATION**

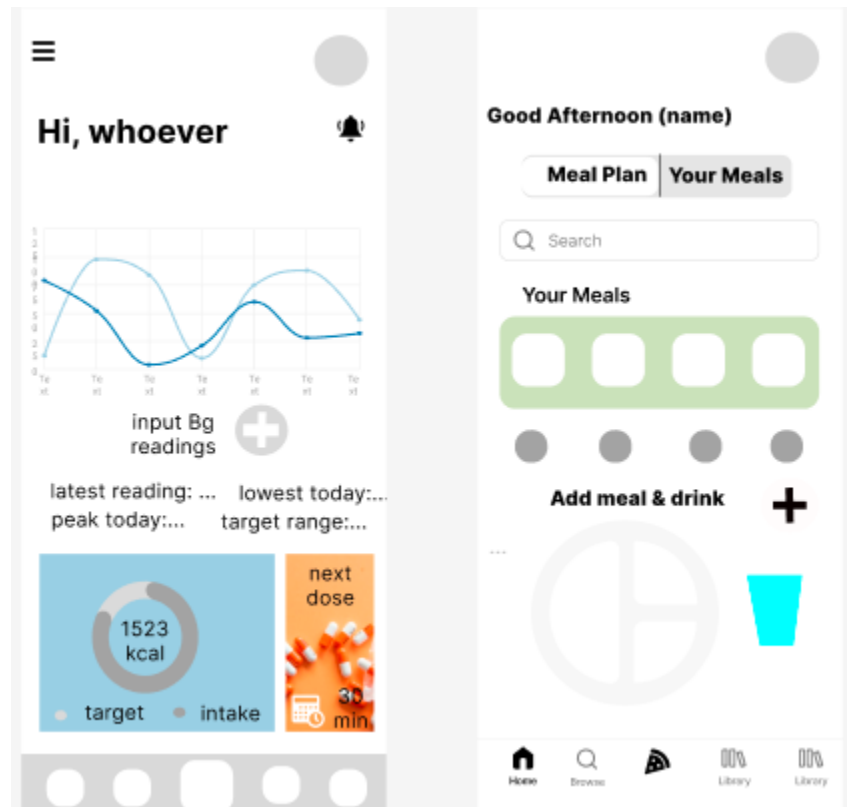
DESIGN SYSTEMS

Software used

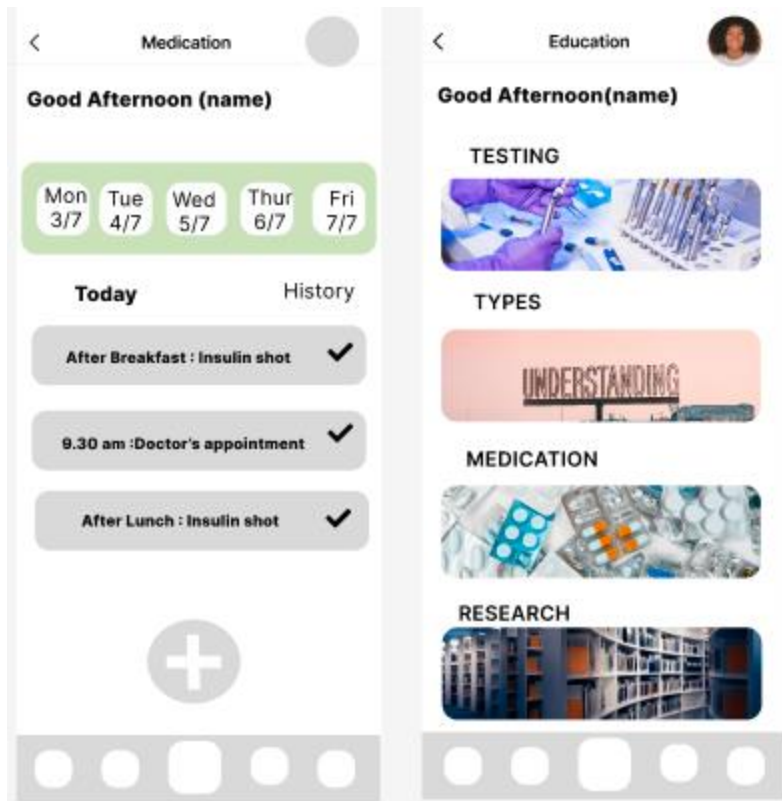
- Figma (for conceptual and initial UI design)

Proof of concept

During the initial design phases, Figma was used to provide a rough overview of the system. Early design phases are showcased in the pictures below:



- i. Initial designs of landing page and meals page, showing both visualization tools and navigation bars.



ii. Early designs for Medication and education modules.

The Figma files can be accessed [here](#)(View only).

IMPLEMENTATION SYSTEMS

As Figma was initially used as a test design tool, the finalized designs varied from the initial designs, which were only intended for visualization and discussion purposes. Therefore, after the description of the technology used during implementation, a Feature showcase of each module will be provided, accompanied by one or more images of the module.

1. Frontend (Mobile Application)

Platform: Utilizing Flutterflow for cross-platform development, ensuring a consistent user experience across iOS and Android devices.

Components:

User Interface: Screens and widgets for inputting blood sugar levels, setting medication schedules, accessing dietary details, and educational resources.

Visualization Tools: Graphs and charts to visualize blood sugar trends, medication adherence, and goal progress.

Notifications: Push notifications for medication reminders, blood sugar checks,

and other alerts.

Educational Resources: Access to articles, tips, and tutorials on diabetes management.

2. Backend Services

Platform: Utilize Dart and the Flutter framework for building robust backend services.

Components:

Authentication: Secure user authentication and session management using industry-standard practices.

Data Processing: Logic for processing user data, generating personalized reports.

3. Database

Platform: Firestore database for storing structured data efficiently.

Components:

User Profiles: Store user information including medical history, preferences, and settings.

Health Data: Record blood sugar levels, medication logs, dietary inputs, and physical activity data.

Analytics: Store processed data for trend analysis and generating reports.

4. Third-Party Integrations

Notification Services: Utilize Flutter plugins (Awesome_notifications) for push notifications, ensuring timely reminders for medications, check-ins, and other important tasks.

5. Security and Compliance

Encryption: Implement data encryption both at rest and in transit to safeguard sensitive health information.

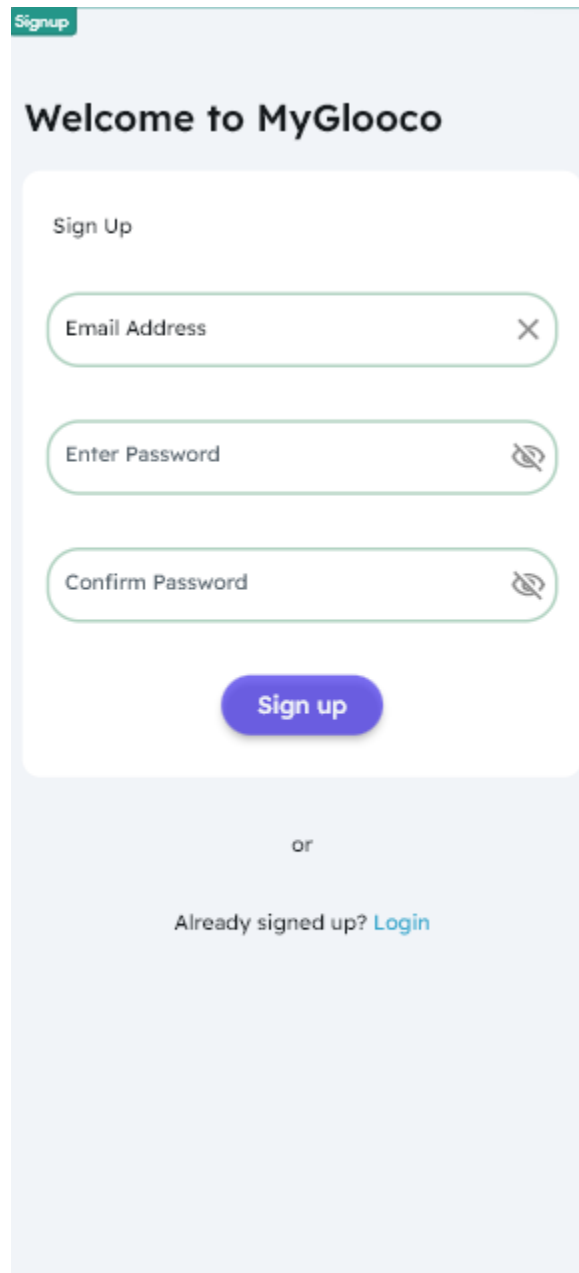
Compliance: Adhere to health data regulations such as HIPAA and GDPR to ensure user data privacy and security.

FEATURE SHOWCASE

This application takes a holistic approach to diabetes management, providing users with a suite of features designed to track progress, gain insights, and make informed decisions.

0. Start up

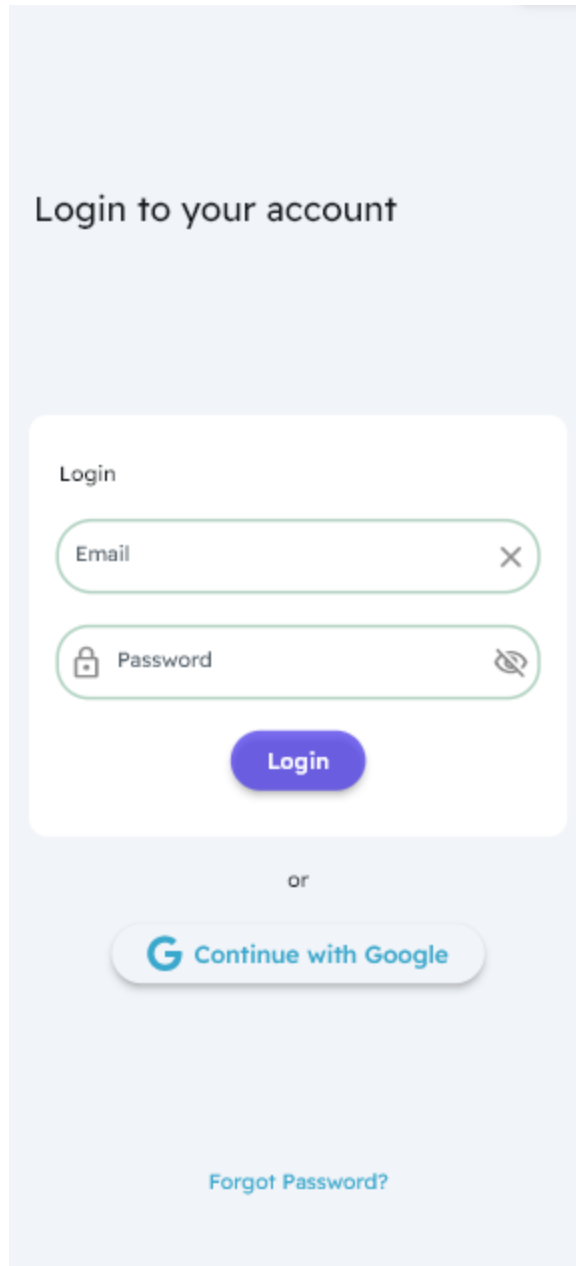
- **Sign up:** for first time users. Registers user with a new account.



