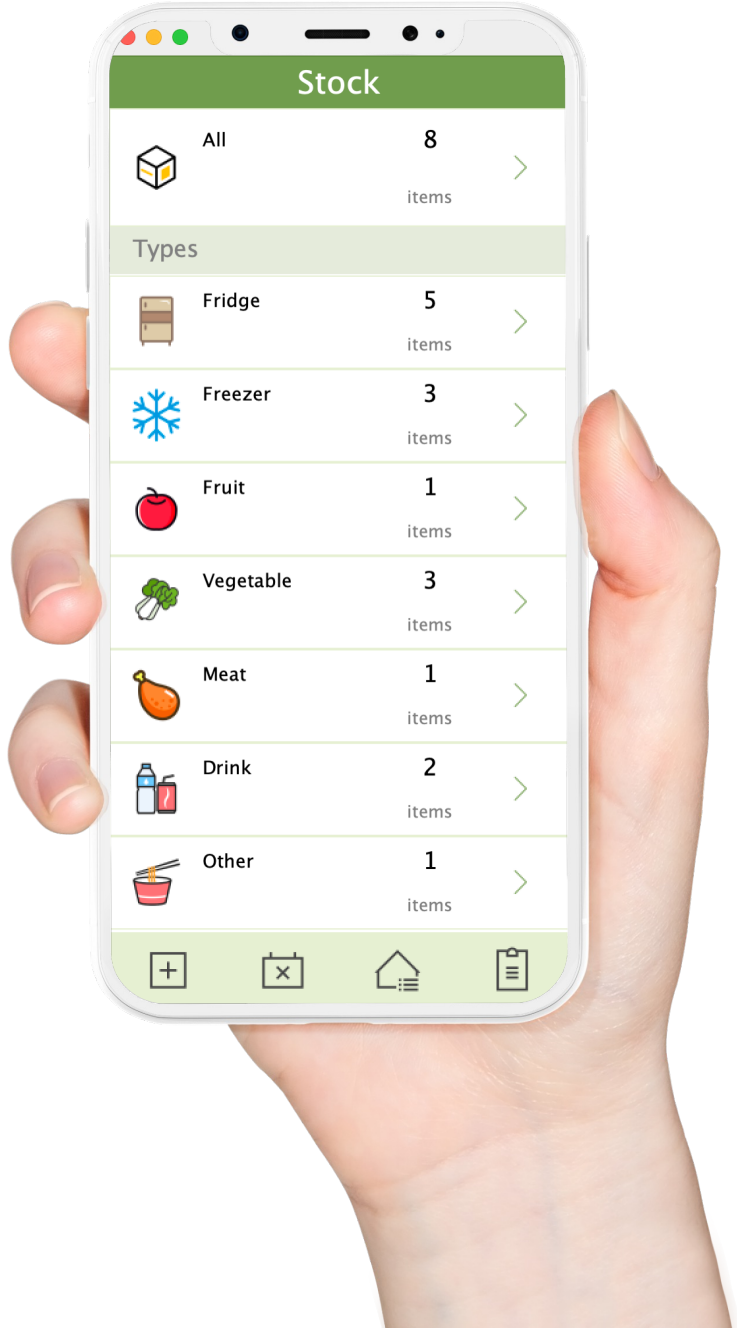
Xiao Xu, Cuichan Wu, Yuyan Lei

The App

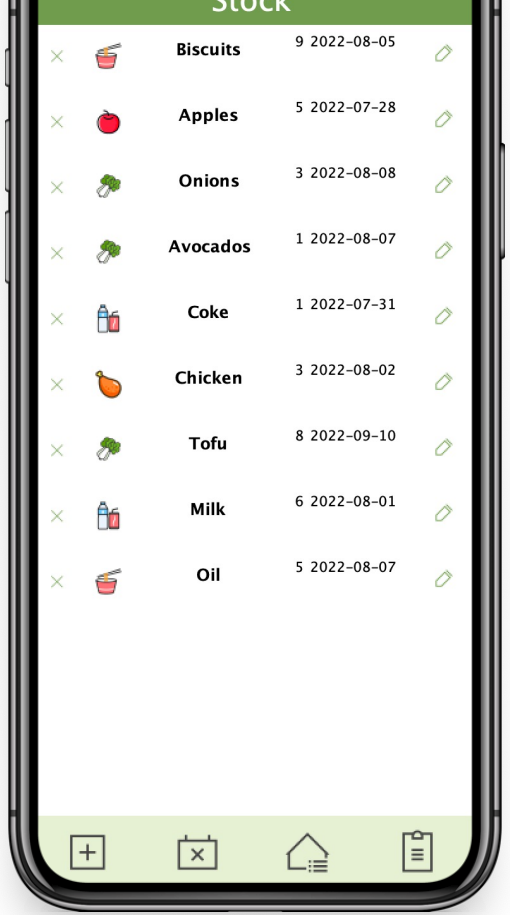
My Fridge

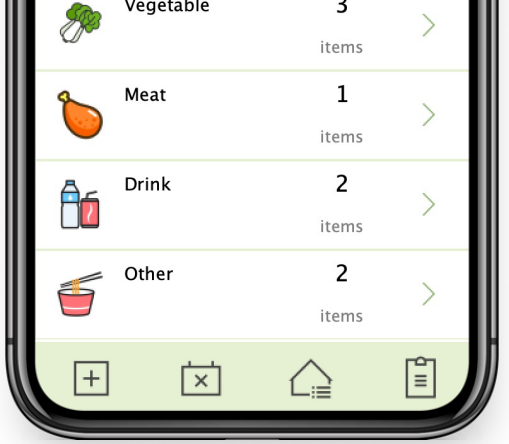


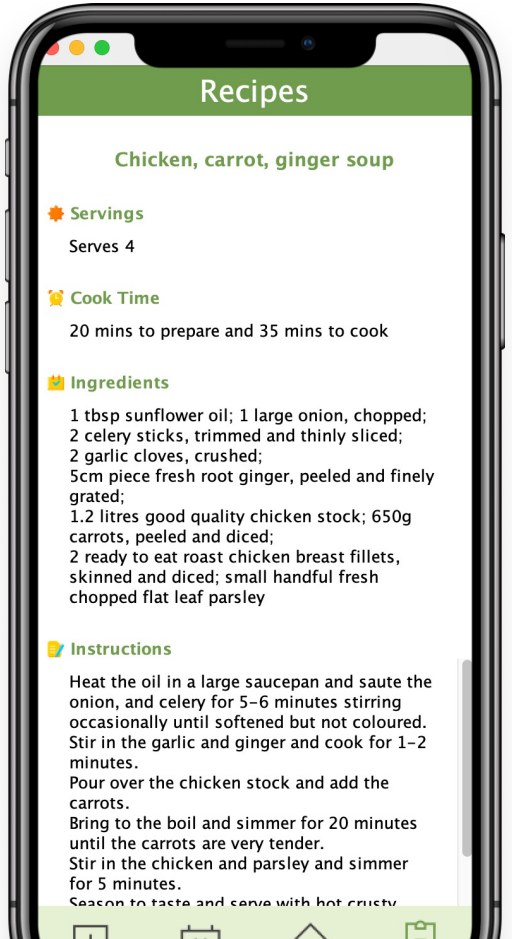
Outline

1. Main idea of our project (purposes & features)
2. High level design with diagrams
3. Each part of the project (demo & highlights in codes)
4. Q&A

Let's Go!







Why -- Everyday situations

- What do I have in my fridge?
- Where did I put the box of eggs?
- Is any food about to expire?
- When do I need to buy the milk again?
- What can I cook today?

What to do...

