Food My Thought

A proposed app

# Contents

[Contents 1](#_Toc176178688)

[Purpose 2](#_Toc176178689)

[Two modes: lister and proposer 3](#_Toc176178690)

[What constitutes A Food Item? 3](#_Toc176178691)

[Viewing Recipes (and Choose your own adventure!) 4](#_Toc176178692)

[Listing ones food items 6](#_Toc176178693)

[Proposing recipes 7](#_Toc176178694)

[Responding to recipes 9](#_Toc176178695)

[Data schema 9](#_Toc176178696)

[I, personally, don’t like making accounts for every itty-bitty thing 10](#_Toc176178697)

[Sharing is caring! 10](#_Toc176178698)

[Our great and glorious human civilization has better things to do with storage power 13](#_Toc176178699)

[When should we delete a larder? 13](#_Toc176178700)

[How should we delete a larder? 14](#_Toc176178701)

[None Pizza without left beef 16](#_Toc176178702)

[Changing Categories. 16](#_Toc176178703)

[Simple renaming 17](#_Toc176178704)

[Category Changes 18](#_Toc176178705)

[To err is to human 19](#_Toc176178706)

[Items with recipies: Ballooning Undo 20](#_Toc176178707)

[Category name change 20](#_Toc176178708)

[Undo the Undo 20](#_Toc176178709)

[Total Refresh 21](#_Toc176178710)

[Ain’t no Rest for the wicked 21](#_Toc176178711)

[Think of how stupid the average person is, and realize half of them are stupider than that! 21](#_Toc176178712)

# Purpose

When I was on my last month when living in Wisconsin, I was beset with the difficulty of using up all the food Items I had amassed over my time in Wisconsin.

Utilizing all this food required designing recipes for each and every item in my larder. Such menu-work was quite a pain. Asking other people IRL generally winds up with the most milquetoast, obvious recipe (AKA nothing useful if I wanted new suggestion). Meanwhile, the internet, despite its vast depository of information, comes up short for things too exotic.

What is to be done? In such a situation, an app to beseech the wisdom of anonymous strangers on the internet would have been quite helpful.

Thus the goal of this app: This allows someone with too much food find ways to prepare such consumables.

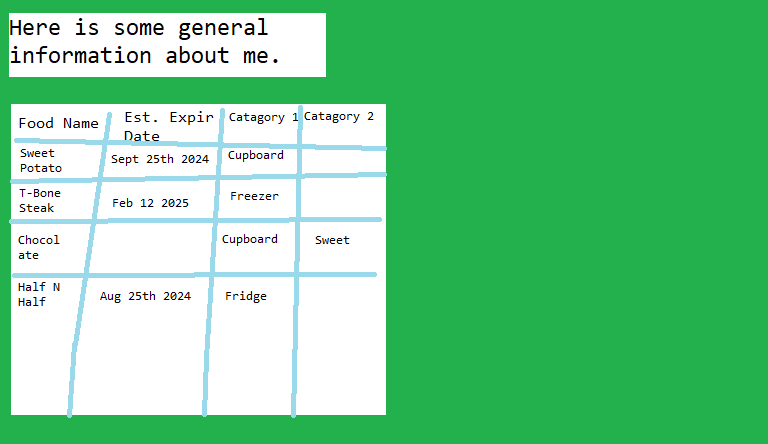
# Two modes: lister and proposer

There are two types of people who are concerned about any given larder: The Larder owner (Hereby referred as the “Lister”, as he is the one who lists all his food items), and the recipe proposer.

Because the tasks these two types of people do are quite different, they will have their own separate screens. These will be covered in later sections. What is covered in this section is how the lister actually gets their larder to the proposer to review.

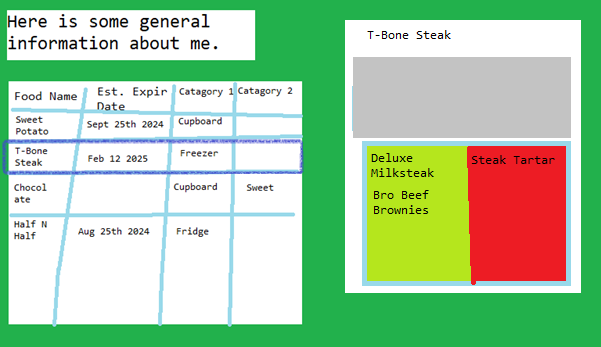
# What constitutes A Food Item?

The content of the communications between comrades consists of consumables. Ergo, no matter the mode, whether lister or proposer, the left hand of the screen shall look the same.



The top of the left-hand side is an dossier written by the lister. It holds general information that applies to all of the recipes. I suspect it will mostly be used for dietary restrictions.

When selected, most of the right-had side is taken up by a card .