The image shows a mobile app interface for signing up. At the top, there is a teal tab labeled 'Signup'. Below it, the heading 'Welcome to MyGlooco' is displayed. The main content area is a white rounded rectangle containing the text 'Sign Up'. There are three input fields: 'Email Address' with a clear icon (X), 'Enter Password' with a toggle icon (eye with slash), and 'Confirm Password' with a toggle icon (eye with slash). A purple 'Sign up' button is positioned below the fields. At the bottom of the white area, the text 'or' is centered. Below the white area, the text 'Already signed up? [Login](#)' is displayed.

(i) Sign up page

- **Login:** for returning users that already have an account.



(ii) Login page

Sample source code

Signup widget model

```
import '/flutter_flow/flutter_flow_util.dart';
import 'signup_widget.dart' show SignupWidget;
import 'package:flutter/material.dart';
```

```

class SignupModel extends FlutterFlowModel<SignupWidget> {
  /// State fields for stateful widgets in this page.

  final formKey = GlobalKey<FormState>();
  // State field(s) for EmailAddress widget.
  FocusNode? emailAddressFocusNode;
  TextEditingController? emailAddressTextController;
  String? Function(BuildContext, String?)? emailAddressTextControllerValidator;
  String? _emailAddressTextControllerValidator(
    BuildContext context, String? val) {
    if (val == null || val.isEmpty) {
      return 'Field is required';
    }

    if (!RegExp(kTextValidatorEmailRegex).hasMatch(val)) {
      return 'Has to be a valid email address.';
    }
    return null;
  }

  // State field(s) for EnterPassword widget.
  FocusNode? enterPasswordFocusNode;
  TextEditingController? enterPasswordTextController;
  late bool enterPasswordVisibility;
  String? Function(BuildContext, String?)? enterPasswordTextControllerValidator;
  String? _enterPasswordTextControllerValidator(
    BuildContext context, String? val) {
    if (val == null || val.isEmpty) {
      return 'Field is required';
    }

    return null;
  }

  // State field(s) for ConfirmPassword widget.
  FocusNode? confirmPasswordFocusNode;
  TextEditingController? confirmPasswordTextController;
  late bool confirmPasswordVisibility;
  String? Function(BuildContext, String?)?
    confirmPasswordTextControllerValidator;
  String? _confirmPasswordTextControllerValidator(
    BuildContext context, String? val) {
    if (val == null || val.isEmpty) {
      return 'Field is required';
    }
  }
}

```

```

        return null;
    }

    @override
    void initState(BuildContext context) {
        emailAddressTextControllerValidator = _emailAddressTextControllerValidator;
        enterPasswordVisibility = false;
        enterPasswordTextControllerValidator =
            _enterPasswordTextControllerValidator;
        confirmPasswordVisibility = false;
        confirmPasswordTextControllerValidator =
            _confirmPasswordTextControllerValidator;
    }

    @override
    void dispose() {
        emailAddressFocusNode?.dispose();
        emailAddressTextController?.dispose();

        enterPasswordFocusNode?.dispose();
        enterPasswordTextController?.dispose();

        confirmPasswordFocusNode?.dispose();
        confirmPasswordTextController?.dispose();
    }
}

```

Login page widget model

```

import '/flutter_flow/flutter_flow_util.dart';
import 'login_widget.dart' show LoginWidget;
import 'package:flutter/material.dart';

class LoginModel extends FlutterFlowModel<LoginWidget> {
    /// State fields for stateful widgets in this page.

    final formKey = GlobalKey<FormState>();
    // State field(s) for Login-Email widget.
    FocusNode? loginEmailFocusNode;
    TextEditingController? loginEmailTextController;
    String? Function(BuildContext, String?)? loginEmailTextControllerValidator;
    String? _loginEmailTextControllerValidator(

```



```

        BuildContext context, String? val) {
    if (val == null || val.isEmpty) {
        return 'Field is required';
    }

    if (!RegExp(kTextValidatorEmailRegex).hasMatch(val)) {
        return 'Has to be a valid email address.';
    }
    return null;
}

// State field(s) for Login-Password widget.
FocusNode? loginPasswordFocusNode;
TextEditingController? loginPasswordTextController;
late bool loginPasswordVisibility;
String? Function(BuildContext, String?)? loginPasswordTextControllerValidator;
String? _loginPasswordTextControllerValidator(
    BuildContext context, String? val) {
    if (val == null || val.isEmpty) {
        return 'Field is required';
    }

    return null;
}

@override
void initState(BuildContext context) {
    loginEmailTextControllerValidator = _loginEmailTextControllerValidator;
    loginPasswordVisibility = false;
    loginPasswordTextControllerValidator =
        _loginPasswordTextControllerValidator;
}

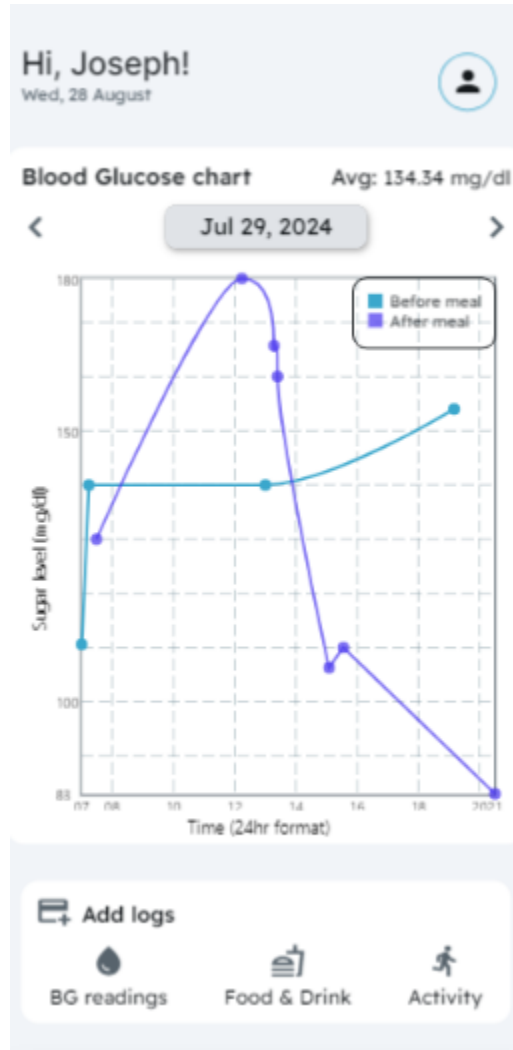
@override
void dispose() {
    loginEmailFocusNode?.dispose();
    loginEmailTextController?.dispose();

    loginPasswordFocusNode?.dispose();
    loginPasswordTextController?.dispose();
}
}

```

1. Core Tracking

- **Visualization Tools:** Utilize charts and graphs to visualize trends in blood sugar, allowing for easy identification of patterns and progress throughout the day.



(iii) Homepage (Blood sugar chart)

- **Quick access sheets:** there are three main sheets in the homepage for quick information logging i.e;
 - **Blood Sugar logging:** Log blood sugar readings with timestamps for a clear picture of glucose levels throughout the day.

Hi, Joseph!
Wed, 28 August

Blood Glucose chart Avg: 0 mg/dl

< Aug 28, 2024 >

Before meal
After meal

Sugar level (mg/dl)

Log blood sugar

After breakfast ▾ 28/8 9:06 AM

Input BG reading
95.8 mg/dl ▾

Done

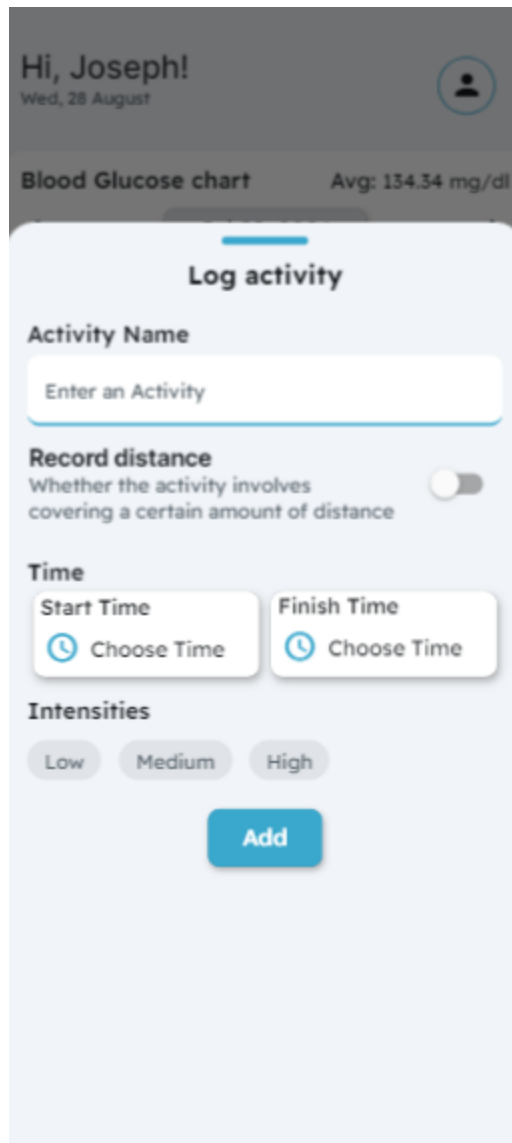
(iv) *Blood sugar logging sheet*

- ***Diet logging:*** for recording recently consumed meals.



(v) *Diet logging sheet*

- **Physical activity logging:** for recording physical activities/exercises performed in the day.



(vi) Activity logging

Sample source code

Home page widget model

```
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'home_widget.dart' show HomeWidget;
import 'package:flutter/material.dart';

class HomeModel extends FlutterFlowModel<HomeWidget> {
  /// Local state fields for this page.
```

```

DateTime? currentChartDate;

double? bloodSugarAvg = 0.0;

/// State fields for stateful widgets in this page.

// Stores action output result for [Firestore Query - Query a collection]
action in Home widget.
List<BGreadingsRecord>? todayReadings;
// Stores action output result for [Firestore Query - Query a collection]
action in Home widget.
List<IndividualRemindersRecord>? unmarkedReminders;
// Stores action output result for [Firestore Query - Query a collection]
action in Icon widget.
List<BGreadingsRecord>? prevDayReadings;
DateTime? datePicked;
// Stores action output result for [Firestore Query - Query a collection]
action in Icon widget.
List<BGreadingsRecord>? nextDayReadings;

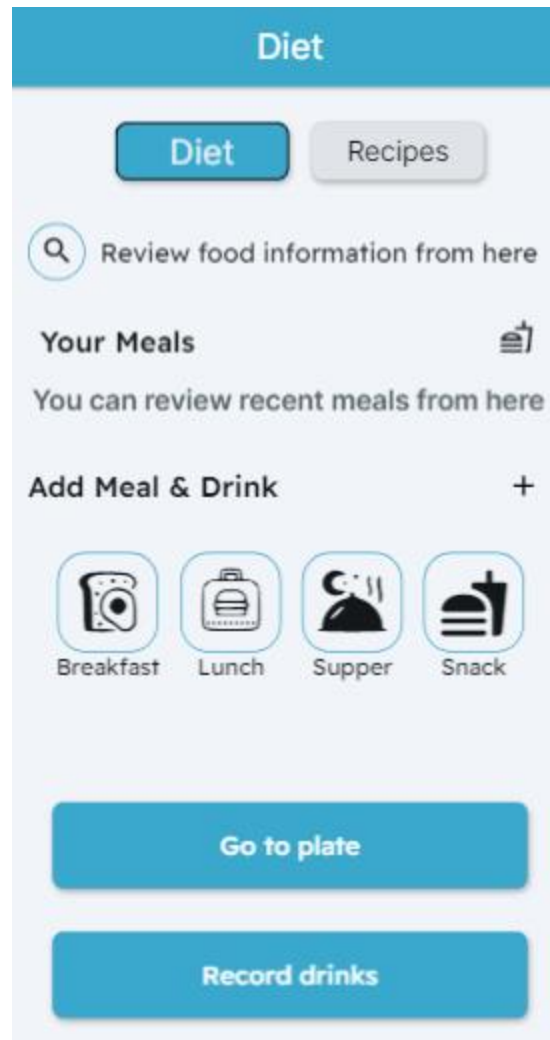
@override
void initState(BuildContext context) {}

@override
void dispose() {}
}