-> Track the stock

-> Get alerts (expiration & low stock)

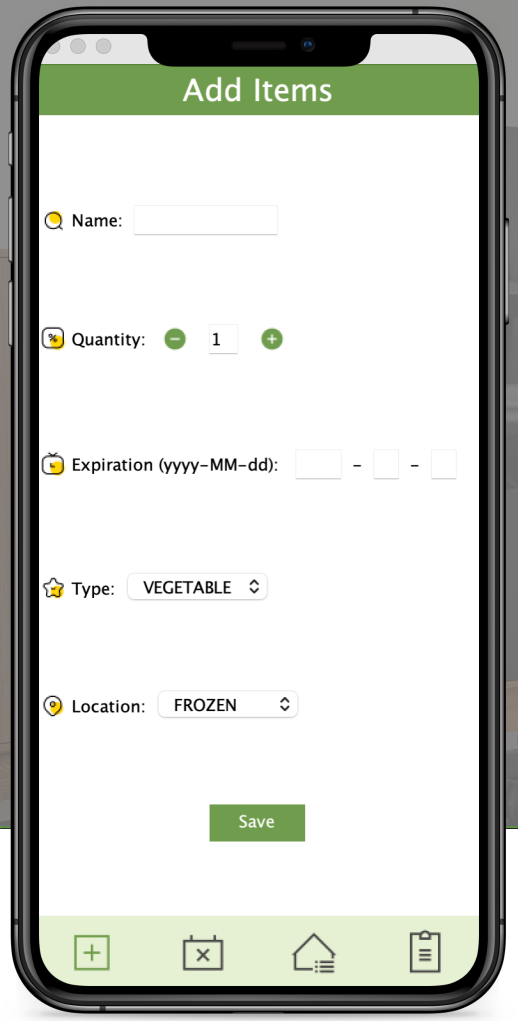
-> Generates recipes

Features

Manage and Organize Your Fridge

Manage Your Fridge

Add
edit
or remove
any items



Features

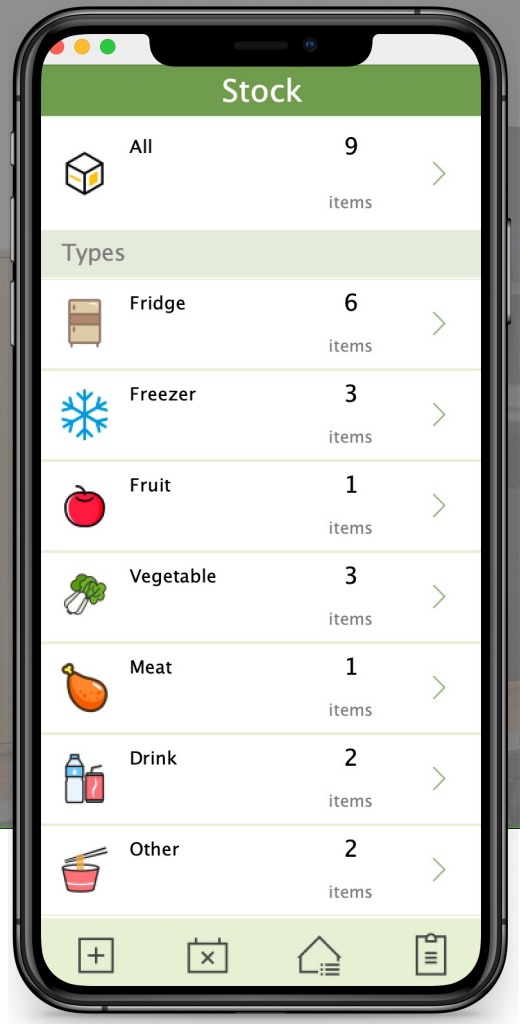
Manage and Organize Your Fridge

Manage Your Fridge

Add
edit
or remove
any items

Track the Stock

Know what you have
in your fridge
with one single click



Features

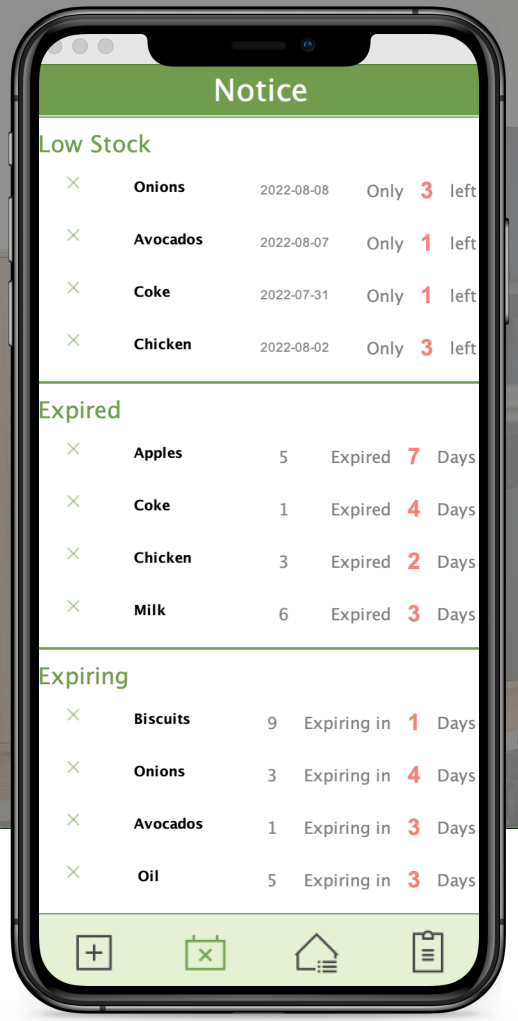
Manage and Organize Your Fridge

Manage Your Fridge

Add
edit
or remove
any items

Track the Stock

Know what you have
in your fridge
with one single click



Notify

Get alerts
when your food is running low,
about to expire, or expired.

