CIS 552 – Database Design

Final Project Report

Calorie Counter Application

Group 11

Tabeesh Fatiya (19)
Yashika Patil (52)
Sanjay Dhayalan (17)

Introduction

In today's fast-paced world, maintaining a healthy lifestyle is increasingly difficult. With the rise of fast food, sedentary habits, and the busy nature of daily life, many people struggle to keep track of their diet, leading to health issues such as obesity, diabetes, and heart disease. As more people become aware of the importance of nutrition and calorie management, there is a growing need for tools that help monitor food intake and manage calories effectively.

Mobile applications have become powerful tools to address this need, allowing users to log their meals, track calorie intake, and make informed dietary choices. These apps not only make it easy to record food consumption but also provide insights into eating habits, helping users take control of their health. Python, combined with frameworks like Kivy for user interface development and MongoDB for backend data management, offers a strong and scalable foundation for building such applications.

This project focuses on developing a comprehensive Calorie Counter application using Python, Kivy, and MongoDB. The app helps users monitor their daily calorie intake by logging meals, retrieving nutritional information, and providing visual feedback on their progress. It also includes user authentication to ensure that each user's data is securely stored and accessible only to them. By leveraging Kivy for cross-platform UI development and MongoDB for managing food and user data, this application aims to deliver a seamless and user-friendly experience for health-conscious individuals.

Problem Statement

This project aims to develop a straightforward and user-friendly Calorie Counter application. The application will focus on simplicity and ease of use, allowing users to log their meals with minimal effort while providing accurate and essential nutritional information. It will also include basic yet effective security features, such as user authentication, to ensure that user data is protected without complicating the user experience. By prioritizing simplicity, essential functionality, and security, this project seeks to create a reliable tool that helps users effortlessly manage their daily calorie intake and make informed dietary choices.

Use Cases

- **Meal Entry Management:** Add meals to the daily intake, selecting from a global database of food items or manually entering new items.
- **View Daily Intake:** View a list of all meals added for the day, along with their calorie values.
- **Update Meal Entries:** Update the details of any previously added meal.
- **Delete Meals:** Delete any meal from the daily intake list.

- **Progress Tracking:** Monitor daily, weekly, and monthly caloric intake to help users achieve their fitness goals.
- **User Profile Management:** Create and manage user profiles, allowing for personalized tracking and recommendations.
- **Data Analysis and Reporting:** Generate reports and visualizations to help users understand their eating habits and make informed decisions.

Technologies used

MongoDB

Our application uses MongoDB, a NoSQL database management system, to store user data, meals, and a food database.

Python (pymongo)

Python is used for the backend of the application. The pymongo package facilitates the connection between Python and MongoDB, enabling various database operations like creating, reading, updating, and deleting (CRUD) records.

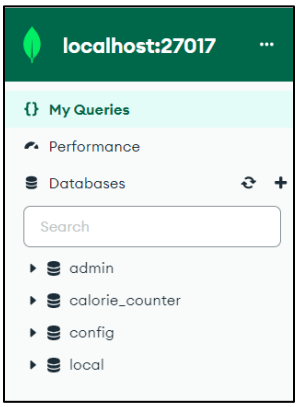
Kivy

The frontend of the application was built using Kivy. It's an open-source Python framework that supports multi-touch events and is compatible with multiple platforms such as Windows, Android, and Linux. Kivy allows for a customizable and responsive UI using the KV language.

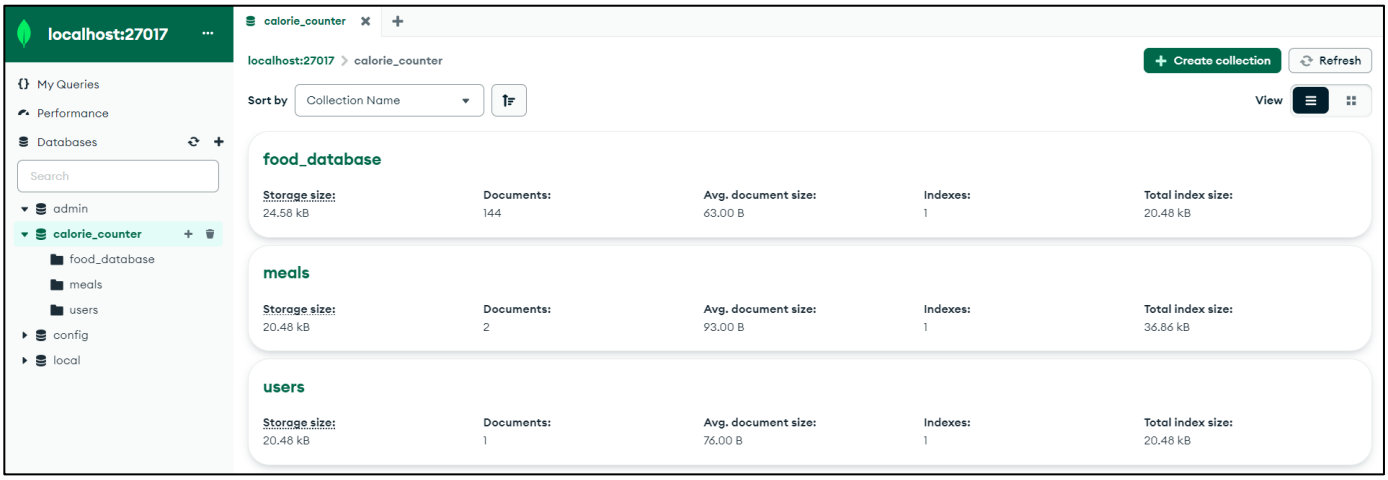
Incorporation of MongoDB

Database implementation and Data population

We started by creating a database called Calorie Counter.