```

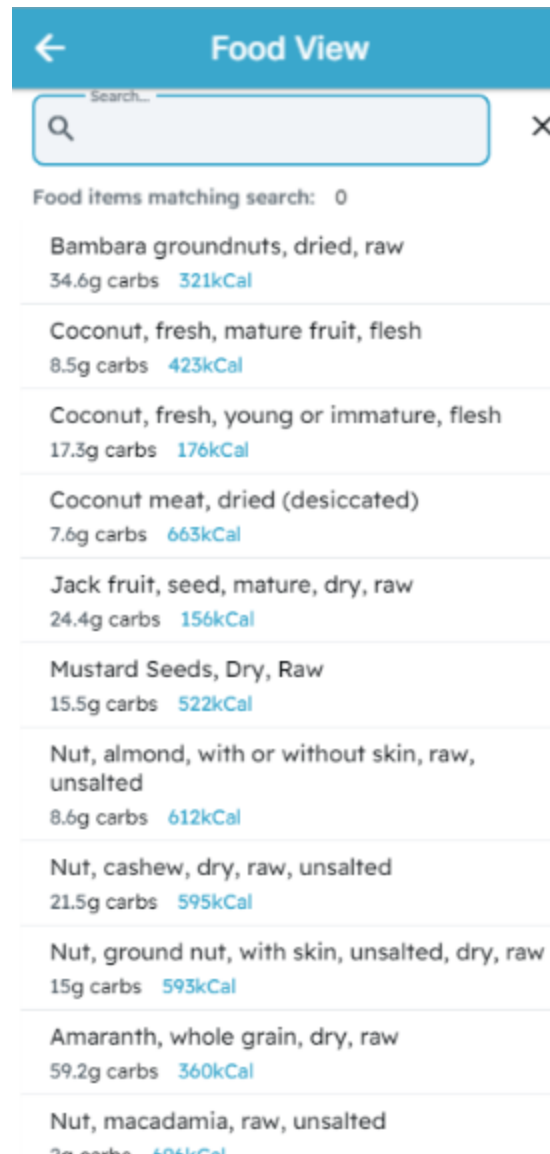
2. Dietary section

- **Main page:** contains various links to different diet subsections.



(vii) Main diet page

- **General food search:** Review different food items and their information, ranging from caloric to protein/carbohydrate levels. (view only)



(viii) General search

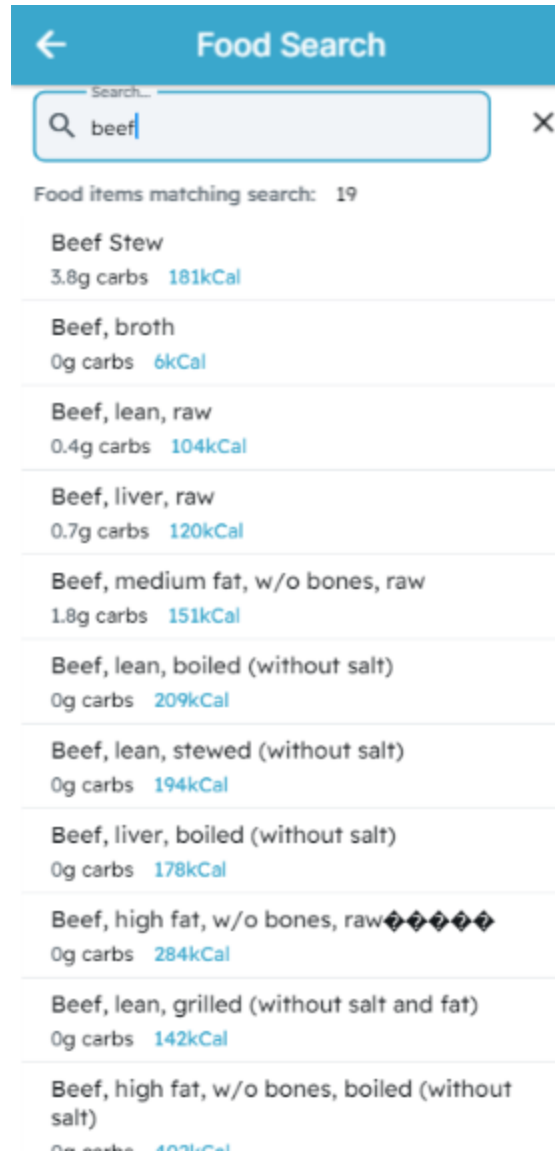
- **Dietary Management**
 - **Plate Method:** Utilizes a plate method food builder to create balanced meals with clear portion guidance and calorie estimation.



(ix) Plate section (empty)



(x) Carb plate search
(when carb sector is pressed on the plate)



(xii) Protein plate search
(when protein sector is pressed on the plate)

←

Plate

+

Default estimations are per 100g of edible portion on fresh weight basis. (You can change the estimate mass in each food item's mass textfield)

To search for uncategorized food items, click (+) button in the bottom-right corner of the plate.

Summary

Total Kcal: 443.40 Kcal

Refined Maize Flour Ugali

130 g

×

192.4 kCal
35.49 g Carbs

4.42 g Protein
2.47 g Fats

Stir-fried Cabbage

70 g

×

70 kCal
2.17 g Carbs

0.84 g Protein
6.02 g Fats

Beef Stew

100 g

×

181 kCal
3.8 g Carbs

17.7 g Protein
10.4 g Fats

Record Meal

(xiii) *Filled plate*
(after individual searching)

← Plate

×

Confirm selection


Supper

Total Kcal: 443.40 Kcal

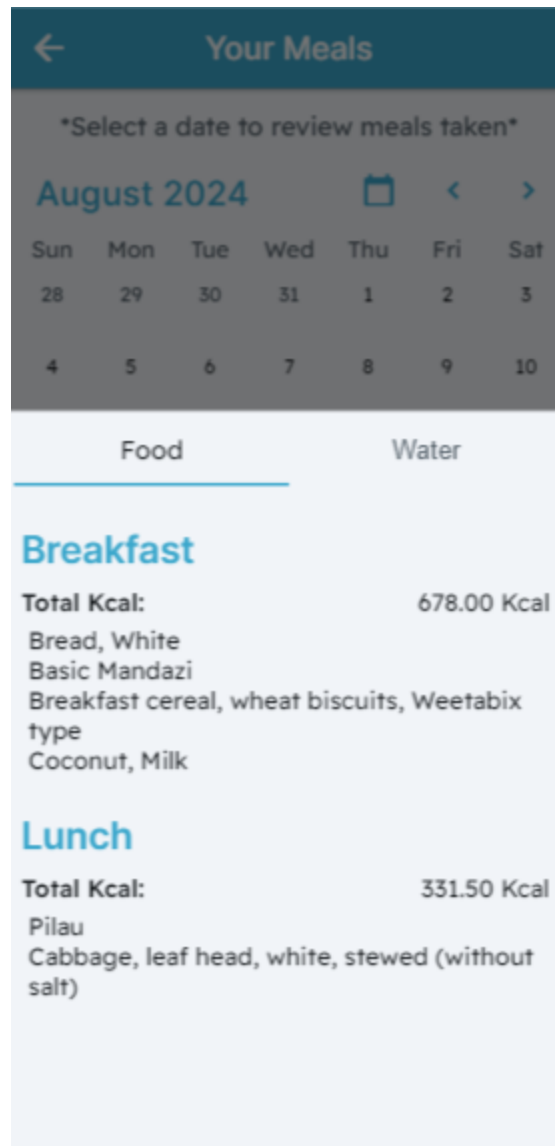
Refined Maize Flour Ugali

Stir-fried Cabbage

Beef Stew

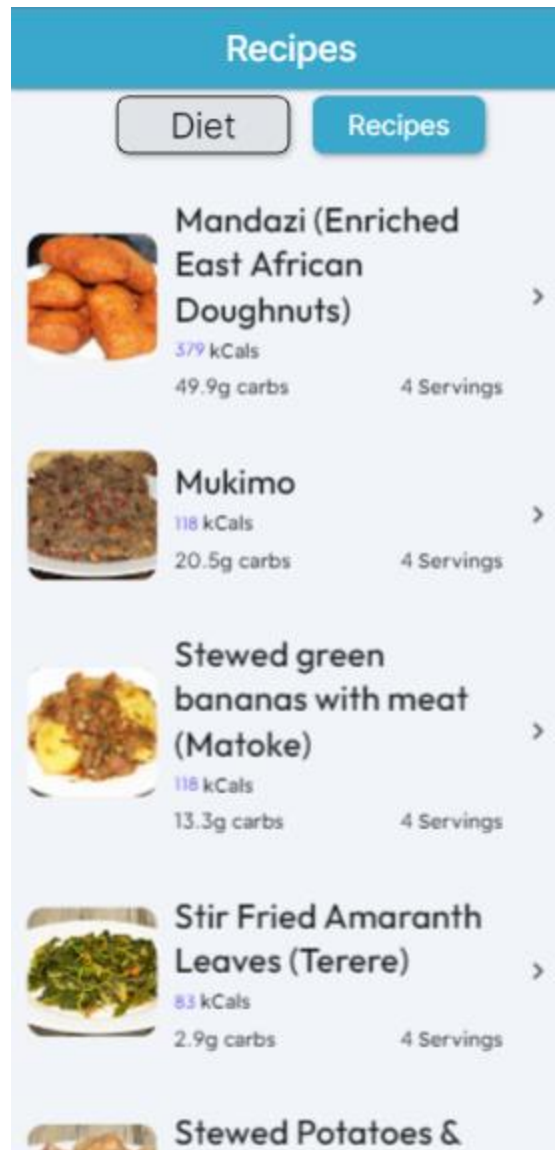
 Record as meal

(xiii) Confirm meal selection and meal type before recording





View meal history for past dates

- **Recipe selection:** List of pre-made local recipes with detailed instructions for preparation.