Features

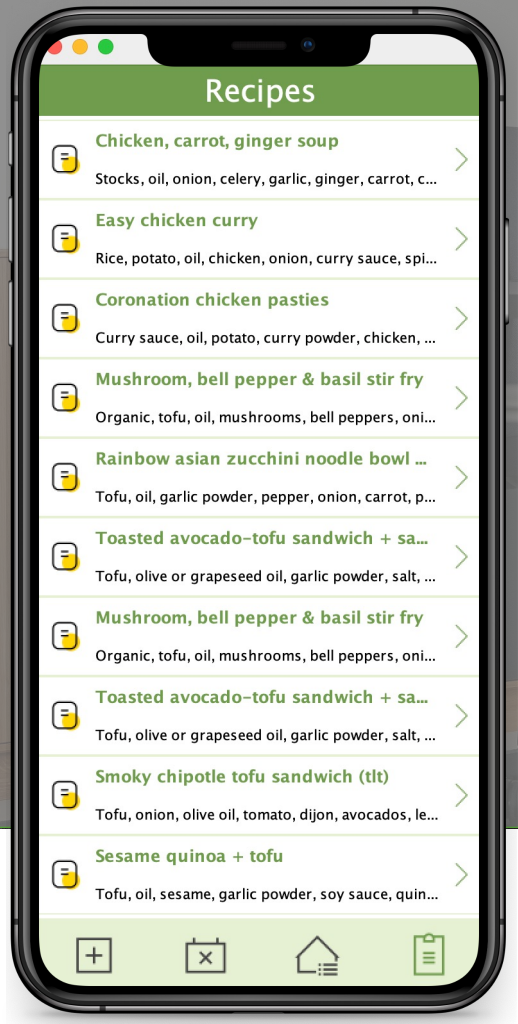
Manage and Organize Your Fridge

Manage Your Fridge

Add
edit
or remove
any items

Track the Stock

Know what you have
in your fridge
with one single click



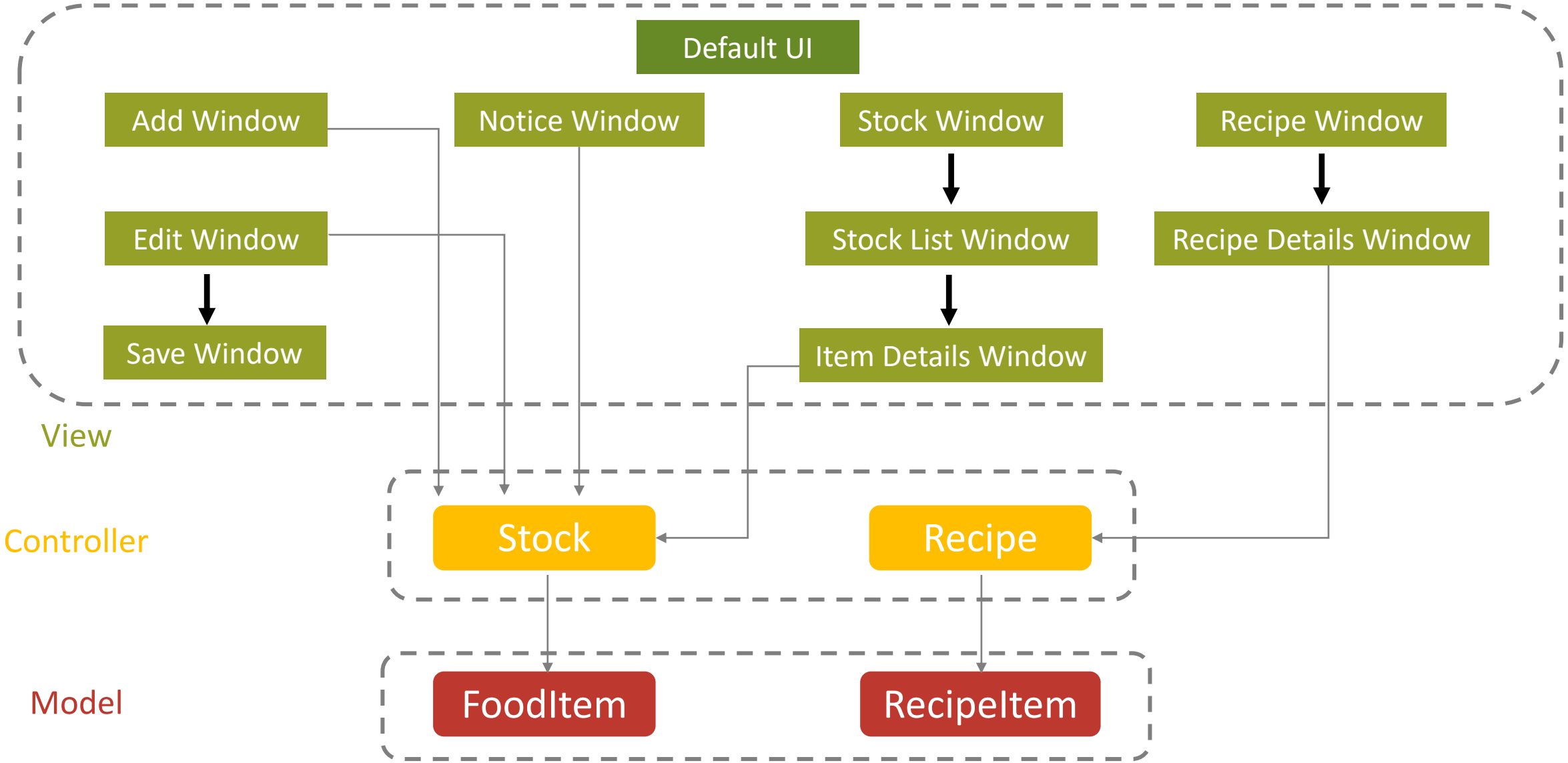
Show Recipes

Automatically generate recipes
based on current ingredients
in your fridge

Notify

Get alerts
when your food is running low,
about to expire, or expired.

High Level Design



Group Works

Xiao Xu

- Add Window
- Edit Window
- Save Window

Cuichan Wu

- Default UI
- Notice Window
- Stock Window

Yuyan Lei

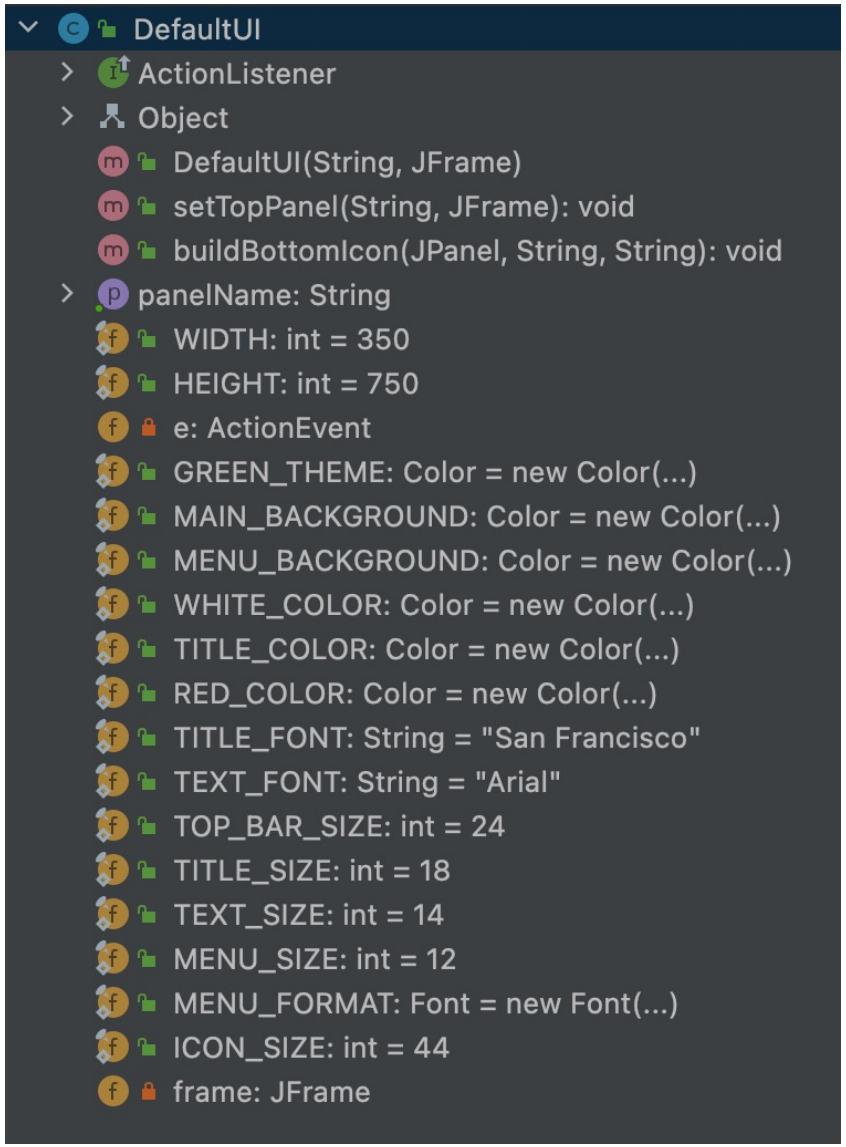
- Stock List Window
- Item Details Window
- Recipe Window
- Recipe Details Window
- Stock
- Recipe
- FoodItem
- RecipeItem

Default UI

UI & Menu Button

Default UI

- **Variables**
  - Frame size
  - All background color
  - All Text size and font
- **Default UI**:
  - Fixed frame
  - Top panel shows each feature
  - Bottom menu buttons
    - Color changed by different features