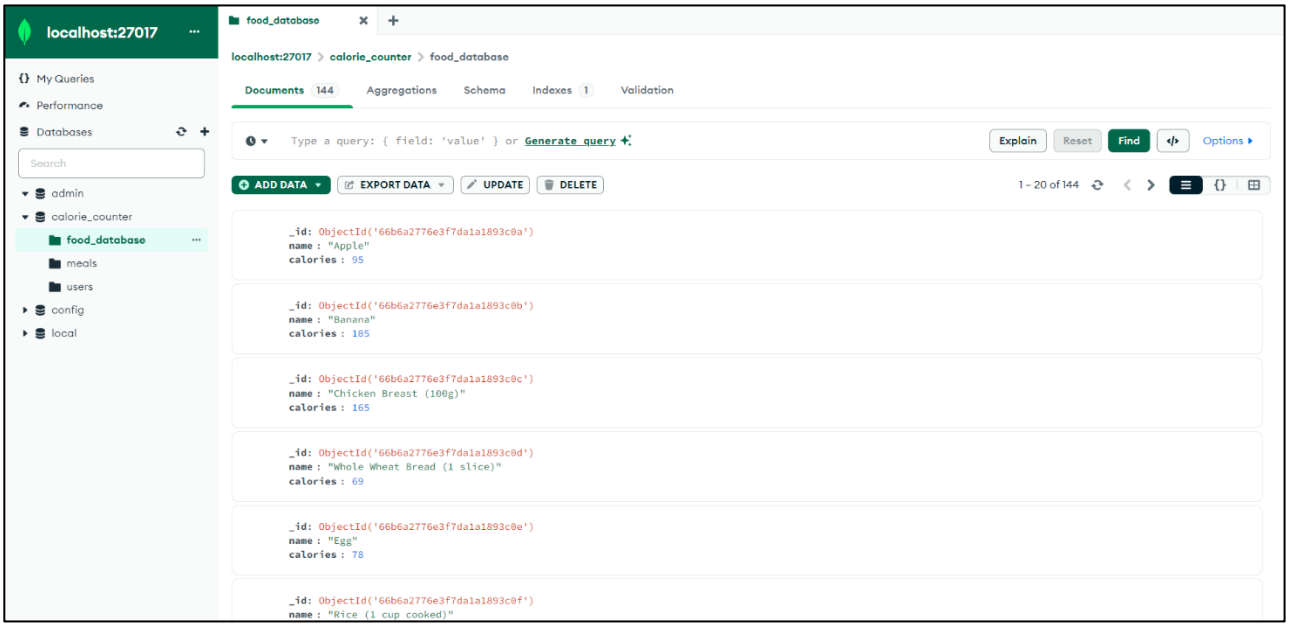


To effectively manage CRUD operations, we organized our MongoDB database around 3 collections: food_database, meals and users.

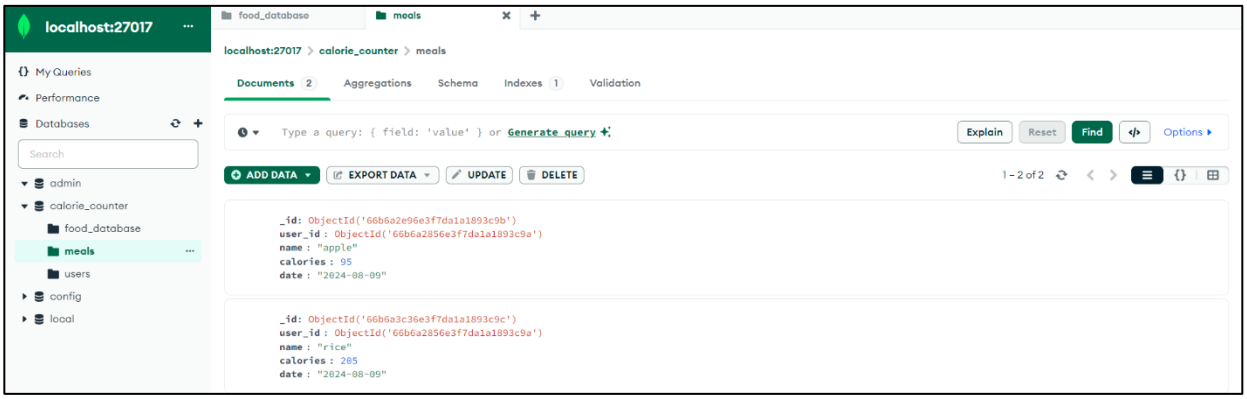


We utilized JSON files with dummy data, we populated the database. Each file corresponded to a collection, ensuring consistency with our schema. This approach provided a robust foundation for testing and development.

