Project Kitchen Buddy
Final project

The final project will expand on assignment 2. The deadline for the final project will be **Sunday, June 22 2025**. Schedule of project presentations will be shared in last week of June.

This project can be done in groups, but the scope of the project will grow when more people are involved. A pair will also implement a grocery list extension. **If you plan to be in a team, you must inform me.**

Better ingredient handling (everyone)

The application is extended with additional functionality and ingredient data.

- Persistence: the application's data persists accross execution.
- Ripeness: fresh ingredients have a ripeness or maturity status (e.g., green, ripe/mature, advanced, too ripe). This maturity status can be edited, and the date when it was edited is stored (something ripe a week ago might not be good anymore!).
- Frozen: fresh ingredients can be frozen. This also extend their expiration date to be at least 6 months.
- Open: some ingredients last only a short time after being opened (e.g., a yogurt) When an ingredient is set as open, their expiry date can be changed to account for this.
- Barcode scanning: the application uses Expo's BarCodeScanner API to read barcodes. It can then query OpenFoodFacts, and if succesful, it retrieves data about the item automatically.
- Brand: some items can have brands, in addition to names.

Additional queries are implemented:

- Items that have a ripeness status need to be checked regularly. If the last check was more than 3 days ago, they are added.
- Items that are ripe, and open items, are added to the "expiring soon" query; items that are frozen are removed from it (unless their new expiry date is coming up)

Grocery list (additional work for a pair)

The application is extended with:

- Quantities: ingredients support quantities, either as numbers (e.g., a dozen of eggs), or as fractions of the initial content (e.g., the milk or jam is half empty). The quantity can be decreased when an ingredient is used. Note that adding new ingredients of the same type is done differently (as they likely would have different expiration dates).

An additional tab is added to the application, to manage a grocery list. This tab contains the list of ingredients that needs to be bought. Items can be added to the grocery list in in the following ways:

- there is an additional query that shows ingredients that have either a low quantity, or that are empty. Each of these has an "Add to groceries" button. These items are added to the list, but without the data that is not relevant (e.g. the expiration date depends on the actual item to buy).
- the "grocery list" panel has a "quick-add" entry, where grocery list items can be added via text only (just adding "pasta").

When in the shop, items can be bought, and are removed from the list. However, you wouldn't want to the complete data entry while in the shop. Instead, you would do that at home:

- A "recently bought" query is added to allow edition of such items.
- In addition, if an item is "re-bought", the application can propose an expiry date based on the previously bought item. For instance, if the previous item was bought with an expiration date of two weeks from then, the same time interval is proposed, but can be edited.

Finally, shops can be defined by their location, and their type (general, butcher's shop, etc ...). The application then automatically switches to the grocery list tab when near a shop rather than at home. It is also able to sort the priority list based on location, (e.g., don't show the fish if we are in a butcher's shop).

Important details for evaluation

Functional programming Your project should, as much as possible, follow functional programming principes. Functions should be small, do one thing only, and should not modify their inputs.

- Instead of changing an input, return an updated version of your input as the output of the function. Remember that the spread operators in Javascript can be used to make copies of objects and lists, which will help you for this.
- An example of doing one thing only would be that instead of directly printing to the console, you should define functions that format the data in the correct way, and define separate functions that actually print on the console.

Typescript Your project should, as much as possible, use type annotations. At the very least, any data structure you define should have a type definition, and the functions you define should have type annotations for their arguments and return types.

Submission

- The submission should be the snack link to the project
- The readme.md file should include
    - list all the components that the application defines, along with their props, state, and the types of props and states
    - also display the components in a tree that matches the way they are organized in the UI
- Show how control flow affects the application. Annotate on the tree (possibly with different colors):
    - how callbacks are passed down from parents to child's components
    - how these callbacks change the state of the applications
    - how changes of state change the tree of the application