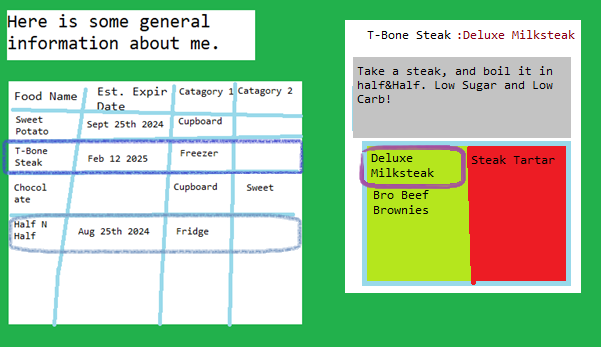
The card for a particular food-item doesn’t display the generic information (AE it’s expiration date & Categories.) This is because it is already visable on the left –hand side.

On the bottom of the card are two lists: the lime-green is a list of suggested recipes, and the red is the list of recipes the listed has rejected. As you can see, the chef-to-be is still pondering making deluxe milk steak or bro-beef-brownies with his T-bone. He has, however, rejected the suggestion of using his T-bone for steak tartare

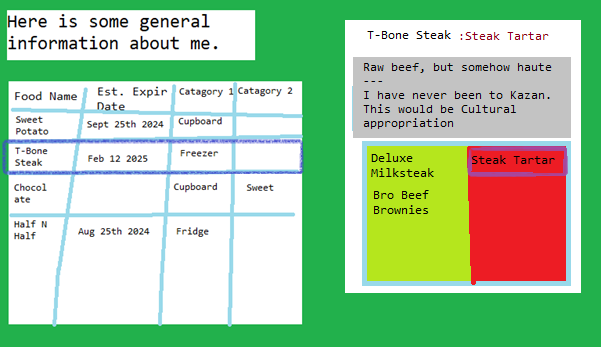
What then, takes up the grey area?

# Viewing Recipes (and Choose your own adventure!)

When selecting a recipe, one sees something like this

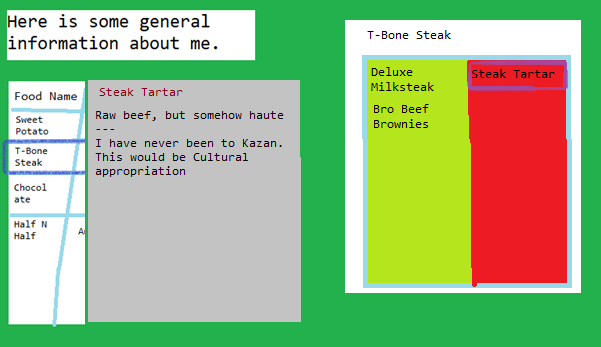


As you can see, the grey section is being taken up by the description of a recipe. Furthermore, Half and Half, another ingredient in the larder is highlighted.



Here a rejected recipe is selected, and we can see the lister’s response.

HOWEVER! I am posting this on github. The PUBLIC has POWER. By that, I mean, it is also possible for a selected recipe to look like this:



It leaves a whole bunch more room for the recipe’s description, as well as for the list of recipes.

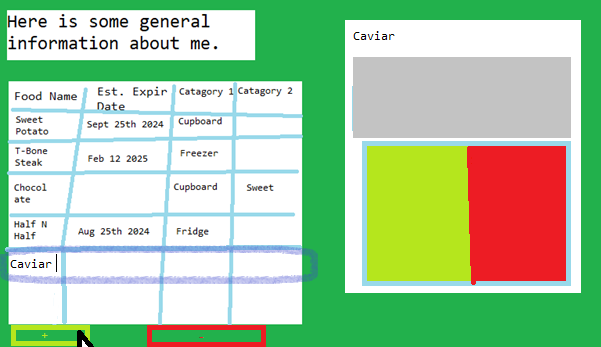
It does have a cost of ditching the information about categories and expiration dates, but I am assuming that persons reading a recipe likely is focused on the recipe, and doesn’t care about the associated information for random foods.

This is where the public comes in. I would prefer version one (with the recipe description part of the fooditem card), in part because it keeps the RHS consistent, and in part because it would be (Presumibally) easier to code.

But if the professional public profoundly prefers version 2 (Replacing associated information with the recipe discussion), I’ll go with that.

# Listing ones food items

The lister, as one might expect, needs to be able to list all the fooditems in their larder. Pursuant of that, the lister screen has a button for adding a food-item.