(xiv) Sample list of available recipes


 Recipe Details




Recipe Details

 118 kCal

 13.3g carbs

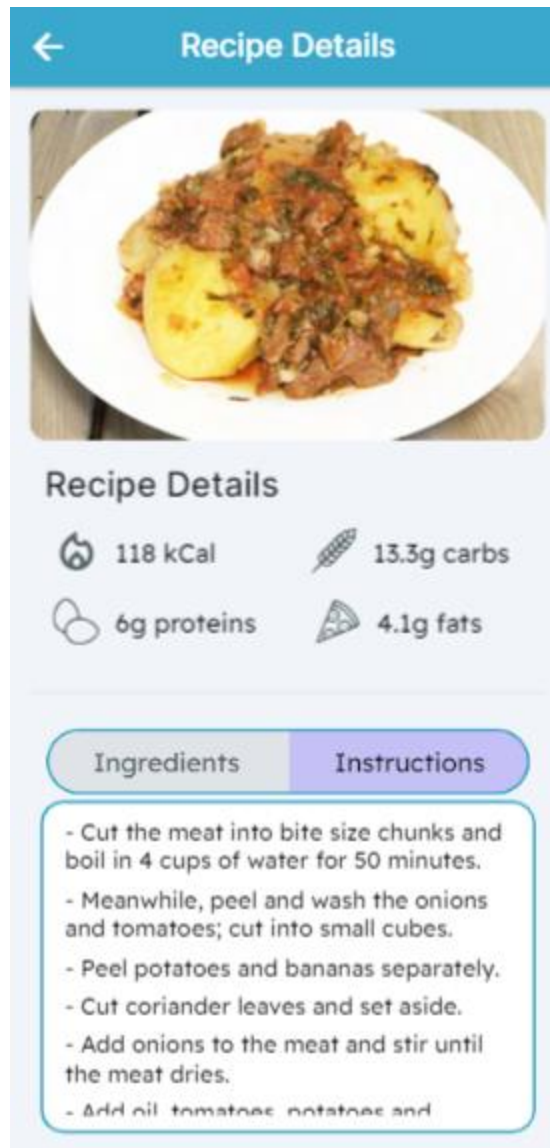
 6g proteins

 4.1g fats

IngredientsInstructions

- 456 g beef, medium fat
- 1.5 kg green bananas, unpeeled, raw
- 1 onion, unpeeled, red skinned, raw (184 g)
- 3 tomatoes, red, ripe (329 g)
- 106 g coriander leaves
- 8 potatoes, unpeeled, Irish, white (1.1 kg)

(xv) List of ingredients for a sample recipe



(xvi) Preparation instructions for a sample recipe

Sample source code

Main diet page widget model

```
import '/flutter_flow/flutter_flow_util.dart';
import 'diet_home_widget.dart' show DietHomeWidget;
import 'package:flutter/material.dart';

class DietHomeModel extends FlutterFlowModel<DietHomeWidget> {
  @override
  void initState(BuildContext context) {}
}
```

```
@override
void dispose() {}
}
```

Plate page widget model

```
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'plate_widget.dart' show PlateWidget;
import 'package:flutter/material.dart';

class PlateModel extends FlutterFlowModel<PlateWidget> {
  /// Local state fields for this page.

  List<FoodRecord> updatedFoodList = [];
  void addToUpdatedFoodList(FoodRecord item) => updatedFoodList.add(item);
  void removeFromUpdatedFoodList(FoodRecord item) =>
    updatedFoodList.remove(item);
  void removeAtIndexFromUpdatedFoodList(int index) =>
    updatedFoodList.removeAt(index);
  void insertAtIndexInUpdatedFoodList(int index, FoodRecord item) =>
    updatedFoodList.insert(index, item);
  void updateUpdatedFoodListAtIndex(int index, Function(FoodRecord) updateFn) =>
    updatedFoodList[index] = updateFn(updatedFoodList[index]);

  @override
  void initState(BuildContext context) {}

  @override
  void dispose() {}
}
```

Search page widget model

```
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'foodsearch_widget.dart' show FoodsearchWidget;
import 'package:flutter/material.dart';

class FoodsearchModel extends FlutterFlowModel<FoodsearchWidget> {
  /// Local state fields for this page.
```

```

List<FoodRecord> searchFoodList = [];
void addToSearchFoodList(FoodRecord item) => searchFoodList.add(item);
void removeFromSearchFoodList(FoodRecord item) => searchFoodList.remove(item);
void removeAtIndexFromSearchFoodList(int index) =>
    searchFoodList.removeAt(index);
void insertAtIndexInSearchFoodList(int index, FoodRecord item) =>
    searchFoodList.insert(index, item);
void updateSearchFoodListAtIndex(int index, Function(FoodRecord) updateFn) =>
    searchFoodList[index] = updateFn(searchFoodList[index]);

/// State fields for stateful widgets in this page.

// State field(s) for Searchbar widget.
FocusNode? searchbarFocusNode;
TextEditingController? searchbarTextController;
String? Function(BuildContext, String?)? searchbarTextControllerValidator;
List<FoodRecord> simpleSearchResults = [];

@override
void initState(BuildContext context) {}

@override
void dispose() {
    searchbarFocusNode?.dispose();
    searchbarTextController?.dispose();
}
}

```

Recipe page widget model

```

import '../flutter_flow/flutter_flow_util.dart';
import 'recipe_widget.dart' show RecipeWidget;
import 'package:flutter/material.dart';

class RecipeModel extends FlutterFlowModel<RecipeWidget> {
    /// State fields for stateful widgets in this page.

    // State field(s) for TabBar widget.
    TabController? tabBarController;
    int get tabBarCurrentIndex =>
        tabBarController != null ? tabBarController!.index : 0;

    @override

```

```

void initState(BuildContext context) {}

@override
void dispose() {
  tabBarController?.dispose();
}
}

```

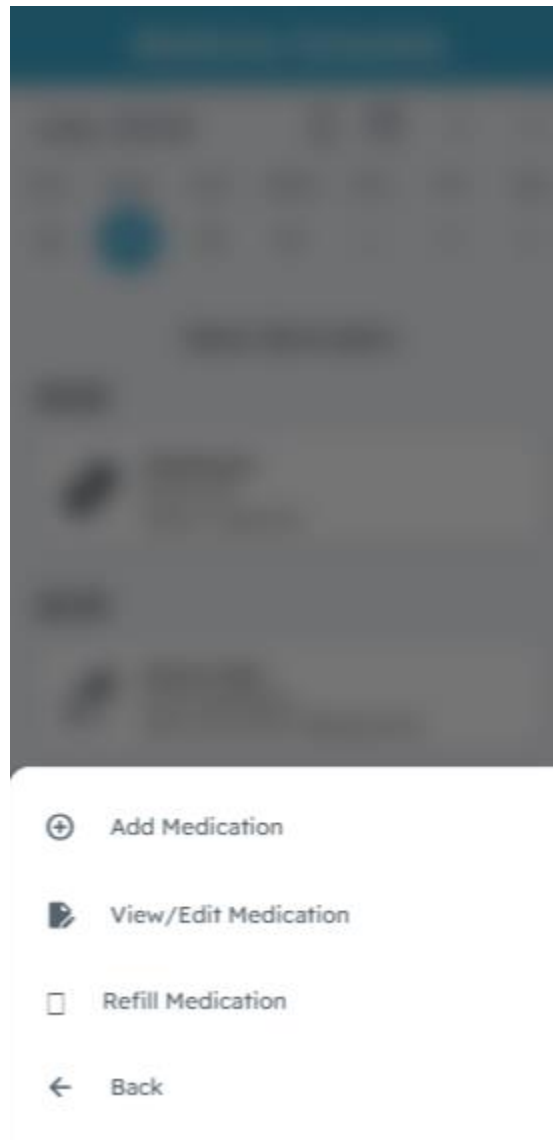
3. Medication Management

- **Medication Tracking:** The main page of the medication section keeps track of medications, dosages, and schedules for a streamlined medication routine.



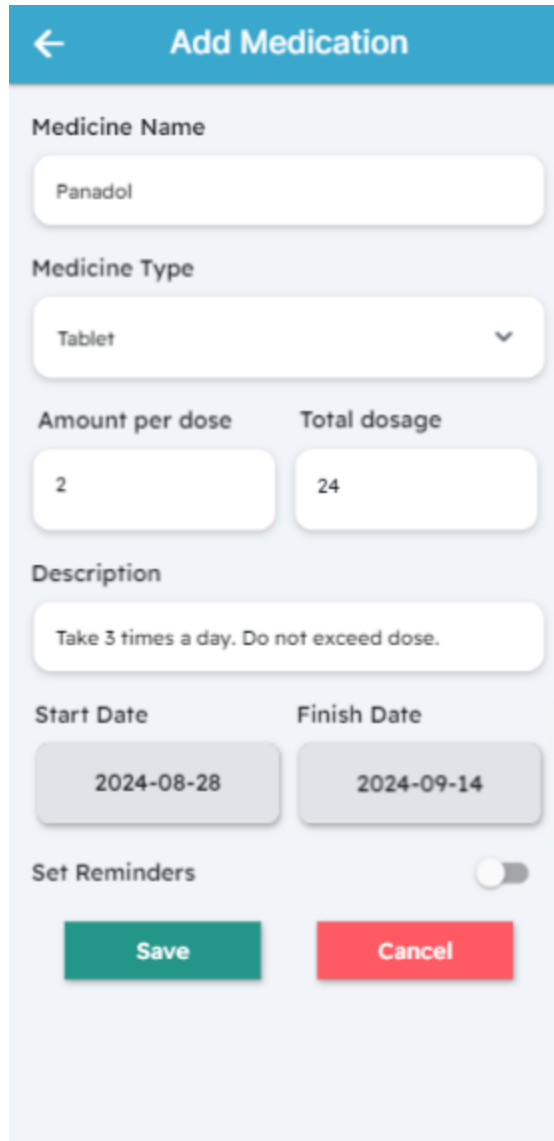
(xvii) Main medication page

- **Managing medicines:** The floating action button on the bottom right corner of the main page brings up a submenu with various options for dealing with the user's medicines.



(xviii) Medication submenu

- **Add Medication:** Brings up a medication form for recording medication information.

A mobile application form titled "Add Medication" with a blue header bar containing a back arrow. The form is divided into several sections: "Medicine Name" with a text input field containing "Panadol"; "Medicine Type" with a dropdown menu showing "Tablet"; "Amount per dose" and "Total dosage" with text input fields containing "2" and "24" respectively; "Description" with a text input field containing "Take 3 times a day. Do not exceed dose."; "Start Date" and "Finish Date" with date pickers showing "2024-08-28" and "2024-09-14"; and "Set Reminders" with a toggle switch. At the bottom are two buttons: "Save" (green) and "Cancel" (red).

← Add Medication

Medicine Name

Panadol

Medicine Type

Tablet

Amount per dose

2

Total dosage

24

Description

Take 3 times a day. Do not exceed dose.

Start Date

2024-08-28

Finish Date

2024-09-14

Set Reminders

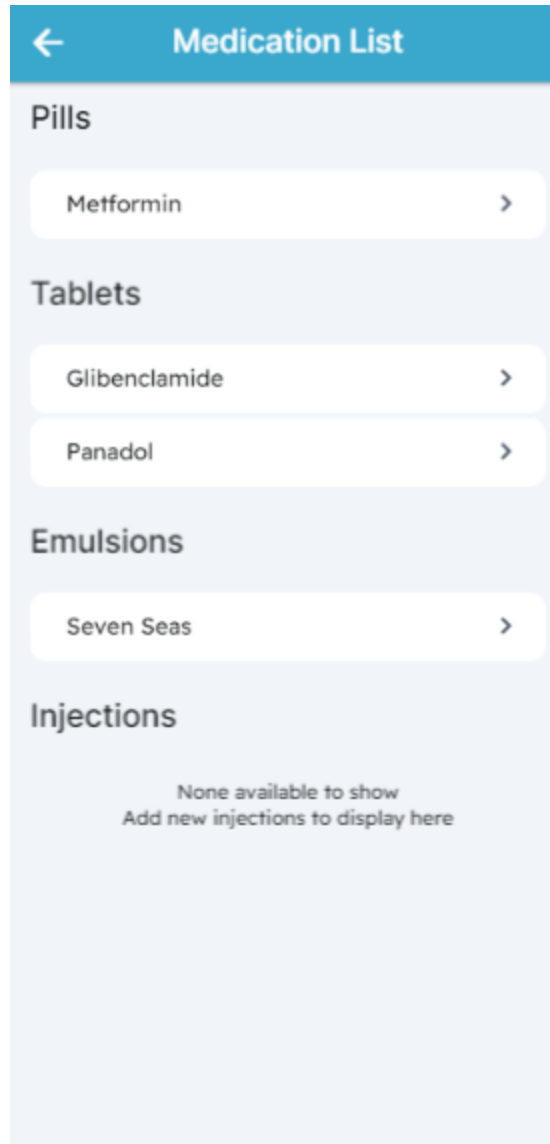
Save Cancel

(xix) Medication form

- **View/Edit Medication:**

Displays a list of medicines the user has recorded over time.