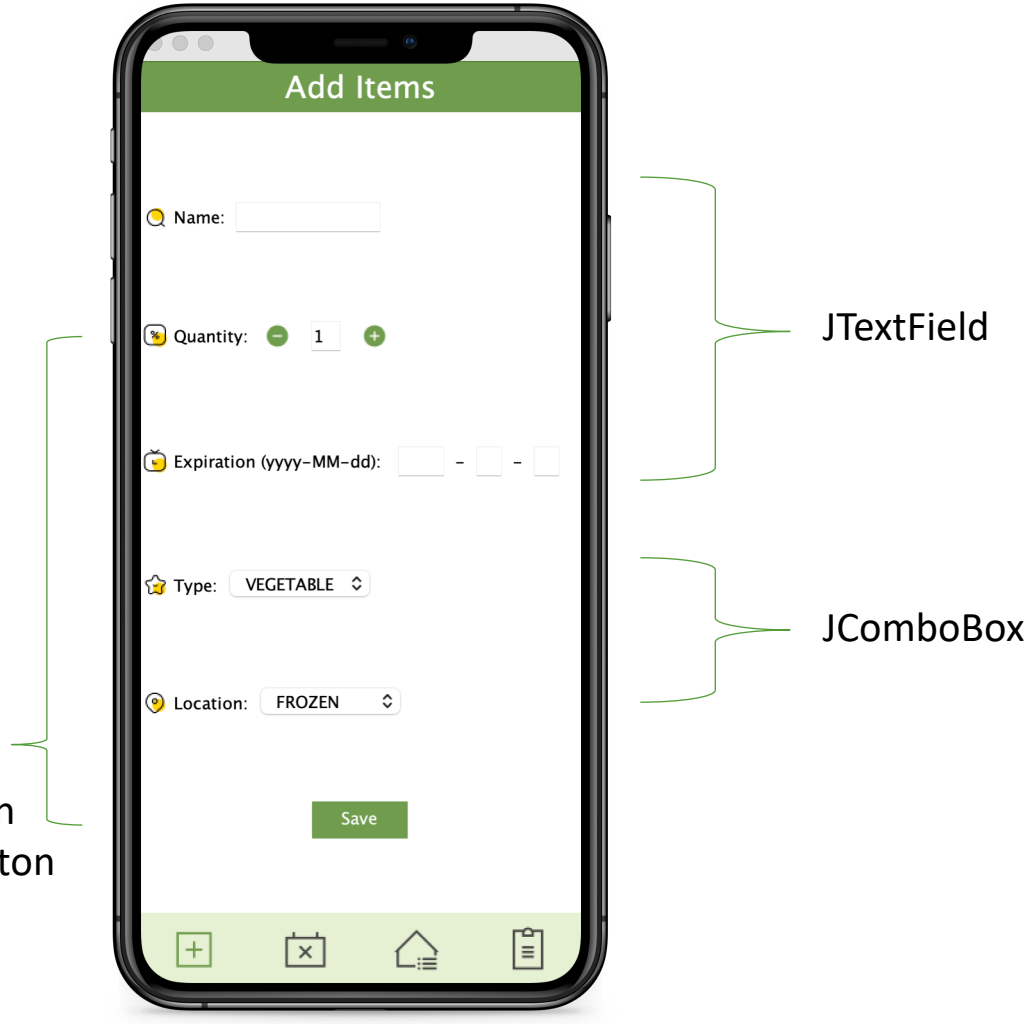


Feature 1 – Add to the stock

Add or Edit an Item

Add Window

- **Error check**
  - 1. **Name** -- cannot be empty
  - 2. **Quantity** -- should be a positive integer number (excludes 0)
  - 3. **Expiration date --** should be within 2022/1/1 – 2032/12/31



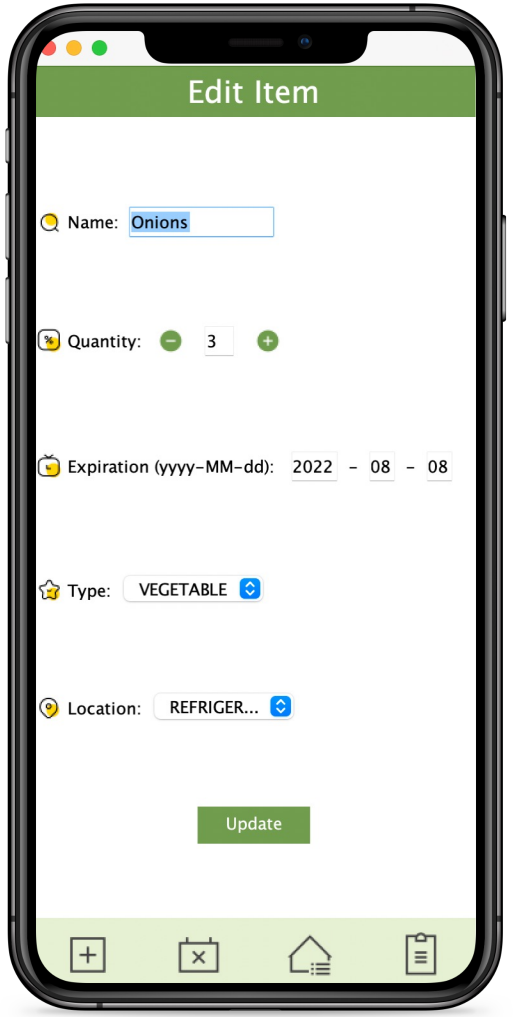
Edit Window

Similar as the Add Window, except the **JTextFileds have default value**.

```
// No error situation
if (isValidate) {
    try {
        Stock newItem = new Stock();
        theName = String.valueOf(name.getText());
        theQuantity = Integer.parseInt(quantity.getText());
        theExpiration = new Date((Integer.parseInt(expirationYear.getText()) - 1901),
                (Integer.parseInt(expirationMonth.getText()) + 11),
                Integer.parseInt(expirationDay.getText()));
        theType = FoodItem.FoodType.valueOf(String.valueOf(typeOption.getSelectedItem()));
        theLocation = FoodItem.PlaceLocation.valueOf(String.valueOf(locationOption.getSelectedItem()));
        newItem.updateItem(theItem, theName, theQuantity, theExpiration, theType, theLocation);
        SaveWindow aNewWindow = new SaveWindow();
    } catch (ParseException | IOException ex) {
        throw new RuntimeException(ex);
```

3 buttons:

- Quantity add button
- Quantity minus button
- Update button

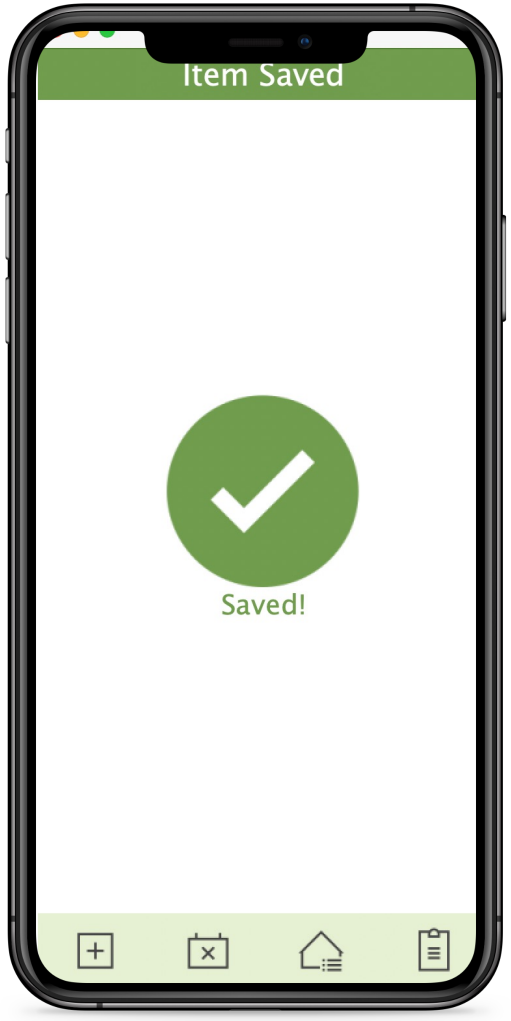


JTextField

JComboBox

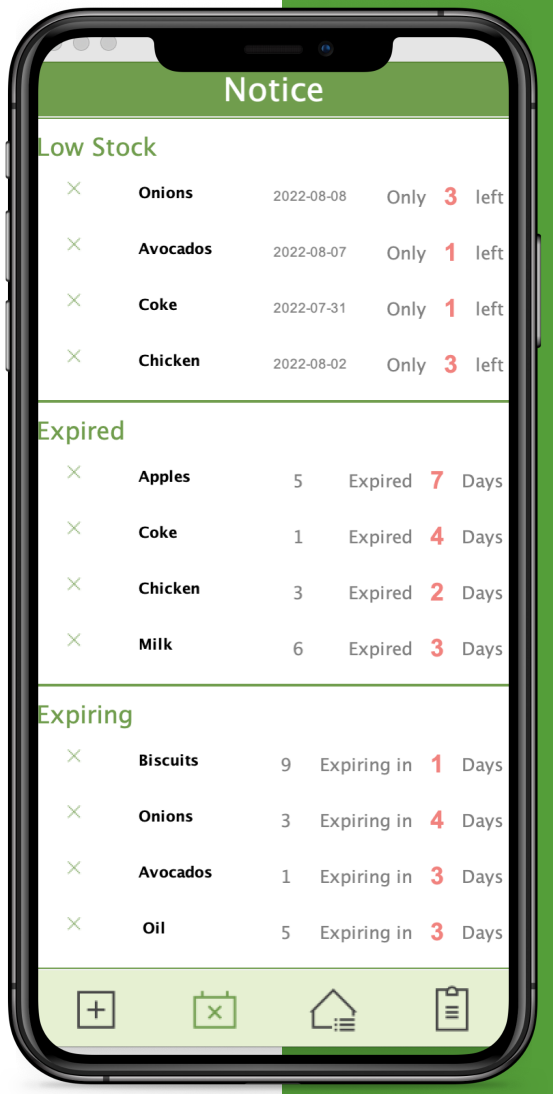
Save Window

If it’s error free, then your item is saved to stock.



Feature 2 -- Notice

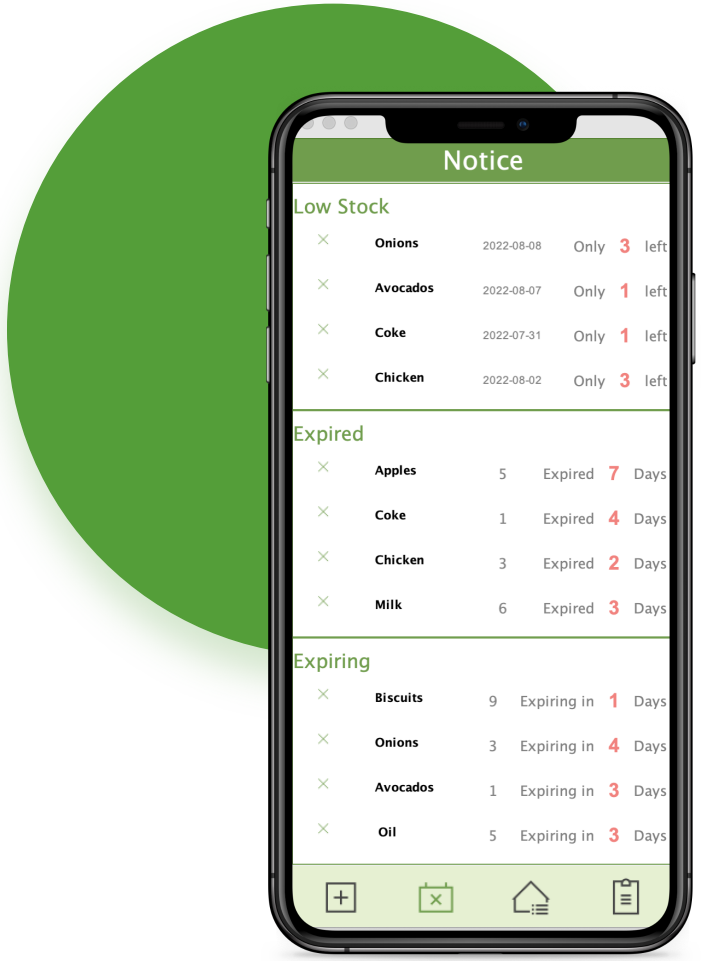
Get Notice about items In your Fridge



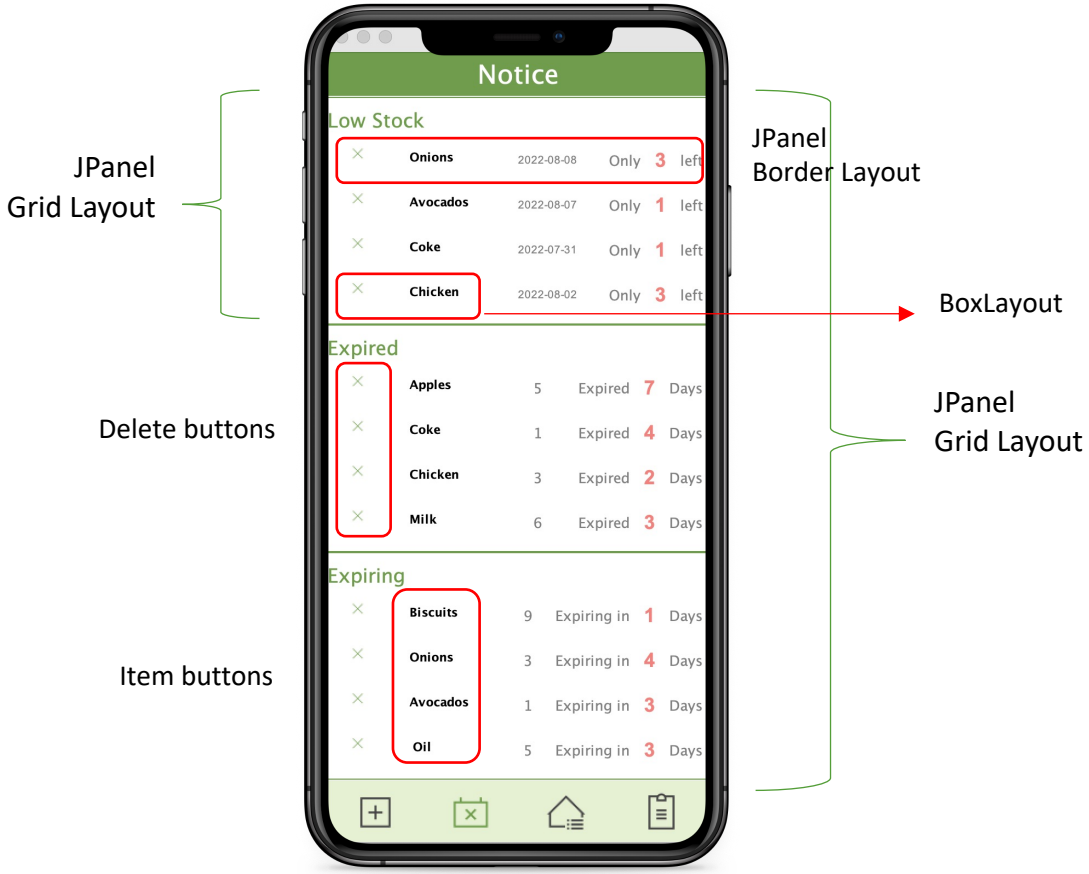
Notice Window

Get notifications for three different states:

- **Low stock items**
- **Expired items**
- **Expiring items**

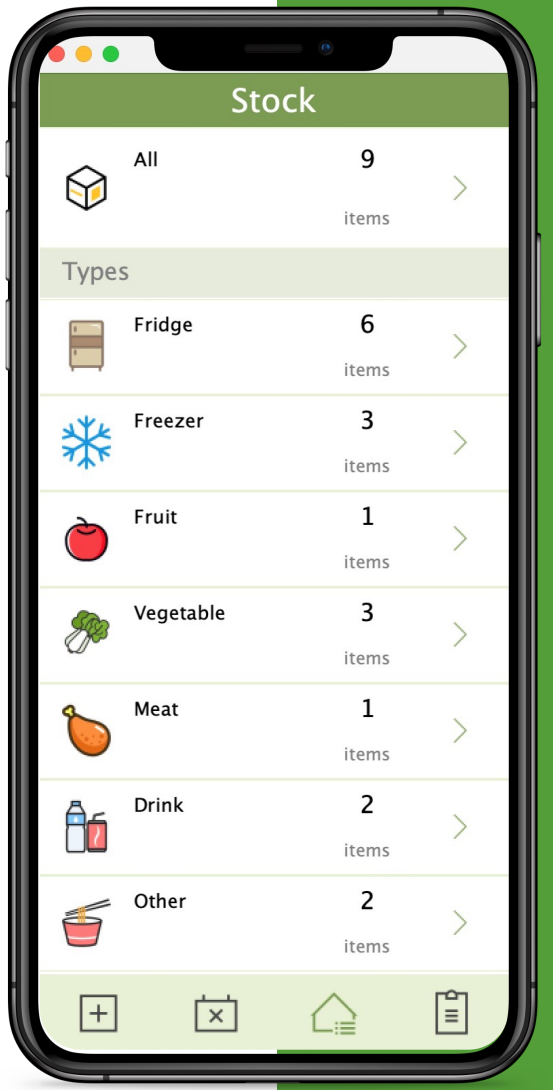


Notice Window



Feature 3 -- Stock

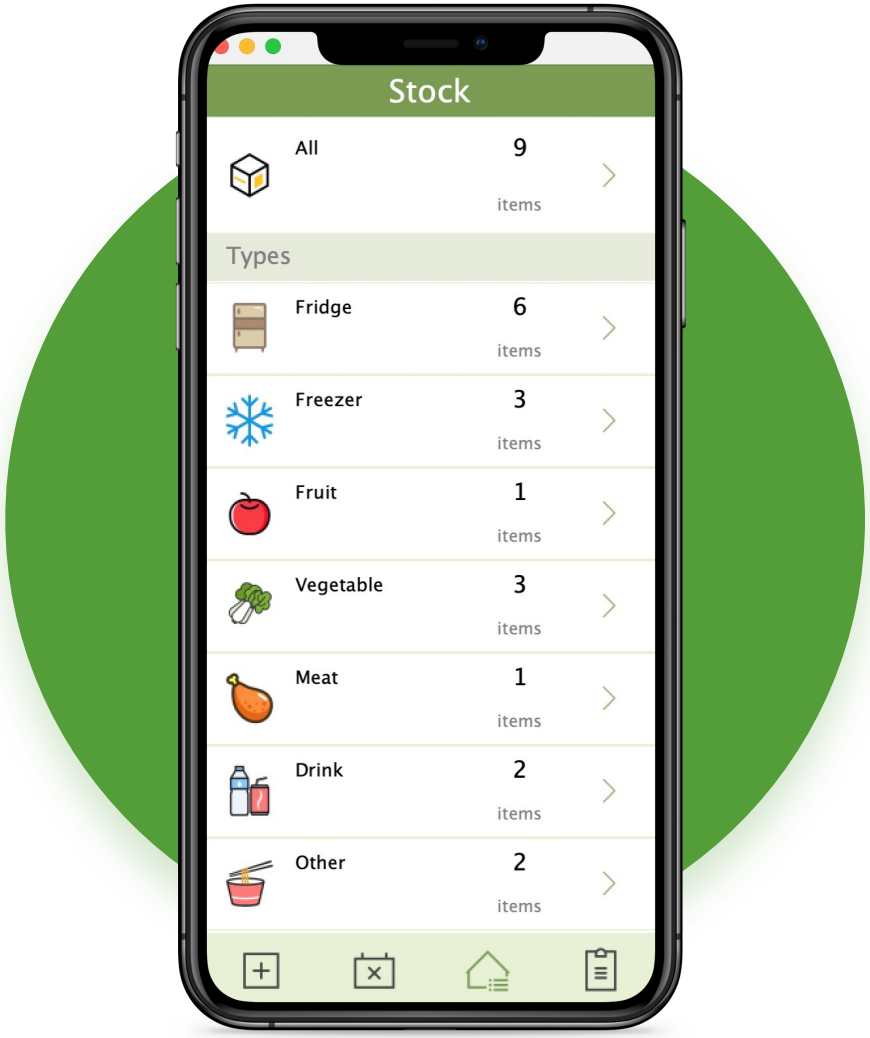
Manage and organize your Fridge



Stock Window

Check your stock by types

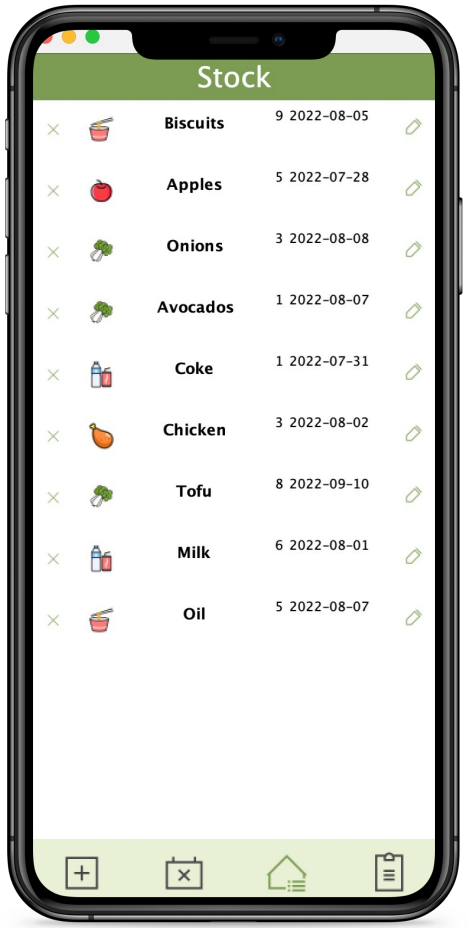
- **All – all items listed**
- **By location – fridge / freezer**
- **By types – fruit / vegetable / meat / drink / other**



Stock List Window

01. Remove the item

02. Edit the item



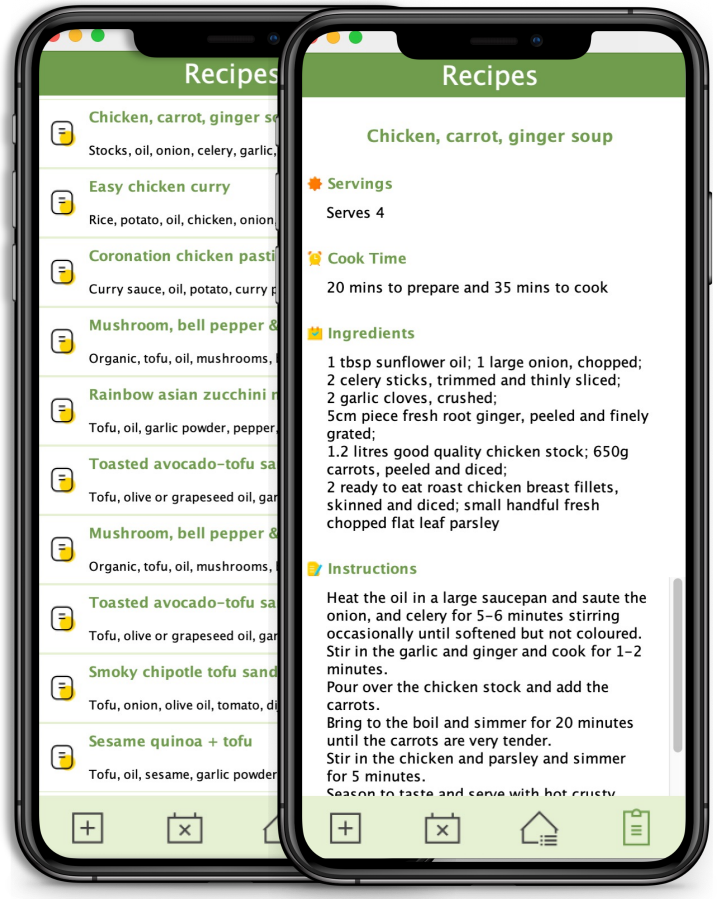
03. Type Icons – Easier to recognize the item

04. Expiration & Quantity

Feature 4 -- Recipe

Get Available Recipes in One. Single. Tap.