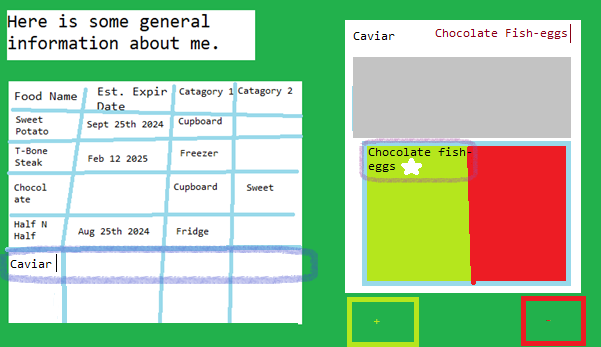
Here he has recently clicked the “Add fooditem” button. This adds another fooditem to the list, and the lister can fill in the information by typing it in. Navigation can be done via tabbing or by clicking on a specific fooditem.

There is also a “Delete FoodItem” button, as well. As food items may have recipes associated with them, this adds additional complexity: See the [None Pizza without left beef](#_None_Pizza_without) section.

# Proposing recipes

Both the lister and proposer have the option to propose recipies. The proposer, because her job is solely to propose recipes, and the lister so he isn’t deprived of some of the fun of this joyous app.

We’ve already seen enough from the lister’s POV. So here’s the proposer, having already selected a fooditem, and having already pressed the “Add Recipe” button.



Control of the cursor immediate leaps to the top-left corner, or the title of the recipe. The user can then simply write the name of the recipe. The name of the recipe in the recipe list will be updated automatically, using the magic of react.

I put the editing of the name above the description pane because that would allow a very logical tab-sequence. Tabbing goes down. Eazy-Peazy.

Of course, Chocolate Fish-eggs also include, well, chocolate. Thus it is essential to allow selecting of other ingredients when describing a recipe.



Once again, there is a delete recipe button for deleting recipes. Deleting the recipe does what it says on the tin, as well as deleting the recipe from the recipe list of every food item that is tagged by the recipe.

You may have noticed a small star next to the recipe for chocolate fish-eggs. The program shall remember that this particular recipe was written in this particular session, and allow the proposer to both edit or delete such a recipe in the future.

# Responding to recipes

Once the lister’s list has been fed with the finest of culinary arts the internet can provide, he needs to be able to curate his suggestions.

First of all, the lister is able to type respond to any recipe suggestions.

Second of all, there is a new button! This button allow the user to shift a particular recipe into the “Rejected Recipes” list. Here you can see that the lister is a tad bit miffed with the proposer’s proposal to ruin both perfectly good chocolate and perfectly good caviar.



At the bottom you can se that the “Reject this recipe” button has become the “Reconsider, this might actually be good” button.

# Data schema

Data shall be stored using SQL. Mostly because pretty much every job description for anything software development related askes for it. I have yet to be convinced from my position that “Anyone smart enough to understand Minecraft Redstone is smart enough to ‘get’ SQL”, but that’s what the market is asking. The market in this case is flawed and stupid, but even if the invisible hand has brainworms, it’s still the invisible hand.

Where was I? Oh yes, data shall be stored using SQL.

We have already touched on the attreibutes that:

* each profile needs (description and a list of fooditems, as well as the [estimated death date](#_Our_great_and)), as well as the attributes of
* Fooditems (Name, Expiration date, category 1 value, category 2 value, list of recipies, both proposed and rejected) Lastly, we already know how the data pertaining to each:
* Recipe (Name, Fooditems utilized, description, response, rejection status)

Note that each fooditem and each recipe are unique: That is if 2 listers list “Chocolate” as one of their fooditems, the program will store two chocolates. One for lister 1, and another for lister 2. After all, both listers will send their larders to different people.

# I, personally, don’t like making accounts for every itty-bitty thing

That is, I would be hesitant to use this app if it required me to send my e-mail and type in a password every time I called on nameless trashposters and internet trolls to tell me what exactly to do with my Quinoa.

As such, this app would rely on cookies.

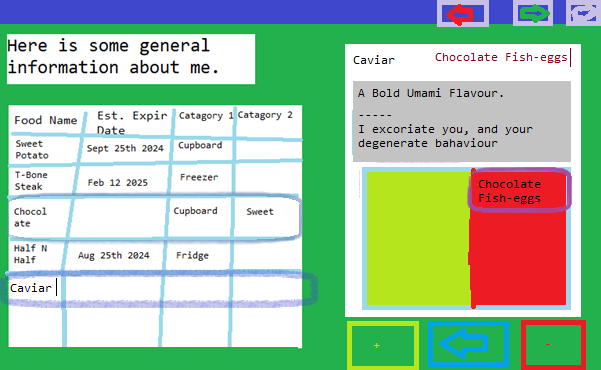
The url for the lister page would be the same, irreguardless of who is the one posting. Each lister would get a cookie, which contains a unique code that would allow the lister to connect with the correct larder. This means that:

1. The larder is stored on the internet, in case a proposer wanted to take a look at it.
2. The lister can return to his pantry, with the full administration powers that come with being the owner of said larder.
3. No one needs to make an account!