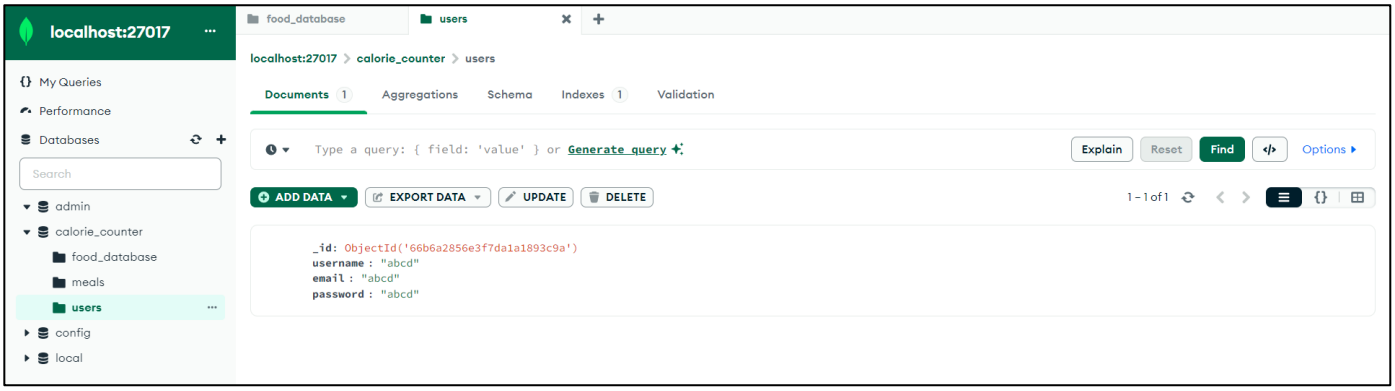
Below screenshot shows the populated food_database collection.



Below screenshot shows the meals collection.



Below screenshot shows the users collection.



Front end Interface Design

The front-end interface for the health tracking application is developed using Kivy, a Python framework for building cross-platform applications. The design emphasizes simplicity and user-friendliness, focusing on intuitive navigation and aesthetically pleasing elements. Below is an overview of the key components and design choices:

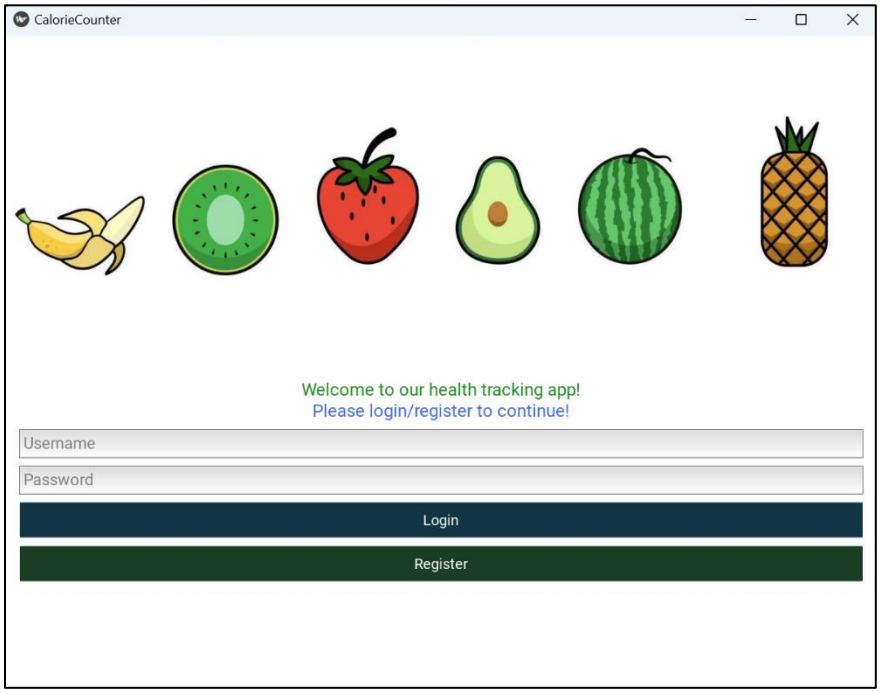
1. Login Screen

The login screen serves as the application's entry point, giving users the option to log in or register. Key features include:

- **Main Layout:** A `BoxLayout` with a vertical orientation is used as the main container, ensuring that elements are stacked vertically, creating a clean and organized look.
- **Background Image:** A background image is added to enhance visual appeal. The image is set to maintain its aspect ratio, providing a consistent experience across different screen sizes.
- **Welcome Message:** At the top of the form, a welcome message is displayed using a `Label`. The message is styled with a green and blue color scheme to create a warm and inviting tone. The text size is increased to improve readability.
- **Text Inputs:** The form includes `TextInput` widgets for entering the username and password. The height of these inputs is carefully adjusted to maintain a balanced appearance.
- **Buttons:** Two primary buttons, "Login" and "Register," are provided for user actions. The buttons are styled with distinct background colors and increased font size to make them more noticeable and accessible.
- **Message Label: A Label is included at the bottom of the form to display messages like**

login errors. The text is set to red to draw attention to any issues.

- **Positioning:** The form layout is centered on the screen using `pos_hint`, ensuring that it remains the focal point regardless of the screen size.



2. Color and Styling

The application employs a consistent color scheme throughout the interface, with white backgrounds and accent colors for buttons and text. This minimalist approach keeps the interface clean and uncluttered, allowing users to focus on the content.

3. Layout Components

The use of Kivy's layout components such as `BoxLayout`, `GridLayout`, and `ScrollView` ensures that the interface is adaptable to different devices and screen sizes. The layouts are designed to be responsive, providing a seamless experience on both mobile and desktop platforms.