One can view information on each individual medicine and edit it if needed.



(xx) Medication list

Panadol ×

View or Edit medicine information.

Form: Tablet
Single Dose: 2 tablets
Total Dosage: 24 tablets
Starting Date: 2024-08-28
Ending Date: 2024-09-14
Scheduled times: 08:00

Special Instructions:

Take 3 times a day. Do not exceed dose.

[Edit](#) [Back](#)

(xxi) View medication information

← Edit Medication

Medicine Name

Panadol

Medicine Type

Tablet

Amount per dose

2

Total dosage

30

Description

Take 3 times a day. Do not exceed dose.

Start Date

2024-08-28

Finish Date

2024-09-14

Set Reminders

Reminders

Reminder 1

Auto-Remind

Automatically reminds you regularly

(xxii) Edit medication details

○ **Refill Medication:**

A medicine can be refilled at any point in time, it will update the remainder with the additional amount.

If a medicine is close to running out, the user is alerted and prompted to refill it.

The image shows a mobile application interface for refilling medicine. At the top, there is a dark teal header with the text "Refill Medicine(Panadol)". Below this, there is a list of items, each with a small icon and text, but they are blurred. Below the list, there is a section titled "Refill Medicine(Panadol)". Under this title, there is a label "Amount added:" followed by a text input field containing the number "16" and the word "tablets". Below this input field, there is a large blue button with the text "Refill" in white.

(xxiii) Refilling a medicine

- **Reminder System:** Receive timely reminders to ensure consistent medication adherence, through use of push notifications.

←

Add Medication

Start Date

2024-08-28

Finish Date

2024-09-14

Set Reminders

Reminders

Reminder 1

Auto-Remind

Automatically reminds you regularly

Frequency

Daily

Time:

08:00

Custom date range

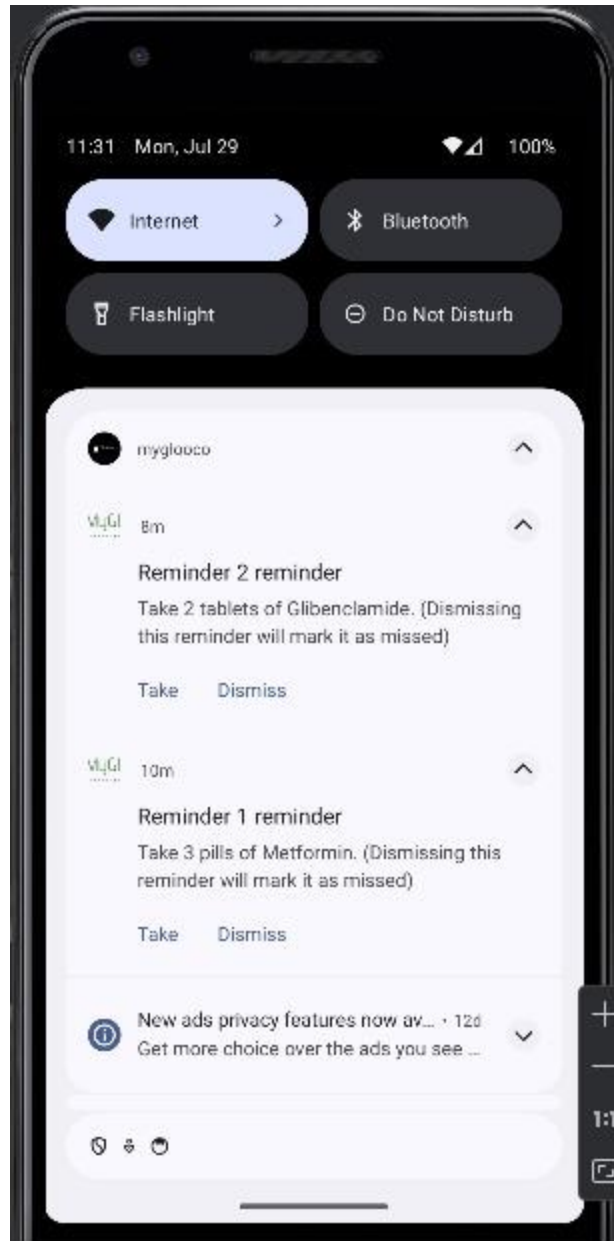
Set specific start and end dates for this reminder

Add reminder

Save

Cancel

(xxiv) Setting a reminder (in medication form)



(xxv) Reminders get triggered at scheduled time

Sample source code

Main Medication page widget model

```
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_calendar.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/medication/home_reminder/home_reminder_widget.dart';
import '/medication/no_elements/no_elements_widget.dart';
import 'medication_home_widget.dart' show MedicationHomeWidget;
import 'package:flutter/material.dart';
```

```

class MedicationHomeModel extends FlutterFlowModel<MedicationHomeWidget> {
  /// Local state fields for this page.

  List<String> listOfUpcomingTimes = [];
  void addToListOfUpcomingTimes(String item) => listOfUpcomingTimes.add(item);
  void removeFromListOfUpcomingTimes(String item) =>
    listOfUpcomingTimes.remove(item);
  void removeAtIndexFromListOfUpcomingTimes(int index) =>
    listOfUpcomingTimes.removeAt(index);
  void insertAtIndexInListOfUpcomingTimes(int index, String item) =>
    listOfUpcomingTimes.insert(index, item);
  void updateListOfUpcomingTimesAtIndex(int index, Function(String) updateFn) =>
    listOfUpcomingTimes[index] = updateFn(listOfUpcomingTimes[index]);

  List<RemindersRecord> upcomingReminders = [];
  void addToUpcomingReminders(RemindersRecord item) =>
    upcomingReminders.add(item);
  void removeFromUpcomingReminders(RemindersRecord item) =>
    upcomingReminders.remove(item);
  void removeAtIndexFromUpcomingReminders(int index) =>
    upcomingReminders.removeAt(index);
  void insertAtIndexInUpcomingReminders(int index, RemindersRecord item) =>
    upcomingReminders.insert(index, item);
  void updateUpcomingRemindersAtIndex(
    int index, Function(RemindersRecord) updateFn) =>
    upcomingReminders[index] = updateFn(upcomingReminders[index]);

  String selectedDay = 'none';

  List<String> listOfTakenTimes = [];
  void addToListOfTakenTimes(String item) => listOfTakenTimes.add(item);
  void removeFromListOfTakenTimes(String item) => listOfTakenTimes.remove(item);
  void removeAtIndexFromListOfTakenTimes(int index) =>
    listOfTakenTimes.removeAt(index);
  void insertAtIndexInListOfTakenTimes(int index, String item) =>
    listOfTakenTimes.insert(index, item);
  void updateListOfTakenTimesAtIndex(int index, Function(String) updateFn) =>
    listOfTakenTimes[index] = updateFn(listOfTakenTimes[index]);

  List<IndividualRemindersRecord> takenReminders = [];
  void addToTakenReminders(IndividualRemindersRecord item) =>
    takenReminders.add(item);
  void removeFromTakenReminders(IndividualRemindersRecord item) =>
    takenReminders.remove(item);

```

```

void removeAtIndexFromTakenReminders(int index) =>
    takenReminders.removeAt(index);
void insertAtIndexInTakenReminders(
    int index, IndividualRemindersRecord item) =>
    takenReminders.insert(index, item);
void updateTakenRemindersAtIndex(
    int index, Function(IndividualRemindersRecord) updateFn) =>
    takenReminders[index] = updateFn(takenReminders[index]);

List<String> listOfMissedTimes = [];
void addToListOfMissedTimes(String item) => listOfMissedTimes.add(item);
void removeFromListOfMissedTimes(String item) =>
    listOfMissedTimes.remove(item);
void removeAtIndexFromListOfMissedTimes(int index) =>
    listOfMissedTimes.removeAt(index);
void insertAtIndexInListOfMissedTimes(int index, String item) =>
    listOfMissedTimes.insert(index, item);
void updateListOfMissedTimesAtIndex(int index, Function(String) updateFn) =>
    listOfMissedTimes[index] = updateFn(listOfMissedTimes[index]);

List<IndividualRemindersRecord> missedReminders = [];
void addToMissedReminders(IndividualRemindersRecord item) =>
    missedReminders.add(item);
void removeFromMissedReminders(IndividualRemindersRecord item) =>
    missedReminders.remove(item);
void removeAtIndexFromMissedReminders(int index) =>
    missedReminders.removeAt(index);
void insertAtIndexInMissedReminders(
    int index, IndividualRemindersRecord item) =>
    missedReminders.insert(index, item);
void updateMissedRemindersAtIndex(
    int index, Function(IndividualRemindersRecord) updateFn) =>
    missedReminders[index] = updateFn(missedReminders[index]);

String today = 'date';

String calendarDate = 'date';

List<IndividualRemindersRecord> pastMissedIndividualReminders = [];
void addToPastMissedIndividualReminders(IndividualRemindersRecord item) =>
    pastMissedIndividualReminders.add(item);
void removeFromPastMissedIndividualReminders(
    IndividualRemindersRecord item) =>
    pastMissedIndividualReminders.remove(item);
void removeAtIndexFromPastMissedIndividualReminders(int index) =>

```

```

        pastMissedIndividualReminders.removeAt(index);
void insertAtIndexInPastMissedIndividualReminders(
    int index, IndividualRemindersRecord item) =>
    pastMissedIndividualReminders.insert(index, item);
void updatePastMissedIndividualRemindersAtIndex(
    int index, Function(IndividualRemindersRecord) updateFn) =>
    pastMissedIndividualReminders[index] =
        updateFn(pastMissedIndividualReminders[index]);

List<IndividualRemindersRecord> todayMissedIndividualReminders = [];
void addToTodayMissedIndividualReminders(IndividualRemindersRecord item) =>
    todayMissedIndividualReminders.add(item);
void removeFromTodayMissedIndividualReminders(
    IndividualRemindersRecord item) =>
    todayMissedIndividualReminders.remove(item);
void removeAtIndexFromTodayMissedIndividualReminders(int index) =>
    todayMissedIndividualReminders.removeAt(index);
void insertAtIndexInTodayMissedIndividualReminders(
    int index, IndividualRemindersRecord item) =>
    todayMissedIndividualReminders.insert(index, item);
void updateTodayMissedIndividualRemindersAtIndex(
    int index, Function(IndividualRemindersRecord) updateFn) =>
    todayMissedIndividualReminders[index] =
        updateFn(todayMissedIndividualReminders[index]);

/// State fields for stateful widgets in this page.

// Stores action output result for [Firestore Query - Query a collection]
action in MedicationHome widget.
List<RemindersRecord>? allReminders;
// Stores action output result for [Firestore Query - Query a collection]
action in MedicationHome widget.
List<IndividualRemindersRecord>? currentSubReminders;
// Stores action output result for [Custom Action - requestPermissions] action
in MedicationHome widget.
bool? permissionsGranted;
// Stores action output result for [Firestore Query - Query a collection]
action in MedicationHome widget.
List<MedicineRecord>? lowCapacityMeds;
// State field(s) for Calendar widget.
DateTimeRange? calendarSelectedDay;
// Stores action output result for [Firestore Query - Query a collection]
action in Calendar widget.
List<RemindersRecord>? calendarReminders;