# Sharing is caring!

Of course, the lister’s larder lacks usability without any way to send it over to the proposer.

To compensate for this, the lister should have a button, which copies the larder into the clipboard.

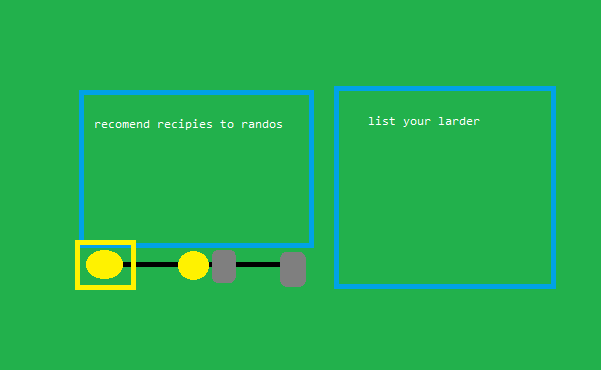


The lister can then use that link to pose as a proposer, or post the link in a group chat or anywhere else online

That’s all well and good, but what if the lister has no friends? Or, almost as bad, all their friends are too busy focused on real life? Or has already asked his friends and prefers the endless creativity of various internet strangers?

Indeed, up to this point I neglected to show a picture of the home screen: That is, not everyone using this app is going to be cataloging all of their food. Which means automatically shoving any new user to write down their own larder would be pointless.

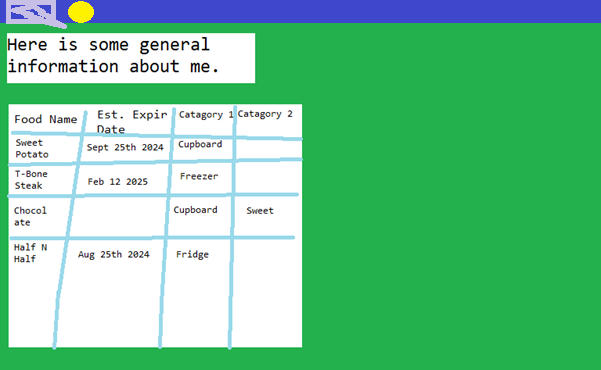
Which means, upon opening this website, the user gets greeted with the below.



“Recommend recipes to randos” will send the user to a random larder as a proposer, while “list your larder” sends the user to their larder as a lister.

You may have noticed the slider at the bottom. By default (Yellow Smiley face emoji), it only allows us to cycle through live larders. But if the proposer wishes to delve into the esoteric world of dead larders (Grave Emoji), they have the option to do so. They also can have a complete random (Represented by both emojis)

Going back to the homepage of this website every time you hop onto a different larder to recommend would be a bit of a hassle.

Which means, of course, that we need an additional two buttons.

Here we see it as a proposer. The first button sends the user to another random larder, again as a proposer. A lister will also have this button, if he wants to switch colors and become a proposer.

The second button again allows us to choose what we cycle through: Only live larders, live Larders and lifeless larders, or only lifeless larders. This button changes which randomization/liveliness setting will be used for the next roll

# What can you make with Moldy cheese?

How do we avoid continuously serving our prospective proposers with long-forgotten larders? If the lister never logs on to look at their larder , and the larder is boring enough that no-one has commented on any of the food items, what is to be done? The proposer is an inventor, a woman who brings forth visions. I suppose that the proposer would like to only allot her limitless creativity only on such men/listers who have the fortitude to see her vision to fruition. For that end, I propose we assume that larders which are left unused long enough will be marked as dead.

This allows most proposers to only be matched with listers who actually keep tabs on their larders. Unless they switch their randomization rolling settings.

## When should we delete a larder?

How should this liveliness assumption work? Fortunately, we have an already-existing timeframe of when the larder is kaput: The estimated spoilage variable. We simply find the least perishable item, and use that date as the day we forget about the given larder.

Of course that runs into three problems: Milk and honey.

First: **Laziness**. Filling out the expiration dates for every bit of food is work. Some people might choose not to do it. Now this app is up the creek without a paddle. It doesn’t know when to delete this particular larder.

**Milk** spoils quite fast. If a particular lister only has milk, bananas, and avocados, then using the expiration date means we have to wipe out this particular larder after only a week or so. So not enough time for a proposer to leave a recipe suggestion without getting that proposal discontinued before the lister even has time to read it.

**Honey**, as we all know, lasts forever. If a lister honestly listed the expiration date for their jar of honey, this would slate the larder to be unalived by year 4000 AD.