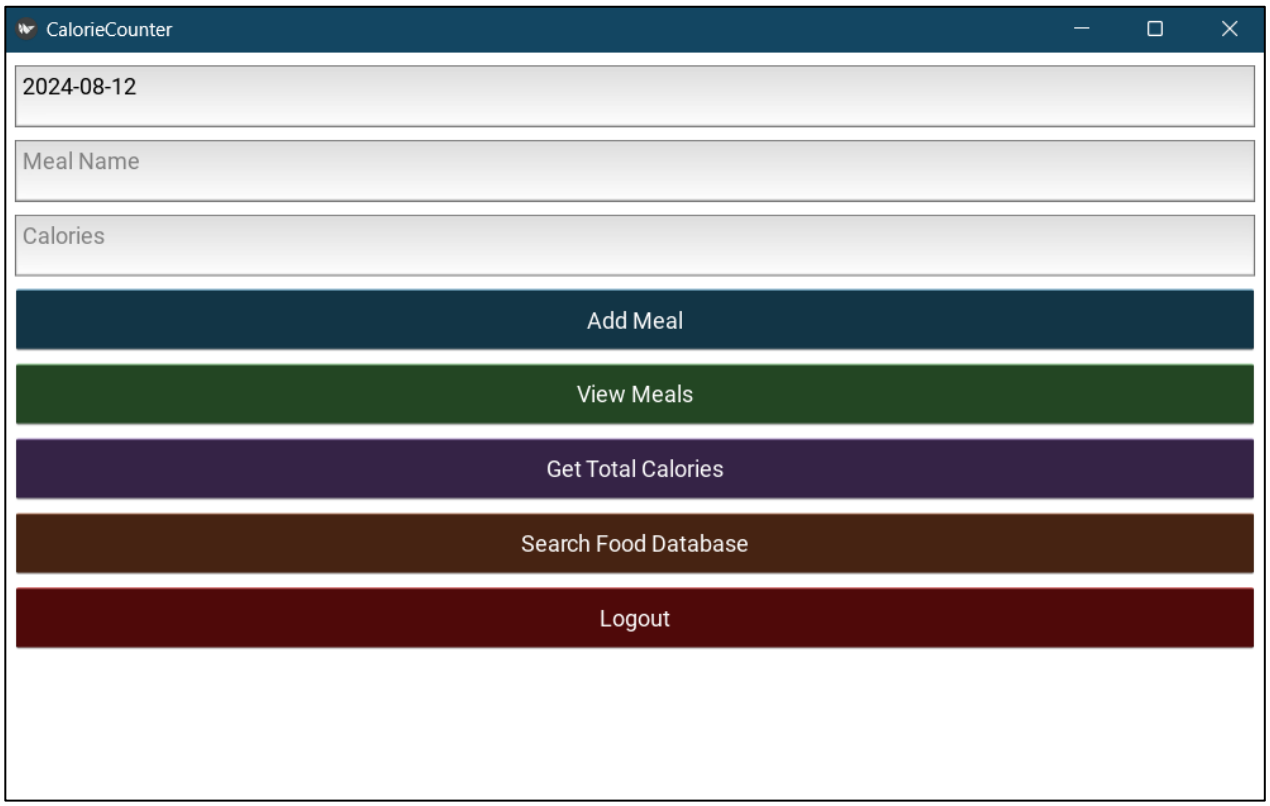
4. Custom Components

A custom `BackgroundColorBoxLayout` is used to handle the background color of the main layout. This class overrides the `__init__` method to draw a white background and binds the size and position of the layout to ensure the background is always correctly sized.

5. Transitions

The application utilizes `ScreenManager` with `FadeTransition` to manage different screens within the app. This creates smooth transitions between the login screen and the main application screen, enhancing the user experience.

The Main Screen has other components like Buttons and Labels to call different functions and perform different actions to the database. Buttons in the Main screen can be bound to specific embedded query functions to perform different tasks within the application.



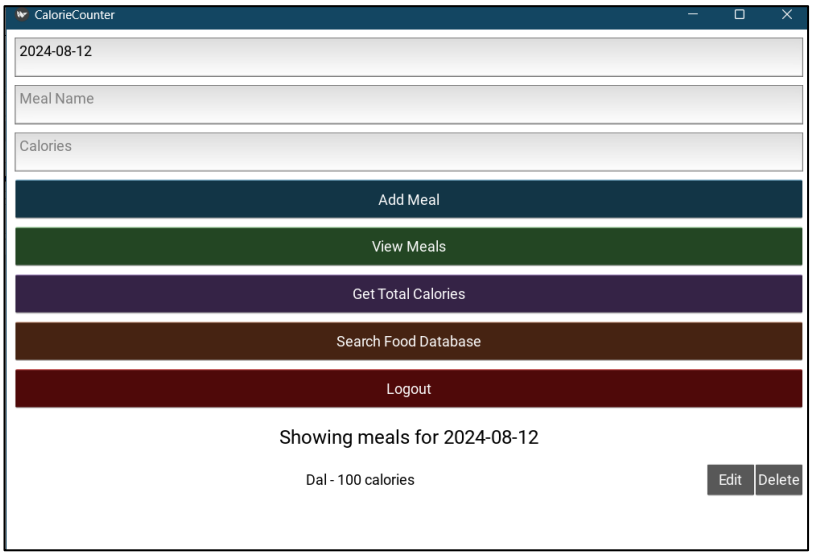
Functionalities Included

We've integrated CRUD functions for every collection, as well as complex queries such as search, and table joins for display purposes. All these functionalities are implemented using pymongo in Python.

Create Operation

We have implemented a Create function using pymongo in python. This function utilizes insert_one function of pymongo to execute the operation.

For this purpose, we have created a form to take text input from the user and store the input data within a dictionary in python. This dictionary variable is later inserted into the database using the insert_one function within the pymongo package. Shown below is the process of adding a new meal into the Meals collection. Upon filling the form and pressing 'Add Meal', the data is added to the database. In the screenshot below, a meal named "Dal" is added.