```



```

    // Stores action output result for [Firestore Query - Query a collection]
    action in Calendar widget.
    List<IndividualRemindersRecord>? todaySubReminders;
    // Stores action output result for [Firestore Query - Query a collection]
    action in Calendar widget.
    List<IndividualRemindersRecord>? calendarSubReminders;
    // Models for homeReminder dynamic component.
    late FlutterFlowDynamicModels<HomeReminderModel> homeReminderModels;
    // Model for noElements component.
    late NoElementsModel noElementsModel1;
    // Model for noElements component.
    late NoElementsModel noElementsModel2;
    // Model for noElements component.
    late NoElementsModel noElementsModel3;

    @override
    void initState(BuildContext context) {
        calendarSelectedDay = DateTimeRange(
            start: DateTime.now().startOfDay,
            end: DateTime.now().endOfDay,
        );
        homeReminderModels = FlutterFlowDynamicModels(() => HomeReminderModel());
        noElementsModel1 = createModel(context, () => NoElementsModel());
        noElementsModel2 = createModel(context, () => NoElementsModel());
        noElementsModel3 = createModel(context, () => NoElementsModel());
    }

    @override
    void dispose() {
        homeReminderModels.dispose();
        noElementsModel1.dispose();
        noElementsModel2.dispose();
        noElementsModel3.dispose();
    }
}

```

Medication Form widget model

```

import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/flutter_flow/form_field_controller.dart';
import '/medication/new_reminder/new_reminder_widget.dart';
import 'medication_form_widget.dart' show MedicationFormWidget;
import 'package:flutter/material.dart';

```

```

class MedicationFormModel extends FlutterFlowModel<MedicationFormWidget> {
  /// Local state fields for this page.

  List<DocumentReference> reminderIDList = [];
  void addToReminderIDList(DocumentReference item) => reminderIDList.add(item);
  void removeFromReminderIDList(DocumentReference item) =>
    reminderIDList.remove(item);
  void removeAtIndexFromReminderIDList(int index) =>
    reminderIDList.removeAt(index);
  void insertAtIndexInReminderIDList(int index, DocumentReference item) =>
    reminderIDList.insert(index, item);
  void updateReminderIDListAtIndex(
    int index, Function(DocumentReference) updateFn) =>
    reminderIDList[index] = updateFn(reminderIDList[index]);

  String startDate = '2000-01-01';

  String endDate = '2100-01-01';

  bool reminderSetState = false;

  /// State fields for stateful widgets in this page.

  // State field(s) for mainColumn widget.
  ScrollController? mainColumn;
  // State field(s) for MedName widget.
  FocusNode? medNameFocusNode;
  TextEditingController? medNameTextController;
  String? Function(BuildContext, String?)? medNameTextControllerValidator;
  // State field(s) for MedType widget.
  String? medTypeValue;
  FormFieldController<String>? medTypeValueController;
  // State field(s) for SingleDose widget.
  FocusNode? singleDoseFocusNode;
  TextEditingController? singleDoseTextController;
  String? Function(BuildContext, String?)? singleDoseTextControllerValidator;
  // State field(s) for TotalDose widget.
  FocusNode? totalDoseFocusNode;
  TextEditingController? totalDoseTextController;
  String? Function(BuildContext, String?)? totalDoseTextControllerValidator;
  // State field(s) for DescriptionText widget.
  FocusNode? descriptionTextFocusNode;
  TextEditingController? descriptionTextTextController;
  String? Function(BuildContext, String?)?

```

```

        descriptionTextTextControllerValidator;
DateTime? datePicked1;
DateTime? datePicked2;
// State field(s) for RemindersSet widget.
bool? remindersSetValue;
// State field(s) for ListView widget.
ScrollController? listViewController;
// Models for newReminder dynamic component.
late FlutterFlowDynamicModels<NewReminderModel> newReminderModels;
// Stores action output result for [Backend Call - Create Document] action in
Row widget.
RemindersRecord? newReminderID;

@override
void initState(BuildContext context) {
    mainColumn = ScrollController();
    listViewController = ScrollController();
    newReminderModels = FlutterFlowDynamicModels(() => NewReminderModel());
}

@override
void dispose() {
    mainColumn?.dispose();
    medNameFocusNode?.dispose();
    medNameTextController?.dispose();

    singleDoseFocusNode?.dispose();
    singleDoseTextController?.dispose();

    totalDoseFocusNode?.dispose();
    totalDoseTextController?.dispose();

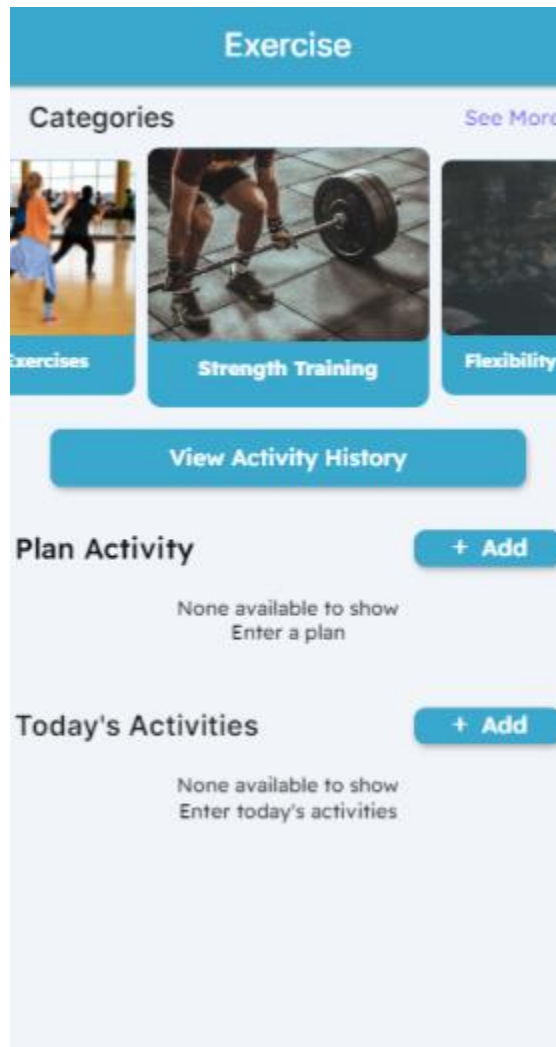
    descriptionTextFocusNode?.dispose();
    descriptionTextTextController?.dispose();

    listViewController?.dispose();
    newReminderModels.dispose();
}
}

```

4. Physical Activity Integration

- **Exercise Module:** The main page contains links to different sections of the module.



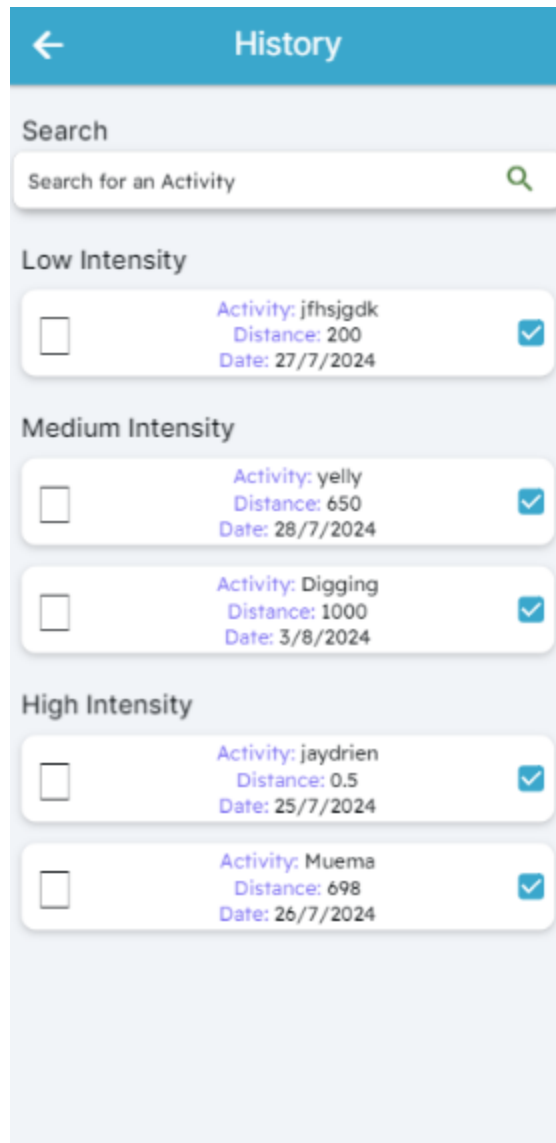
(xxvi) Exercise main page

- **Planning/Logging activities:** One can schedule activities to perform in future or log those that have been recently performed if they were not already planned.

The screenshot shows a mobile application interface for planning an exercise activity. At the top, there is a dark blue header with the word "Exercise". Below it, a section titled "Categories" shows two images of people exercising, with a "See More" link. The main part of the screen is a light blue modal titled "Plan activity". Inside this modal, there are several input fields and controls: "Activity Name" with the text "Sprinting"; "Record distance" with a toggle switch turned on and a description "Whether the activity involves covering a certain amount of distance"; "Distance Covered" with the value "1000"; "Date" with a "Pick Date" button and the selected date "30/8/2024"; "Time" with "Start Time" and "Finish Time" both set to "Aug 28, 2024"; and "Intensities" with three buttons: "Low", "Medium" (which is highlighted), and "High". At the bottom of the modal is a blue "Add" button.

(xxvii) Activity planning sheet

- **Activity History:** Viewing a list of all previously completed exercises/activities.