I greatly commend the descendant’s dedication to preserving family heirlooms. Due to their exceptional filielial piety, they have in their possession a sacred talisman of their long-lost homeword. This steadfast guardianship of a relic of the ancient paradise of olds has brought great prestige to the house of lister.

Of course, it would be profane, blasmephous, unholy, and quite uncouth to pollute Sweet Nectar of the Queen of Stripes. Doubly so under the direction of a bored teen from 2020’s New Jersey. All that is to say, the great and pious house lister no longer has need for guidance.

That is, of course, even supposing that the server hosting this app lasts that long. Which, in turn, assumes that AWS was left unscathed by the intergalactic Human-Forgwyz Cyberwar of 2783-2789. Which I doubt.

**I hope that amuse-bouche of science fiction writing provided a welcome change of pace from boring technical documentation. Now, back to boring technical documentation**. What I was trying to get with the previous sections is there are flaws with relying solely on self-reported expiration dates. Instead, each larder keeps track of the date it’s supposed to be unalived.

Upon first creating a larder, the deletion date is set for either:

1. 1 year, 6 months from the creation date, or
2. The expiration date of the least perishable food item

Whichever is further in the future.

After the larder is created, its unalived date can be pushed back by various activites

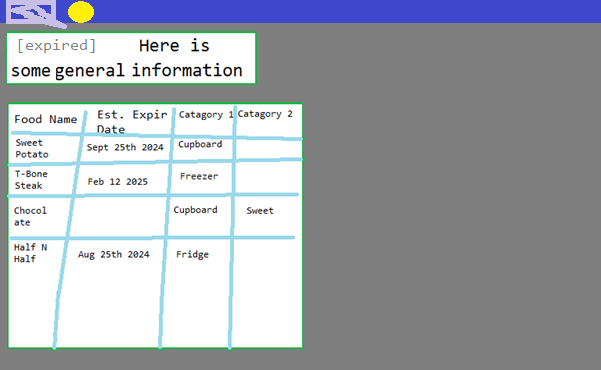
|  |  |
| --- | --- |
| Activity | Additional Time before larder is unalived. |
| Leister looks at his larder a subsequent time | Half a month |
| Proposer proposes a recipe | 1 week |
| Lister writes a text-response to a recipe | Half a month |
| Lister moves a recipe to or from the rejected recipe section | 1 week |
| Lister adds an additional fooditem | One month |
| Lister deletes a fooditem | Half a month |
| Lister changes a category name | Half a month |
| Lister changes a category value after changing a category name | 0 |
| Lister changes a category value, without having previously changed the category name | 1 week |

To avoid the sticky situation that honey puts us in, we shall cap this counter at 3 years. If any addition to the “mark as dead” date puts the “mark as dead” date more than 3 years in the future, any additional time is simply lopped off.

One last point: if a user logs onto their larder after it was slated to be unalived (Which is entirely possible, given our deletion schema), then we simply un-unalive the larder. AE we once again re-set the “To be marked as dead” date to 1 year, 6 months, or expiration date of the hardiest fooditem.

## Please let me gander at this corpse of a larder?

Should a larder be marked as dead, this changes how it looks:



The grey signifies that larder’s is bereft of life, shuffled off 'is mortal coil. If that’s not enough, we also affix a nice “Expired” tag to the front of the larder’s description.

You may notice that this is from the proposer’s point of view. This is because, as stated earlier, having the listed log on re-sets an deceased larder back to life.

## How should we unalive a larder?

Should we have the server run a clock constantly checking the liveliness of each larder? No. For two reasons. One, it’ll take up too much electricity, and two, it sounds like something that would be hard.

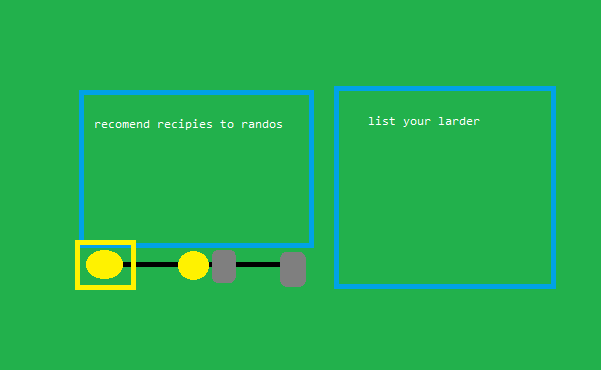
Luckily, we don’t need to rely on such a clock. As Stated earlier, there is a button to take a potential proposer to a random larder. If we press that enough times, we’ll eventually land on our out-of-date larder, simply via the law of large numbers.

Therefore, we rely on our random “clock” to mark our unused larders.

If a proposer navigates to an expired larder, there is a request to the server for all the information related to said larder. The server checks the date, and if today’s date is after the delete date, then the server marks the larder as expired.