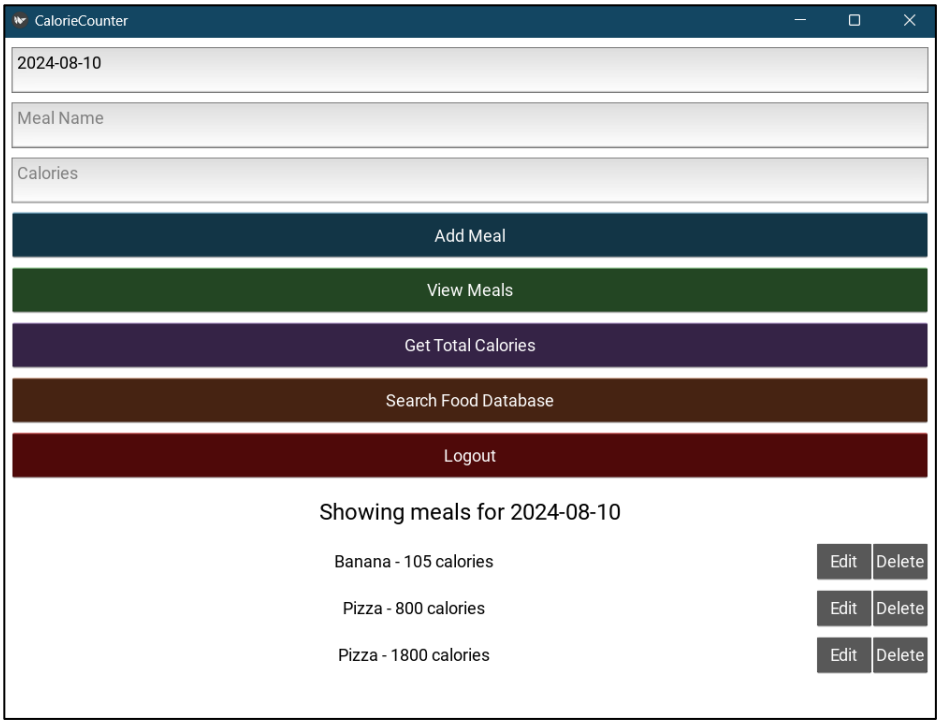


Retrieve Operation

Retrieve operation is performed as a display functionality in the applications, each collection has its own display functions, which retrieve the data from the database and displays it on the respective screen.

This operation is performed in pymongo using the find function, this function connects to the database and retrieves all documents from respective collection. The retrieved data is formatted as required before displaying.

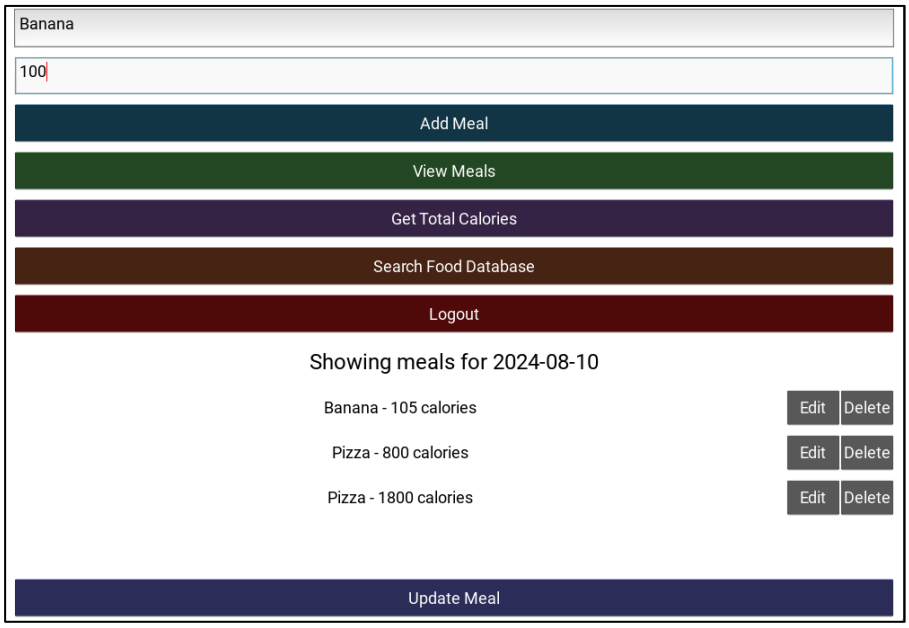
Shown below is the display option for Meals. The application retrieves all the Meals within the Meals table and displays them in a structured format.



Update Operation

All the collections allow users to modify the existing data from the database with Update operation. The update functionality takes input like title or ID and if such a document exists in the database, it displays a form with pre-existing data to update the details of the specified document. If no item with the input title or id exists, an error message is printed.

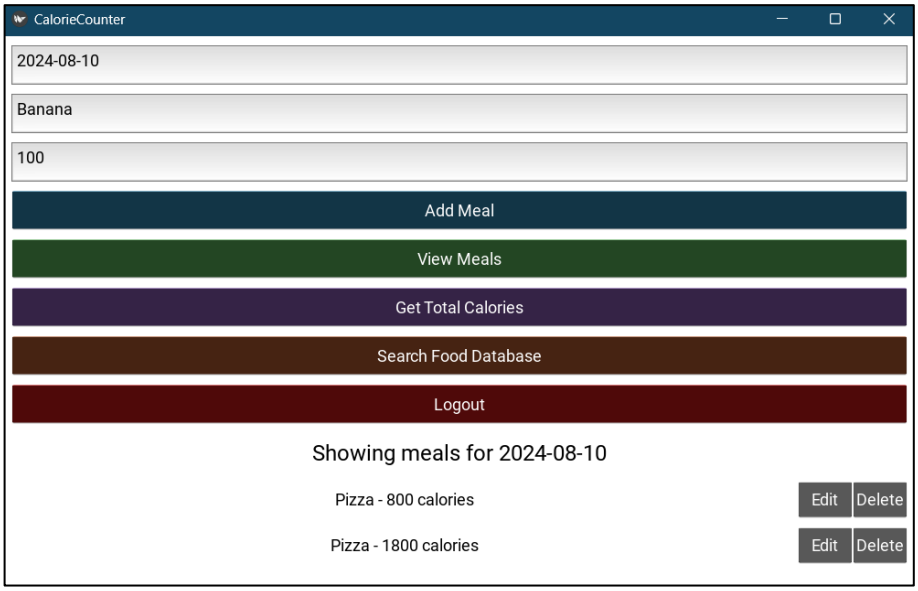
In the figure below, we have updated the newly created Meal to Update its Calorie Count. The same change is reflected in the MongoDB.