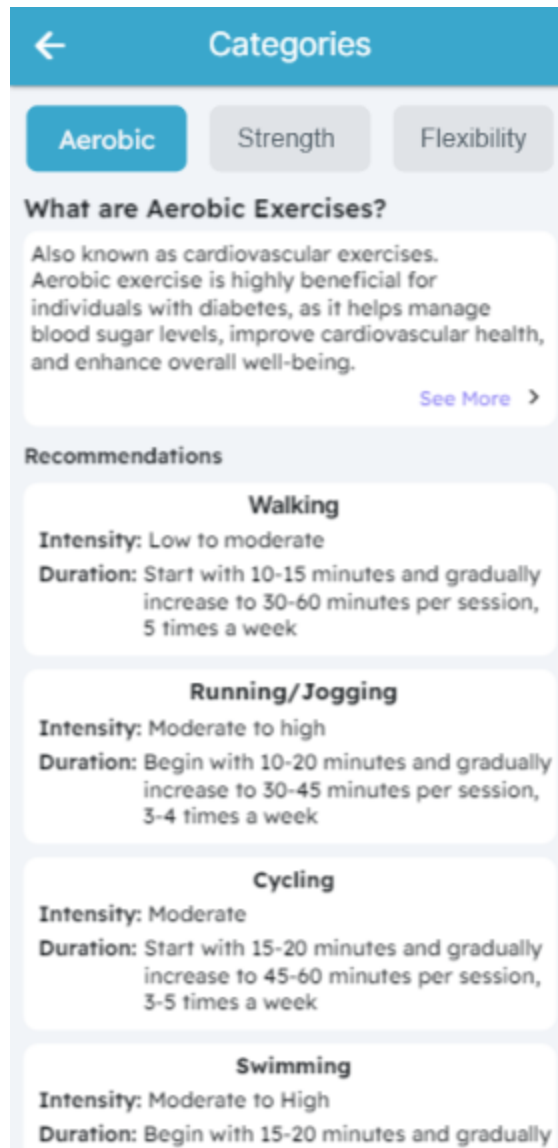
(xxviii) Previously completed activites/exercises

- One can also search for a specific completed activity instead of looking for it in a possible long list.



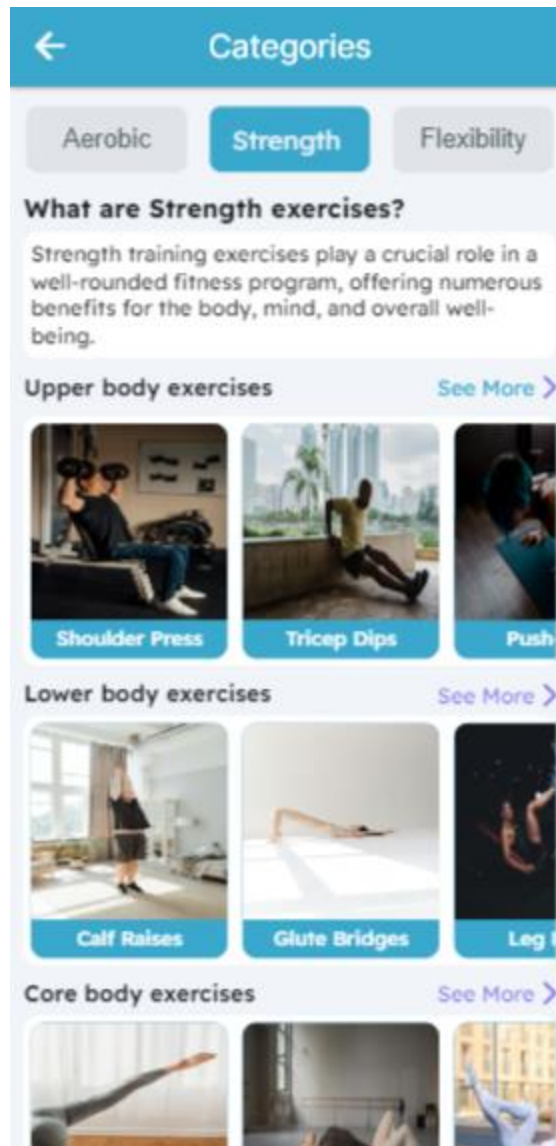
(xxix) Activity search

- **Exercise Categories:** Categorization of different types of exercises. User can choose from an assorted list of various exercises in each category.
 - **Aerobic activities:** contains a description of aerobic activities with examples.

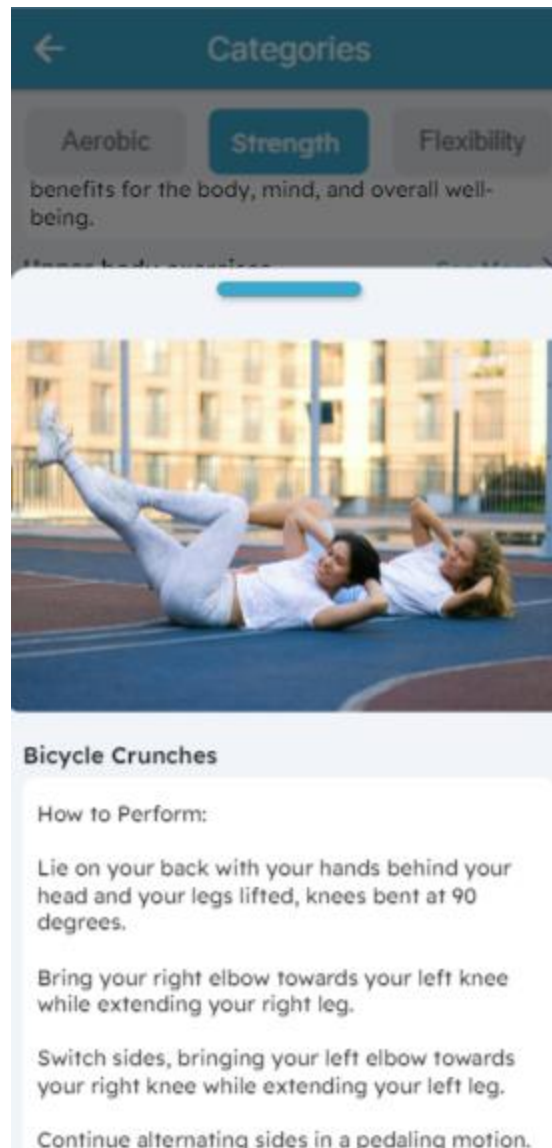


(xxx) Sample of aerobic activities

- **Strength exercises:** contains a description of strength exercises with examples.

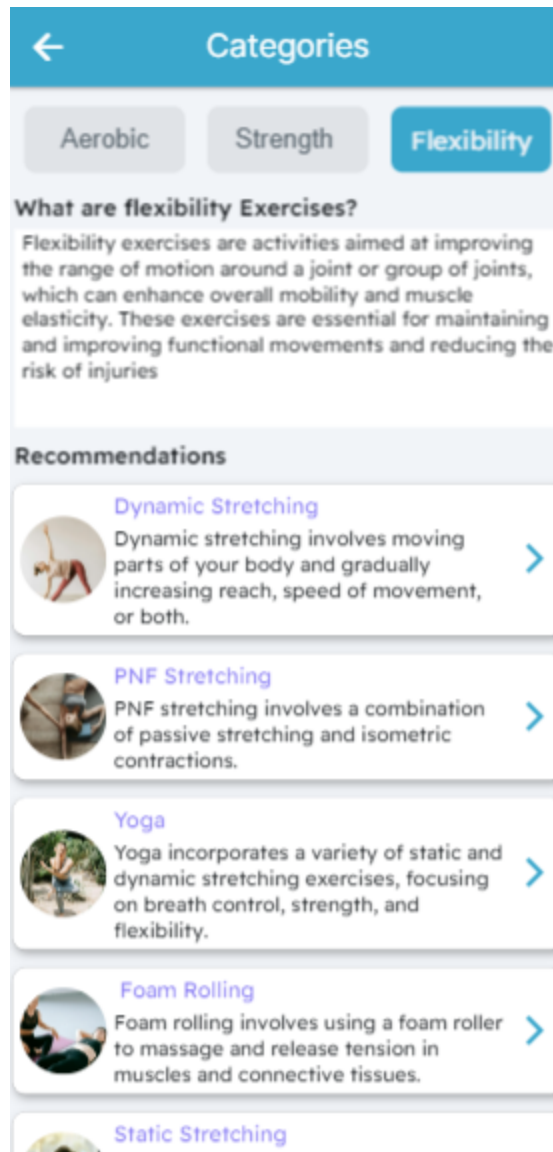


(xxxi) Sample of different strength exercise types



(xxxii) Description of a sample strength exercise

- **Flexibility exercises:** contains a description of flexibility exercises with examples.



(xxxiii) Sample of different flexibility exercises

Sample source code

Main Exercise Page widget model

```
import
'/exercises/exercisetaskstodaycomponent/exercisetaskstodaycomponent_widget.dart';
import '/exercises/incompleteactivitylist/incompleteactivitylist_widget.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'exerciseshomepage_widget.dart' show ExerciseshomepageWidget;
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flutter/material.dart';
```

```

class ExercisehomepageModel extends FlutterFlowModel<ExercisehomepageWidget> {
  /// State fields for stateful widgets in this page.

  // State field(s) for Carousel widget.
  CarouselController? carouselController;
  int carouselCurrentIndex = 1;

  // Models for incompleteactivitylist dynamic component.
  late FlutterFlowDynamicModels<IncompleteactivitylistModel>
    incompleteactivitylistModels;
  // Models for exercisetaskstodaycomponent dynamic component.
  late FlutterFlowDynamicModels<ExercisetaskstodaycomponentModel>
    exercisetaskstodaycomponentModels;

  @override
  void initState(BuildContext context) {
    incompleteactivitylistModels =
      FlutterFlowDynamicModels(() => IncompleteactivitylistModel());
    exercisetaskstodaycomponentModels =
      FlutterFlowDynamicModels(() => ExercisetaskstodaycomponentModel());
  }

  @override
  void dispose() {
    incompleteactivitylistModels.dispose();
    exercisetaskstodaycomponentModels.dispose();
  }
}

```

Categories page model widget

```

import
'/exercises/flexibilityexercisecomponent/flexibilityexercisecomponent_widget.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'categoriespage_widget.dart' show CategoriespageWidget;
import 'package:flutter/material.dart';

class CategoriespageModel extends FlutterFlowModel<CategoriespageWidget> {
  /// State fields for stateful widgets in this page.

  // State field(s) for TabBar widget.
  TabController? tabBarController;
  int get tabBarCurrentIndex =>

```

```

        tabBarController != null ? tabBarController!.index : 0;

// Models for flexibilityexercisecomponent dynamic component.
late FlutterFlowDynamicModels<FlexibilityexercisecomponentModel>
    flexibilityexercisecomponentModels;

@override
void initState(BuildContext context) {
    flexibilityexercisecomponentModels =
        FlutterFlowDynamicModels(() => FlexibilityexercisecomponentModel());
}

@override
void dispose() {
    tabBarController?.dispose();
    flexibilityexercisecomponentModels.dispose();
}
}

```

History List page widget model

```

import
'/exercises/highintensitylistcomponents/highintensitylistcomponents_widget.dart';
import
'/exercises/lowintensitylistcomponent/lowintensitylistcomponent_widget.dart';
import
'/exercises/mediumsintensitylistcomponent/mediumsintensitylistcomponent_widget.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'listpage_widget.dart' show ListpageWidget;
import 'package:flutter/material.dart';

class ListpageModel extends FlutterFlowModel<ListpageWidget> {
    /// State fields for stateful widgets in this page.

    // Models for lowintensitylistcomponent dynamic component.
    late FlutterFlowDynamicModels<LowintensitylistcomponentModel>
        lowintensitylistcomponentModels;
    // Models for mediumsintensitylistcomponent dynamic component.
    late FlutterFlowDynamicModels<MediumsintensitylistcomponentModel>
        mediumsintensitylistcomponentModels;
    // Models for highintensitylistcomponents dynamic component.
    late FlutterFlowDynamicModels<HighintensitylistcomponentsModel>
        highintensitylistcomponentsModels;
}

