Nutrition App

A nutrition tracking app that allows users to log their meals and track their daily calorie intake. The app will maintain the following details on the server side:

- Id - the unique identifier for each meal. Integer value greater than zero.
- Name - The name of the meal. A string of characters.
- Type - The type of meal. A string of characters. Eg. "breakfast", "lunch", "dinner", "snack", etc.
- Calories - The number of calories in the meal. An integer value.
- Date - The date the meal was consumed. A string in the format "YYYY-MM-DD".
- Notes - Additional notes or comments about the meal. A string of characters.

The application should provide the following features:

- Record Section (separate screen)
  a. **(2p)**(1p) View all the meals in the system using **GET /all** calls. The user can retrieve all the meals. If offline, the app will display an offline message and a way to retry the connection and the call. Once retrieved, even online, the meal details should always be available, no other server calls are needed.
  b. **(1p)**(0.5p) Record a new meal using **POST /meal** call by specifying all the meal details. Available online and offline.
- Manage Section (separate screen, available only online)
  a. **(1p)**(0.5p) View all the available meal types in the system. Using **GET /types** calls, the user will retrieve all the meal type names.
  b. **(1p)**(0.5p) View all the available meals for the selected type in a list. Using **GET /meals** calls, the user will retrieve all the available meals of the selected type.
  c. **(1p)**(0.5p) Delete a meal, the user will be able to delete the selected meal. Using **DELETE /meal** call, by sending the meal id.
- Reports Section (separate screen, available only online)
  a. **(1p)**(0.5p) View the top 10 meals in a list containing the meal name, type, calories, and date. Using the same **GET /all** call. The list should present the result in descending order by their calorie value. Note that the list received from the server is not ordered.
  b. (1p) View the total number of calories consumed for each meal type. Using the same **GET /all** call. The list should display the total number of calories consumed for each meal type.
  c. (1p) View the meal history for a selected date range. Using the same **GET /all** call the user should be able to filter the values by specifying the start and end date. The list should present the meal name, type, calories consumed, and date. The list should be ordered by the meal date.
- Profile Section (separate screen)
  a. (0.5p) Record the user's personal details, like name, age, height, and weight in the application settings. Persisted to survive app restarts.
  b. (0.5p) View the user's personal details on a separate screen/dialog.
  c. (0.5p) Edit the user's personal details and update the details in the application settings.
- Notifications (should work on all screens)
  (1p) On the server side, once a new meal is added to the system, the server will send a message using a WebSocket channel to all the connected clients/applications with the new meal object. Each application that is connected will display the received meal name, type, and calorie values in human form using an in-app "notification" (like a snackbar or toast or a dialog on the screen).
- Developer Details
  (0.5p) On all server/DB operations, a progress indicator will be displayed.
  (0.5p) On all server interactions, if an error message is received, the app should display the error message using a toast or snackbar and log the initial error message. A log message should be recorded on all interactions (server or DB calls).