Delete Operation

Delete functionality is implemented as follows. The user is able to delete the meal using the delete button built in Kivy. Upon clicking the button, it is deleted from the database using the delete_one function of pymongo. Similar CRUD functionality is implemented for the collection's users, meals and the parent food-database.

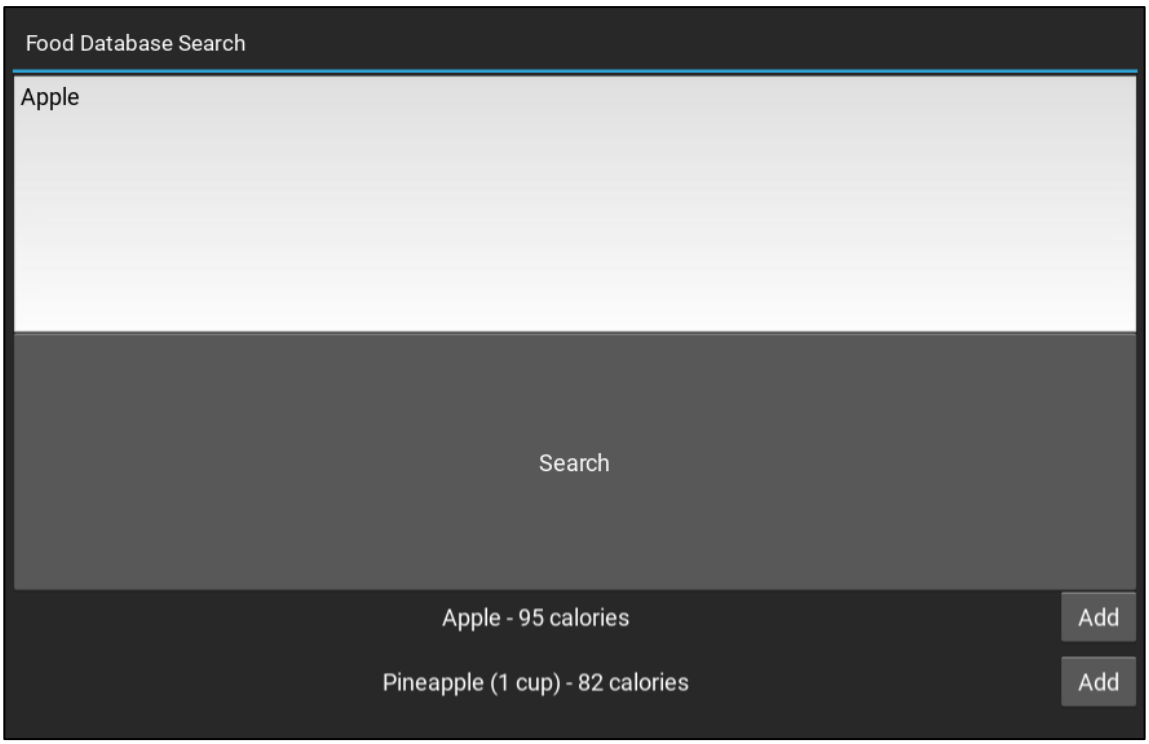
After deleting “Pizza” from Meals, the same is reflected in MongoDB.



Search Operation

Search functionality in Calorie Counter allows users to search for and display data based on specific input. This functionality takes text input from users to find the food item in given collection accordingly.

Search function utilizes the find_one function in pymongo by taking the given keyword as attribute. This operation is made to be case insensitive and to accept and execute functionality with even partial keywords. Entering apple may also display 'Pineapple' through the food-database collection



Source Code

Login Screen

```
# Login screen class
class LoginScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Main layout for the screen
        main_layout = BoxLayout(orientation='vertical')

        # Create a white background using a canvas
        with self.canvas.before:
            Color(1, 1, 1, 1)  # White color
            self.rect = Rectangle(size=self.size, pos=self.pos)

        # Background image for the login screen
        self.bind(size=self._update_rect, pos=self._update_rect)

        # Load and display the background image
        background_image = Image(source='background.jpg', allow_stretch=False, keep_ratio=True)

        # Form layout for username and password inputs
        form_layout = BoxLayout(orientation='vertical', padding=[20, 20, 20, 50], spacing=10)

        # Welcome message at the top of the form
        welcome_message = Label(
            text='[color=#228B22]Welcome to our health tracking app![/color]\n[color=#4169E1]Please login/register to continue![/color]',
            size_hint_y=None, height=60, font_size=24,  # Increased font size
            halign='center',
            markup=True  # Enable markup for colored text
        )

        form_layout.add_widget(welcome_message)
```

```
# Text inputs for username and password
self.username_input = TextInput(hint_text='Username', size_hint_y=None, height=40)
self.password_input = TextInput(hint_text='Password', password=True, size_hint_y=None, height=40)

# Buttons for login and registration
login_button = Button(
    text='Login', on_press=self.login, size_hint_y=None, height=50,  # Increased height for better appearance
    background_color=(0.2, 0.6, 0.8, 1), color=(1, 1, 1, 1),
    font_size=20  # Increased font size
)
register_button = Button(
    text='Register', on_press=self.register, size_hint_y=None, height=50,  # Increased height for better appearance
    background_color=(0.3, 0.7, 0.4, 1), color=(1, 1, 1, 1),
    font_size=20  # Increased font size
)

# Label for displaying messages (e.g., login errors)
self.message_label = Label(text='', color=(1, 0, 0, 1), font_size=16)

# Add widgets to the form layout
form_layout.add_widget(self.username_input)
form_layout.add_widget(self.password_input)
form_layout.add_widget(login_button)
form_layout.add_widget(register_button)
form_layout.add_widget(self.message_label)

# Add the background image and the form layout to the main layout
main_layout.add_widget(background_image)
main_layout.add_widget(form_layout)
```