Where does that leave the proposer? If the proposer was using the “Go to random larder” button, the server simply randomly chooses a different larder. If the proposer used a link, we just give them the expired larder. If this app lacks luck and there is no live larder to look at, that also gives a screen telling the potential proposer of the unfortunate situation, plus the buttons to either create their own larder or ~~navigate to a random larder~~ get sent back to the home screen.

You may have noticed that in the “you cannot take a look at a larder, for they are all expired.” situation, the screen described is remarkably similar to the home screen.



Which, of course, means that the home screen has to be set up so that it can take an error text variable, for situations like telling a proposer that they cannot reach a larder.

# Our great and glorious human civilization has better things to do with storage power

~~How do we avoid clogging up our server with long-forgotten larders? If no-one logs on to look at their larder in a long time, and no-body’s commenting on it anymore, what use is it? For that end, I propose we assume that larders which are left unused long enough can be safely deleted. Then, after making such an assumption, we delete them.~~

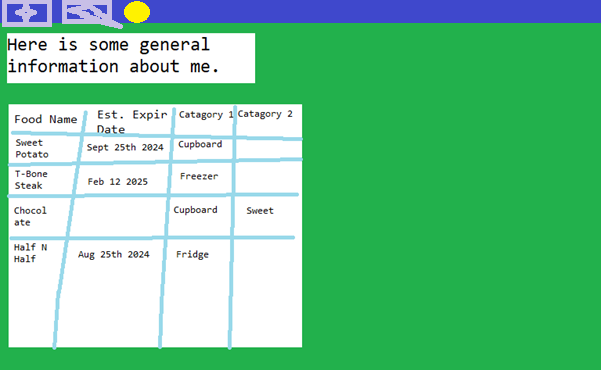
Thanks to a wonderful tactic known as “Kicking the can down the road”, we shall hold off on deleting unused larders until this website gets around a hundred thousand larders. Until then, storage is cheap, and if someone has the deep curiosity to cruise through old larders and recipes, they can knock themselves out.

Unfortunately, some people might get their panties in a twist if they learnt that their previously-reliable list of food, after 4 or so years of being unused, has been entirely deleted. Sigh.

# rOle reversal.

There is already a button for a lister to abandon his post and troll larders as a proposer: it is the “send me to a random larder” button.

But what of the reverse? There is yet another button needed before the proposer’s toolbar is complete.

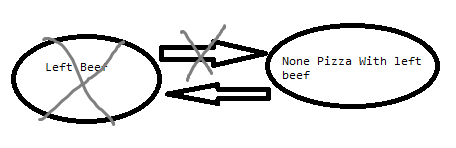


This button, when clicked, returns a proposer to her larder, as a lister.

# None Pizza without left beef

Let’s say we have a larder that lists left beef. Left beef has only a single recipe, “None pizza with left beef”.

Unfortuenently, the lister left his left beef on a lift, and thus our lister is left without left beef. The lister leaves left beef off his list. That is, he deletes the entry for left beef.



As we probibally know by now, the fooditem object for left beef has a recipe list, which points to the recipe object for “None pizza with left beef”. “None pizza with left beef”, in turn, has a list of food items, which points to the recipe object for left beef. Both of these pointers will be deleted: the fooditem’s pointer by deleting the fooditem, and the recipe item’s pointer to prevent recipes from pointing to non-existent fooditems.

Now that the only fooditem that points to “None pizza with left beef” is deleted, “None Pizza with left beef” is orphaned. Normally, how a user views a recipe is to click on a food item that uses said recipe, and from the fooditem’s card, clicks on the recipe to see the recipe’s information. Now that “Left beef” is deleted, there is no way for a user to interact with “None pizza with left beef”. “None pizza with left beef” now solely takes up valuable server memory, and nothing else.

Therefore, I propose that we immediately and without delay kill any and all orphans as soon as it is possible to do so.

That is, once the final fooditem that a recipe utilized is deleted, we delete the recipe.

# Changing Categories.

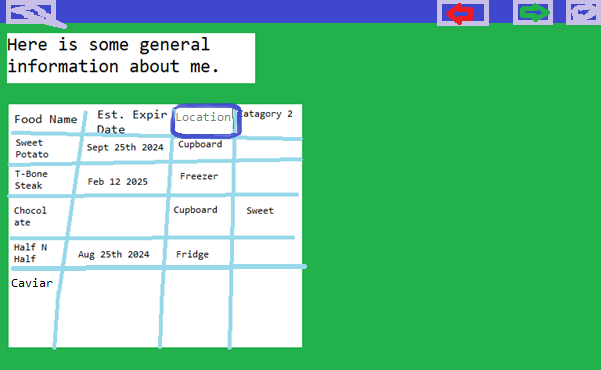
Take this list of food:

|  |  |  |  |
| --- | --- | --- | --- |
| Food Name | Est Expir Date | Category 1 | Category 2 |
| Sweet potato | Sept 25th 2024 | Cupboard |  |
| T-Bone | Feb 12 2025 | Freezer |  |
| Chocolate |  | Cupboard | Sweet |
| Half & Half | August 25th 2024 | Fridge |  |
| Caviar |  |  |  |