```

```

@override
void initState(BuildContext context) {
  lowintensitylistcomponentModels =
    FlutterFlowDynamicModels(() => LowintensitylistcomponentModel());
  mediumsintensitylistcomponentModels =
    FlutterFlowDynamicModels(() => MediumsintensitylistcomponentModel());
  highintensitylistcomponentsModels =
    FlutterFlowDynamicModels(() => HighintensitylistcomponentsModel());
}

@override
void dispose() {
  lowintensitylistcomponentModels.dispose();
  mediumsintensitylistcomponentModels.dispose();
  highintensitylistcomponentsModels.dispose();
}
}

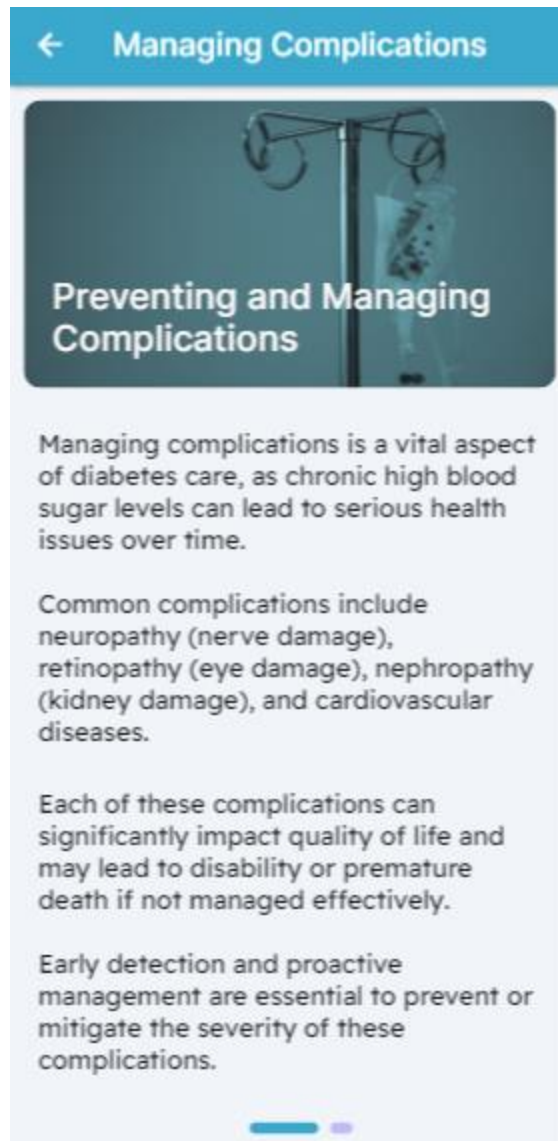
```

5. Educational Support

- **General Diabetes Education:** Explore a dedicated section with comprehensive information about diabetes management.



(xxxiv) *Education main page*



(xxxv) *Example of an educational tab*

Sample source code

Main Education page widget model

```
import '/flutter_flow/flutter_flow_util.dart';
import 'edhome_widget.dart' show EdhomeWidget;
import 'package:flutter/material.dart';

class EdhomeModel extends FlutterFlowModel<EdhomeWidget> {
  @override
  void initState(BuildContext context) {}
}
```



```

@override
void dispose() {}
}

```

Complications module(Example) widget model

```

import '/flutter_flow/flutter_flow_util.dart';
import 'complications_widget.dart' show ComplicationsWidget;
import 'package:flutter/material.dart';

class ComplicationsModel extends FlutterFlowModel<ComplicationsWidget> {
  /// State fields for stateful widgets in this page.

  // State field(s) for PageView widget.
  PageController? pageViewController;

  int get pageViewCurrentIndex => pageViewController != null &&
    pageViewController!.hasClients &&
    pageViewController!.page != null
    ? pageViewController!.page!.round()
    : 0;

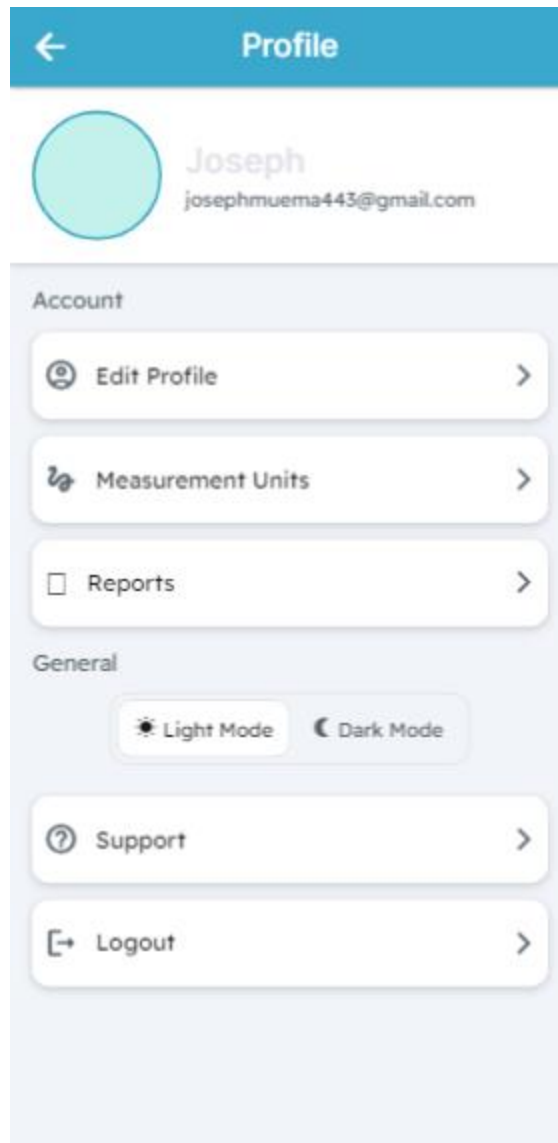
  @override
  void initState(BuildContext context) {}

  @override
  void dispose() {}
}

```

- Every education module covers different aspects of life related to diabetes, as well as providing links to further studies/research.
- This feature set empowers users to take control of their diabetes by providing the tools and knowledge to track progress, make informed decisions, and build healthy habits, all within a single user-friendly application.

6. Profile Settings



(xxxvi) Profile page

- Contains different profile options. The major ones include;
 - **Edit profile:** One can change the details they entered when first signing up.

The screenshot shows a mobile application interface for editing a user's profile. At the top, there is a dark blue header with a back arrow and the title 'Profile'. Below this, a grey section displays the user's current profile: a circular profile picture, the name 'Joseph', and the email 'josephmuema443@gmail.com'. Underneath, a section titled 'Account' contains an 'Edit Profile' button with a right-pointing arrow. The main part of the screen is a white form with rounded corners. It starts with a large circular profile picture placeholder with a green outline and a '+' icon. Below this are three text input fields: 'Firstname' (containing 'Joseph'), 'Surname' (containing 'Muema'), and 'Phone Number' (containing '798730844'). Each field has a small 'x' icon to clear the text. The 'Gender' section has three radio button options: 'Male' (selected), 'Female', and 'Other'. At the bottom of the form is a large blue 'Save' button.

(xxxvii) Changing profile details

- **Report generation:** A user can generate a report on various sections of data they have been recording over time. This can be useful for analysis by health care providers if requested.

←

Profile

Joseph

josephmuema443@gmail.com

Account

Edit Profile

>

Measurement Units

>

Report Configuration

Select Start Date:

Aug 1, 2024

Select End Date:

Aug 28, 2024

Diet Logs, Medication Logs, Blood Sugar

▼

Generate

(xxxviii) Generating a comprehensive data report

Blood Sugar Readings

Date	Period	CGM Reading (mg/dL)
2024-07-28	before meal	188.0
2024-07-29	after meal	110.0
2024-07-28	before meal	122.0
2024-07-26	before meal	120.0
2024-07-29	before meal	140.0
2024-07-29	after meal	83.0
2024-07-29	before meal	154.0
2024-07-29	before meal	140.0
2024-07-29	after meal	130.0
2024-07-29	after meal	160.0
2024-07-29	after meal	178.1
2024-07-29	before meal	110.6
2024-07-29	after meal	106.3
2024-07-29	after meal	165.7

(xxxviii) Example of a blood sugar report generated

Sample source code

Profile page widget model

```
import '/flutter_flow/flutter_flow_util.dart';

import '/flutter_flow/form_field_controller.dart';
import 'edit_profile_widget.dart' show EditProfileWidget;
import 'package:flutter/material.dart';

class EditProfileModel extends FlutterFlowModel<EditProfileWidget> {
  /// State fields for stateful widgets in this component.

  bool isDataUploading1 = false;
  FFUploadedFile uploadedLocalFile1 =
    FFUploadedFile(bytes: Uint8List.fromList([]));
  String uploadedFileUrl1 = '';

  bool isDataUploading2 = false;
  FFUploadedFile uploadedLocalFile2 =
```

```

        FFUploadedFile(bytes: Uint8List.fromList([]));
String uploadedFileUrl2 = '';

// State field(s) for TextField widget.
FocusNode? textFieldFocusNode1;
TextEditingController? textController1;
String? Function(BuildContext, String?)? textController1Validator;
// State field(s) for TextField widget.
FocusNode? textFieldFocusNode2;
TextEditingController? textController2;
String? Function(BuildContext, String?)? textController2Validator;
// State field(s) for TextField widget.
FocusNode? textFieldFocusNode3;
TextEditingController? textController3;
String? Function(BuildContext, String?)? textController3Validator;
// State field(s) for ChoiceChips widget.
FormFieldController<List<String>>? choiceChipsValueController;
String? get choiceChipsValue =>
    choiceChipsValueController?.value?.firstOrNull;
set choiceChipsValue(String? val) =>
    choiceChipsValueController?.value = val != null ? [val] : [];

@override
void initState(BuildContext context) {}

@override
void dispose() {
    textFieldFocusNode1?.dispose();
    textController1?.dispose();

    textFieldFocusNode2?.dispose();
    textController2?.dispose();

    textFieldFocusNode3?.dispose();
    textController3?.dispose();
}
}

```

Report generation page widget model

```

import '/flutter_flow/flutter_flow_util.dart';
import '/flutter_flow/form_field_controller.dart';
import 'report2_widget.dart' show Report2Widget;

```

```

import 'package:flutter/material.dart';

class Report2Model extends FlutterFlowModel<Report2Widget> {
  /// State fields for stateful widgets in this component.

  DateTime? datePicked1;
  DateTime? datePicked2;
  // State field(s) for DropDown widget.
  List<String>? dropDownValue;
  FormFieldController<List<String>>? dropDownValueController;
  // Stores action output result for [Custom Action - generateReport] action in
  Button widget.
  String? maybe;

  @override
  void initState(BuildContext context) {}

  @override
  void dispose() {}
}

```

iv. **EVALUATION & REMARKS**

MyGlooco's effectiveness to addressing grievances by diabetics (established through a survey conducted before requirements were determined) will be evaluated by the following:

Test evaluation and comparison:

The program will be deployed to a selected audience alongside a market competitor and evaluated in comparison to the developed system. Feedback given will be used to improve upon the system.

Factors to be evaluated include the following:

- Glucose logging and analysis(visualization) capabilities
- Dietary planning
- Exercise logging and recommendations
- Quality of life features (User experience, user interface and educational tools and systems)

Remarks

The application is complete as is and there are plans to add even more useful functionalities in future like blood sugar prediction to allow users to be more cautious of their sugar intake, preventing extreme situations.

We hope to make this application accessible to most local users and aid them in their journey in Diabetes management.