Recipe Window



- **Recipe Window**
  - Show 10 recipes based on current stock ingredients
  - Sequence: The recipe that contains the largest number of stock ingredients shows first
- **Recipe details window**
  - Show the recipe name, number of servings, cook time, ingredients and instructions
  - Formatted text -- **JTextPane**
  - Scrollable text -- **JScrollPane**

Models

FoodItem & RecipeItem

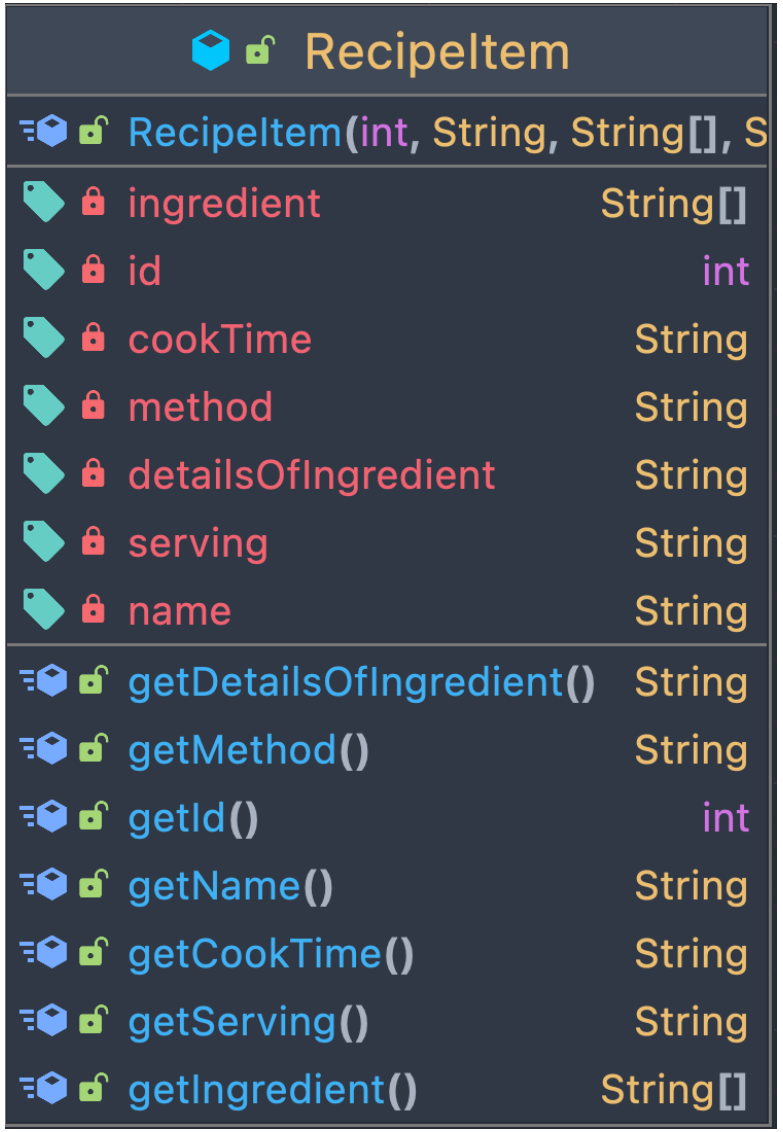
FoodItem

- **Five Instance Variables**:
  - Name
  - Quantity
  - Type (food types)
  - Location
  - id (automatically generated when loading the database; only used for operating database)

- **Two Enums**:
  - FoodType {VEGETABLE, MEAT, FRUIT, DRINK, OTHER}
  - PlaceLocation {FROZEN,REFRIGERATED}

RecipeItem

- **Seven Instance Variables**
  - Name
  - Ingredients (used for matching stock ingredients)
  - Details of ingredients (used for display)
  - Method
  - Cook time
  - Serving
  - id (automatically generated when loading the db; only used for operating database)



Controller 1 -- Stock

Stock Database

Stock Database

Step 1 – Design the database

- Use a **text file** as the stock database
  - Why? – Easier to read & write
  - Would upgrade to a SQL database after taking the class
- **Operations to the database**
  - **Load the stock list** – **Read** the txt & extract information from strings
  - **Add** an item -- **Append** to the txt
  - **Remove** an item – **Rewrite** the txt
  - **Edit** an item – **Rewrite** the txt

```
1  Biscuits, 9, 2022-08-05, OTHER, REFRIGERATED
2  Apples, 5, 2022-07-28, FRUIT, REFRIGERATED
3  Onions, 3, 2022-08-08, VEGETABLE, REFRIGERATED
4  Avocados, 1, 2022-08-07, VEGETABLE, REFRIGERATED
5  Coke, 1, 2022-07-31, DRINK, FROZEN
6  Chicken, 3, 2022-08-02, MEAT, FROZEN
7  Tofu, 8, 2022-09-10, VEGETABLE, FROZEN
8  Milk, 6, 2022-08-01, DRINK, REFRIGERATED
9  Oil, 5, 2022-08-07, OTHER, REFRIGERATED
```

Step 2 – Load the database

- How to load the database from text file?
  - Read lines from the txt file (**BufferedReader**)
  - Extract information from text
    - **Split the string** with the comma sign
  - Create a FoodItem object with the info
  - Then generate a stock list

```java
private void loadStockList() {
    // Store the items in an arraylist.
    stockList = new ArrayList<FoodItem>();

    // Read from the txt file
    try{
        BufferedReader br = new BufferedReader(new FileReader( fileName: "database/DatabaseOfStock
        String s;
        while((s = br.readLine()) != null) {
            String name;
            int quantity;
            Date expiration;
            FoodItem.FoodType type;
            FoodItem.PlaceLocation location;

            // 1. Extract information from text
            String[] arrOfStr = s.split( regex: ", ", limit: 0);
            name = arrOfStr[0];
            quantity = Integer.parseInt(arrOfStr[1]);

            SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd");
            expiration = formatter.parse(arrOfStr[2]);

            type = FoodItem.FoodType.valueOf(arrOfStr[3]);
            location = FoodItem.PlaceLocation.valueOf(arrOfStr[4]);

            // 2. Create objects with the info
            int newId = stockList.size();
            FoodItem newItem = new FoodItem(newId, name, quantity, expiration, type, location);
            stockList.add(newItem);
        }
        br.close();
    } catch (IOException | ParseException e) {
        throw new RuntimeException(e);
    }
}
```

Stock Database

Step 3 – Methods

- Database operations (add, edit, remove)
- Window display

Type	Method Name	Return Type	Functions
Add	**addItem** (**String** name, **int** quantity, **Date** expiration, **FoodItem. FoodType** type, **FoodItem.PlaceLocation** location)	void	Add a new item (Parameters required)
Edit	**updateItem** (**FoodItem** item, **String** name, **int** quantity, **Date** expiration, **FoodItem. FoodType** type, **FoodItem.PlaceLocation** location)	void	Edit a current item (Original object and new parameters required)
Remove	**removeItem** (**FoodItem** item)	void	Remove a current item (Original object required)