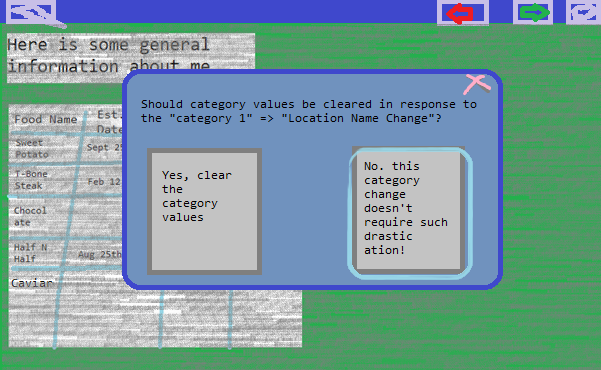
## Simple renaming

The first category is obviously the physical location of the food, but the name is still “Category 1” When I was demonstrating the layout of the larder, leaving this category’s name as category 1 helped demonstrate that the categories are entirely arbitrary. But now it’s time to give category 1 a proper name.

The lister can change the category name by clicking on the category box, and typing in the new name,



Upon either pressing enter or the edited category losing focus, we give the user a pop-up



As you can see, the control defaults to “Don’t clear out our entire list”

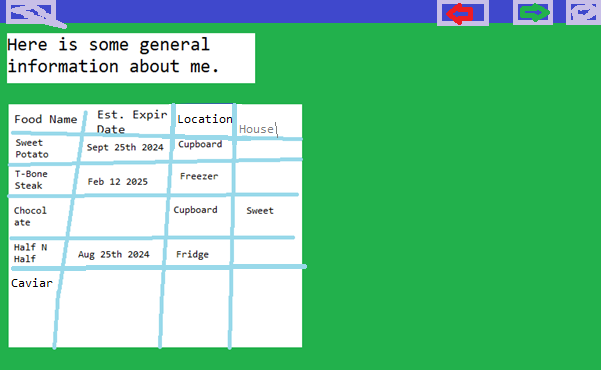
Clicking outside the pop-up box works similar to pressing the (Admittedly hard-to-see) X-out button: That is, it cancels the operation.

Why is there this warning? That brings us to the next button.

## Category Changes

Category 2 is currently a sweet/savory divide. But Chocolate is the only “Sweet” thing on that list. Plus, given that this list has sweet potato, categorized as savory, category 2 breaks down.

Let’s change Category 2 to a more sensible value.



As we all know, there are only 4 values possible for “House”: Gryffindor, Ravenclaw, Hufflepuff, and Slytherin. “Sweet” ain’t in it.

If we did not delete the “Sweet” status from chocolate automatically, the user would have to change it manually. Which would be a pain if the lister lister more only 1 sweet fooditem.

That is to say, there shall be an option to automatically clear every category value for a particular category.

# To err is to human

What if the lister clicks the wrong “Do you want to clear all category list values?” What if they delete a fooditem instead of deleting one of the many recipes?

It is possible for the lister to goof. This is ok.

The proposer can also goof, but as the proposer’s actions are vastly more constrained, if he goof, he can just delete or repeal the proposed recipe. Which means I am focusing solely on the lister.

Therefore, we have to keep an around 10 or so long list of previous user actions.

These will store a log of the changed item, as well as what sections were changed. So, for example, when we change the name of our sweet potato to “Yam”, we put onto the previous actions list both a log stating that the Food name of the Yam/Sweet potato was changed from Sweet potato. Simple as.

A similar situation awaits editing a recipe, only this time, we will create a “recipe changed” log instead of a “fooditem changed” log. We will see situations where both a fooditem and a recipe gets changed at the same time.

## Items with recipies: Ballooning Undo

Now let’s say the lister deleted the half&half. The half and half stores a recipe for deluxe milk steak. As part of our deletion process, we delete the half&half from deluxe-milksteak’s ingredient list. Now riddle me this : do we also create a “Recipe changed” log entry?

The answer is YES! This recipe changed log entry contains nothing but a note that deluxe milksteak used to include half & half. This creates an additional log, but it’ll allow the “Connecting a recipe and a food item” task to always be in the same kind of undo log.

Why the *recipe changed* log? If we delete a recipe that points to multiple fooditems, it would be much simpler to task the single *recipe changed* log to re-connect with its former food items, then to create a *recipe changed* log for everything besides the recipe’s fooditems, and then a horde of *fooditem changed* logs to re-connect the fooditems with the recipes.