Main Screen -

```
class MainScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.user_id = None

        # Main layout with vertical orientation, padding, and spacing
        self.main_layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # White background using a canvas
        with self.canvas.before:
            Color(1, 1, 1, 1)
            self.rect = Rectangle(size=self.size, pos=self.pos)

        # Update the background rectangle when the window is resized
        self.bind(size=self._update_rect, pos=self._update_rect)

        # Create input fields for date, meal name, and calories
        self.date_input = TextInput(hint_text='Date (YYYY-MM-DD)', text=datetime.now().strftime('%Y-%m-%d'))
        self.meal_input = TextInput(hint_text='Meal Name')
        self.calories_input = TextInput(hint_text='Calories')

        # Create buttons for adding, viewing, and managing meals, and customize their appearance
        add_button = Button(
            text='Add Meal', on_press=self.add_meal,
            background_color=(0.2, 0.6, 0.8, 1), color=(1, 1, 1, 1), font_size=18
        )
        view_button = Button(
            text='View Meals', on_press=self.view_meals,
            background_color=(0.4, 0.8, 0.4, 1), color=(1, 1, 1, 1), font_size=18
        )
        total_button = Button(
            text='Get Total Calories', on_press=self.get_total,
            background_color=(0.6, 0.4, 0.8, 1), color=(1, 1, 1, 1), font_size=18
```

```
    )
    search_db_button = Button(
        text='Search Food Database', on_press=self.open_food_db_search,
        background_color=(0.8, 0.4, 0.2, 1), color=(1, 1, 1, 1), font_size=18
    )
    logout_button = Button(
        text='Logout', on_press=self.logout,
        background_color=(0.9, 0.1, 0.1, 1), color=(1, 1, 1, 1), font_size=18
    )

    # Label for displaying results
    self.result_label = Label(text='', color=(0, 0, 0, 1), font_size=16)

    # Add all widgets (input fields, buttons, label) to the main layout
    self.main_layout.add_widget(self.date_input)
    self.main_layout.add_widget(self.meal_input)
    self.main_layout.add_widget(self.calories_input)
    self.main_layout.add_widget(add_button)
    self.main_layout.add_widget(view_button)
    self.main_layout.add_widget(total_button)
    self.main_layout.add_widget(search_db_button)
    self.main_layout.add_widget(logout_button)
    self.main_layout.add_widget(self.result_label)

    # Meals layout with scrollable view
    self.meals_layout = GridLayout(cols=1, spacing=10, size_hint_y=None)
    self.meals_layout.bind(minimum_height=self.meals_layout.setter('height'))

    scroll_view = ScrollView(size_hint=(1, None), size=(400, 200))
    scroll_view.add_widget(self.meals_layout)
    self.main_layout.add_widget(scroll_view)

    # Add the main layout to the screen
    self.add_widget(self.main_layout)
```

Create Meal function -

```
def add_meal(self, instance):
    date = self.date_input.text
    name = self.meal_input.text
    calories = self.calories_input.text
    self.result_label.font_size = 24

    # Validate input before adding the meal
    valid, message = self.validate_input(date, name, calories)
    if not valid:
        self.result_label.text = message
        return

    user_id = self.user_id
    calories = int(calories)

    meal_id = add_meal(user_id, name, calories, date)
    self.result_label.text = f"Meal added successfully!"

    self.date_input.text = datetime.now().strftime('%Y-%m-%d')
    self.meal_input.text = ''
    self.calories_input.text = ''
```

Retrieve Meal function -

```
def view_meals(self, instance):
    self.meals_layout.clear_widgets()
    date = self.date_input.text
    self.result_label.font_size = 24
    try:
        datetime.strptime(date, '%Y-%m-%d')
    except ValueError:
        self.result_label.text = "Invalid date format. Use YYYY-MM-DD."
        return

    meals = get_meals(self.user_id, date)
    if meals:
        for meal in meals:
            meal_layout = BoxLayout(orientation='horizontal', size_hint_y=None, height=40)
            meal_label = Label(text=f"{meal['name']} - {meal['calories']} calories", color=(0, 0, 0, 1))
            edit_button = Button(text='Edit', size_hint_x=None, width=60)
            edit_button.bind(on_press=lambda x, meal=meal: self.edit_meal(meal))
            delete_button = Button(text='Delete', size_hint_x=None, width=60)
            delete_button.bind(on_press=lambda x, id=meal['_id']: self.delete_meal(id))

            meal_layout.add_widget(meal_label)
            meal_layout.add_widget(edit_button)
            meal_layout.add_widget(delete_button)
            self.meals_layout.add_widget(meal_layout)

        self.result_label.text = f"Showing meals for {date}"
    else:
        self.result_label.text = 'No meals found for this date'
```