Type	Method Name	Return Type	Functions
Stock Window Display	**getItems** (**Stock.StockType** type) Type includes： {ALL, FROZEN,REFRIGERATED, VEGETABLE, MEAT, FRUIT, DRINK, OTHER}	ArrayList <FoodItem>	Get the specific type of list of items (Type required)
Notice Window Display	**getExpiredItems** ( )	ArrayList <FoodItem>	Get the list of expired items
	getAlmostExpiredItems ( )	ArrayList <FoodItem>	Get the list of expiring items
	getLowStockItems ( )	ArrayList <FoodItem>	Get the list of low stock items (low stock: quantity < 3)

Stock Database

Difficult part

- How can the database figure out which object to edit/ remove?
- How to solve name conflicts?
- Answer: Assign a unique id to each stock item

Solution

When loading the stock list from the text file: set an id (based on current size of array)

```
int newId = stockList.size();
FoodItem newItem = new FoodItem(newId, name, quantity, expiration, type, location);
stockList.add(newItem);
```

When editing / removing an item: check the id

```
public void removeItem(FoodItem item) throws IOException {
    // Edit the stockList
    stockList.removeIf(itemInList -> itemInList.getId() == item.getId());
```

Controller 2 -- Recipe

Recipe Database

Recipe Database

Step 1 -- Build the database (Python)

- Wrote a web crawler to collect recipes from websites
- Formed a database with a txt file
- Collected 200+ recipes

```python
def recipeCollector():

    # Extract recipe URLs
    recipe_urls = []
    ingredient_list = ['avocado', 'chocolate','kale', 'tofu', 'quinoa', 'blueberries', '
    for ingredient in ingredient_list:
        base_url = f"https://simple-veganista.com/tag/{ingredient}/"
        response = requests.get(base_url)
        soup = bs4.BeautifulSoup(response.text, 'html.parser')
        entry_soup = soup.find_all('h2', class_='entry-title')
        for content in entry_soup:
            recipe_urls.append(content.find_all('a')[0].attrs['href'])

    print("URLs Collected!")
    print(recipe_urls)
    print("==============================")
    time.sleep(10)

    # Get recipe information from each URL
    counter = 0
    for recipe_url in recipe_urls:
        response = requests.get(recipe_url)
        soup = bs4.BeautifulSoup(response.text, 'html.parser')

        # Get information from the URL.
        result = []
        # Name
        name_soup = soup.find_all('h2', class_='tasty-recipes-title')
        name_content = name_soup[0].get_text()
        result.append(name_content)

        # Detailed Ingredients
        ingredient_details_soup = soup.find_all('div', class_ = 'tasty-recipes-ingredien
        if (ingredient_details_soup != []):
            ingredients_details_content = ingredient_details_soup[0].get_text().strip("\
        else:
            ingredients_details_content = "N/A"
        result.append(ingredients_details_content)
```

Some of Python codes

Recipe.java × DatabaseOfRecipes.txt ×

```
***Recipe***
Chicken, carrot, ginger soup///
stocks, oil, onion, celery, garlic, ginger, carrot, chicken, parsley///
1 tbsp sunflower oil; 1 large onion, chopped;
2 celery sticks, trimmed and thinly sliced;
2 garlic cloves, crushed;
5cm piece fresh root ginger, peeled and finely grated;
1.2 litres good quality chicken stock; 650g carrots, peeled and diced;
2 ready to eat roast chicken breast fillets, skinned and diced; small handful fresh chopped
Heat the oil in a large saucepan and saute the onion, and celery for 5-6 minutes stirring o
Stir in the garlic and ginger and cook for 1-2 minutes.
Pour over the chicken stock and add the carrots.
Bring to the boil and simmer for 20 minutes until the carrots are very tender.
Stir in the chicken and parsley and simmer for 5 minutes.
Season to taste and serve with hot crusty bread.///
Serves 4///
20 mins to prepare and 35 mins to cook///
******
***Recipe***
Easy chicken curry///
rice, potato, oil, chicken, onion, curry sauce, spinach///
300g Grower's Harvest long-grain rice;
300g Redmere Farms baby potatoes, halved or quartered if large;
1 tbsp vegetable oil; 365g pack Willow Farms diced chicken breast;
1 Redmere Farms brown onion, sliced; 440g jar Hearty Food Co curry sauce;
100g Redmere spinach; ///
Cook the rice to pack instructions.
Boil the potatoes for 10-15 mins until just soft, then drain.
Meanwhile, heat the oil in a heavy-based frying pan over medium heat.
Add the chicken and cook for 5 mins, turning halfway, until golden all over.
Add the onion and cook for 3-5 mins until soft.Pour in the sauce and add the potatoes.
Bring to the boil, then reduce the heat and simmer for 5-10 mins until the chicken is cooke
Stir in the spinach to wilt, then serve with the rice.

Freezing and defrosting guidelines:
In order to enjoy optimum flavour and quality, frozen items are best used within 3 months o
///
```

The database file

Recipe

Step 2 -- Load the database (Java)

- **Read lines from the txt file**
- **Extract information from text (split the string)**
- **Then generate a recipe list**
- **When we create a Recipe object, itself runs the loading method ↑**

```
public Recipe() {
    loadRecipeList();
}
```

Type	Method Name	Return Type	Functions
Public	**getSpecificRecipeList** (**String[ ]** input Ingredients, **int** range)	ArrayList<RecipeItem>	Input a String[ ] type of ingredients in the stock database, and the number of recipes you need. Get a list of recipes that contains the input ingredients.
Private	**loadRecipeList** ( )	Void	Read recipes from the txt file, and then load a recipe list.

Recipe

```java
public ArrayList<RecipeItem> getSpecificRecipeList(ArrayList<String> inputIngredients, int range) {
    ArrayList<RecipeItem> list = new ArrayList<~>();
    HashMap<Integer, Integer> map = new HashMap<~>();

    // Check every the input ingredients in every recipe.
    for (String ingredient : inputIngredients) {
        for (RecipeItem item : recipeList) {
            // if this recipe contains the specific ingredient, then map value + 1
            if (Arrays.asList(item.getIngredient()).contains(ingredient)) {
                map.put(item.getId(), map.getOrDefault(item.getId(), defaultValue: 0) + 1);
            }
        }
    }
    HashMap<Integer, ArrayList<Integer>> hitMap = new HashMap<~>();
    for (int id : map.keySet()) {
        int hitting = map.get(id);
        if (hitMap.containsKey(hitting)) {
            ArrayList<Integer> value = hitMap.get(hitting);
            value.add(id);
            hitMap.put(hitting, value);
        }
        else {
            ArrayList<Integer> value = new ArrayList<~>();
            value.add(id);
            hitMap.put(hitting, value);
        }
    }
```

Step 3 – Get the recipe list that contains stock ingredients

- Two for-loops:
  - **Cross match** stock ingredients with recipe ingredients
- Store the result in a **Hash Map** (key: recipe; value: hitting count)
- **Reverse the hash map**
  - Key: hitting count; Value: recipe
  - Why? - Easier for ranking
- **Return a list of recipe based on the rank of hitting count**

```java
// Return a list of recipe containing the input ingredients (By the rank of hitti
Object[] keySet = hitMap.keySet().toArray();
Arrays.sort(keySet, Collections.reverseOrder());
for (Object hitting : keySet) {
    int hitCount = (int)hitting;
    for (int id : hitMap.get(hitCount)) {
        list.add(recipeList.get(id - 1));
    }
}

// Only return the recipes in the given range
ArrayList<RecipeItem> result = new ArrayList<~>();
for (int i = 0; i < range; i++) {
    result.add(list.get(i));
}
return result;
```

The last Part

Q & A