Now let’s switch from our proper example larder, back to the [None Pizza without left beef](#_None_Pizza_without).

The lister has Deleted left beef. This gave the Order of Necromancers great power.

Then the lister undid his deletion. Chivalry was restored to the land.

Let’s ignore the rise and fall of darkness, and focus on the lister’s User Experience.

When the user deleted left beef, this also deleted “None Pizza with left beef”. When the user undoes his deletion, he wants to see left beef, and for left beef to contain the recipe for “None Pizza without left beef”. Which means, when the left-beef lister deletes left beef, we have to create at least two logs, a *fooditem changed* log for the left beef, a *recipe changed* log for the “None pizza with left beef”, and, if there are more recipes using only left beef, more *recipe changed* logs for the other left-beef recipes. Again, the *recipe changed* log is both the only log type that can have multiple log entry’s in a single move, and is the one in charge of connecting fooditems and recipes.

## Category name change

Let’s say our category name is “House”. The user changes it to “Hogwarts house”. But oops! The user accidentally clicked the “Yes, clear these category values” button! Oh No!

This lead to our third undo log type. The *category changed* log. This log type stores both the original name of the category, as well as a dictionary connecting every fooditem to their previous category value. This avoids having to make a dozen *fooditem changed* logs for this particular action.

If a user does click the “Yes, clear these category values” button, then we still make a *category changed* log, but with only the original name of the category, and without the dictionary of former fooditem category values.

## General information change

There is extremely limited damage a lister can do by changing the “General information” part of his profile. Technically, it already has a built-in undo: the lister can just remember what was previously there. But it would cause confusion if this section, for some reason, had no undo while every other action the user can do results in an undo log.

Therefore, changing the general information section results in our fourth and final undo log type: the general information changed log.

## Undo the Undo

Of course, an undo without a redo is like a Margarita without a rim: Technically possible, but is disappointing and frowned upon in most circles.

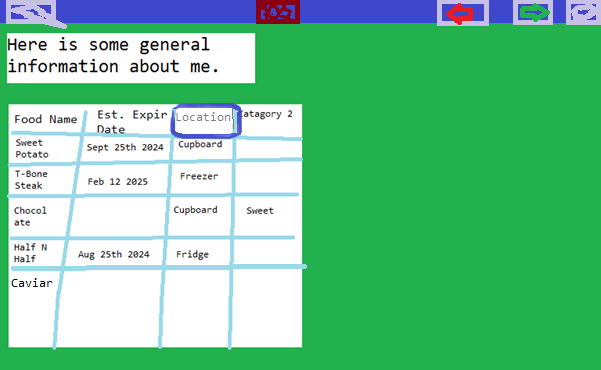
Redo logs are quite similar to undo logs. They would be practically the same, except theonly way to create a redo log is by clicking undo. That, and every action empties the list of redo logs.

I’m unsure what else to write on the topic, because by making all the infrastructure for undo, we would also have 95+% of the infrastructure for redo

# Total Refresh

It is not outside the realm of possibility that, at some point, a lister would like to delete every single item of food they already have listed.

For this, we need yet another button.



It’s in an awkward position, being in the centre, but it’s also far away from any other buttons that don’t have this level of harm.

Clicking on this button gives a warning message, asking if the user really truly understands the fallout of his or her actions. If the user avowedly opts for this outcome, then the entire larder, including recipes, category names, fooditems, and the general information, gets wiped out

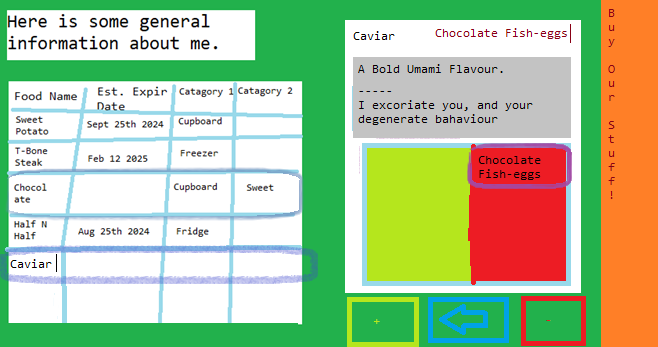
This would create rheams of undo logs. Yes, given the section above, it would be unenjoyable to undo a full wipe, but not too hard.

# Ain’t no Rest for the wicked

Money don’t grow on trees. I got bills to pay, I got mouths to feed. There ain't nothing in this world for free

I know I can't slow down, I can't hold back, though you know I wish I could

That is to say, there will be ads. I haven’t seen any website with ads like seen below,



But hopefully the power of ingenuity overcomes my odd layout, and I don’t go broke.