Update function -

```
# Edit a selected meal
def edit_meal(self, meal):
    self.meal_input.text = meal['name']
    self.calories_input.text = str(meal['calories'])

    update_button = Button(
        text='Update Meal', on_press=lambda x: self.update_meal(meal['_id']),
        background_color=(0.5, 0.5, 1, 1), color=(1, 1, 1, 1), font_size=18
    )
    self.main_layout.add_widget(update_button)

# Update the meal in the database
def update_meal(self, meal_id):
    name = self.meal_input.text
    calories = self.calories_input.text

    is_valid, error_message = self.validate_input(self.date_input.text, name, calories)
    if not is_valid:
        self.result_label.text = error_message
        return

    update_meal(meal_id, name, int(calories))
    self.result_label.text = f'Updated meal with ID: {meal_id}'
    self.meal_input.text = ''
    self.calories_input.text = ''
    self.view_meals(None)
    self.main_layout.remove_widget(self.main_layout.children[0])
```

Delete function -

```
# Function to delete a meal entry
def delete_meal(meal_id):
    result = meals_collection.delete_one({"_id": ObjectId(meal_id)})
    return result.deleted_count
```

```
    def delete_meal(self, meal_id):
        delete_meal(meal_id)
        self.result_label.text = f'Deleted meal with ID: {meal_id}'
        self.view_meals(None)
```

Search Food Database -

```
    # Search the food database for matching items
    def search_food_db(self, instance):
        self.results_layout.clear_widgets()
        search_term = self.search_input.text.strip()

        if search_term:
            results = food_db_collection.find({"name": {"$regex": search_term, "$options": "i"}})#The "i" makes the search case-insensitive.
            for result in results:
                result_layout = BoxLayout(orientation='horizontal', size_hint_y=None, height=40)
                result_label = Label(text=f"{result['name']} ({result['calories']} calories)", font_size=16, color=(0.2, 0.2, 0.2, 1))
                result_layout.add_widget(result_label)
                self.results_layout.add_widget(result_layout)
                self.results_layout.add_widget(result_layout)
        else:
            result_layout = BoxLayout(orientation='horizontal', size_hint_y=None, height=40)
            result_label = Label(text="Please enter a valid search term.", font_size=16, color=(1, 1, 1, 1))
            result_layout.add_widget(result_label)
            self.results_layout.add_widget(result_layout)
```

```
# Popup for searching the food database
class FoodDBSearchPopup(Popup):
    def __init__(self, user_id, **kwargs):
        super().__init__(**kwargs)
        self.title = "Search Food Database"
        self.size_hint = (0.8, 0.8)

        layout = BackgroundColorBoxLayout(orientation='vertical', padding=10, spacing=10)

        self.search_input = TextInput(hint_text='Enter food name', size_hint_y=None, height=40, font_size=16)
        search_button = Button(text='Search', on_press=self.search_food_db, size_hint_y=None, height=50, background_color=(0.1, 0.5, 0.7, 1)
color=(1, 1, 1, 1), font_size=16)
        self.results_layout = GridLayout(cols=1, spacing=10, size_hint_y=None)
        self.results_layout.bind(minimum_height=self.results_layout.setter('height'))

        scroll_view = ScrollView(size_hint=(1, None), size=(400, 200))
        scroll_view.add_widget(self.results_layout)

        layout.add_widget(self.search_input)
        layout.add_widget(search_button)
        layout.add_widget(scroll_view)

        self.add_widget(layout)
```

Conclusion

The development of this calorie counter application demonstrates the successful integration of modern database technology with a user-friendly graphical interface, resulting in a practical tool for personal health management. By leveraging MongoDB's flexible document-based structure and Kivy's robust GUI capabilities, we've created an application that not only serves its primary function of tracking calorie intake but also showcases the potential for scalability and feature expansion in health-focused software.

The choice of MongoDB as the database solution proved advantageous in several ways. Its schema-less nature accommodated the varying structures of meal entries and food database items without requiring rigid predefinition. This flexibility allows for future enhancements, such as incorporating more detailed nutritional information or user-specific data, without necessitating significant database restructuring. Moreover, MongoDB's efficient querying capabilities facilitated quick searches within the food database, enhancing the user experience by providing rapid access to common food items.

The implementation of CRUD (Create, Read, Update, Delete) operations within the application demonstrates a comprehensive approach to data management. Users can easily add new meals, view their daily intake, modify entries as needed, and remove incorrect data. This full spectrum of data manipulation options ensures that users have complete control over their recorded information, promoting accuracy and engagement with the application. The inclusion of a pre-populated food database serves as a valuable feature, streamlining the process of logging meals. By allowing users to quickly search for and add common foods, the application reduces the friction often associated with manual data entry, potentially encouraging more consistent use of the calorie tracking feature.

While the current implementation serves its purpose effectively, there are numerous avenues for future enhancements. These could include implementing user authentication for personalized experiences, incorporating data visualization for long-term trend analysis, integrating with wearable devices for automated activity tracking, or expanding the nutritional tracking beyond just calories to include macronutrients and micronutrients.

In conclusion, this calorie counter application not only fulfills its primary objective of helping users track their daily calorie intake but also serves as a testament to the effective use of modern software development tools and practices. It demonstrates how choosing appropriate technologies and implementing them thoughtfully can result in a functional, user-friendly application with significant potential for future growth and improvement.