Motivation

Motivation

- Description
    - Our system will allow users to search for recipes that can be cooked using a subset of a given list of ingredients. Users can also give recipes ratings and save their favorite recipes
- Key purposes
    - Search for recipes that require a subset of a given list of ingredients.
        - Oftentimes, a user wants to cook something new or needs inspiration for what to cook but they don't have time to go to the grocery store and pick up new ingredients. Using our system, they can specify which ingredients the user already has in their kitchen and get recipes that they can cook using only ingredients that were specified. Also, the user can the amount of each ingredient that they possess, so that the suggested recipes returned will not include recipes that require more of an ingredient than the user has.
    - Allow users to rate recipes.
        - A user cannot always tell how good/tasty a recipe will be just by reading the recipe. User ratings of recipes will help users know better which recipes will yield good results. User ratings can help users choose what to cook.
    - Allow users to save recipes.
        - If a user particularly enjoyed a recipe that they discovered from our system, he or she may want to use that same recipe again. By allowing a user to save or favorite recipes, they can easily access those same favorite recipes again in the future.
- Deficiencies of existing solutions
    - Existing solutions such as Supercook and MyFridgeFood do not support recipe search based on the amount of ingredients. Therefore, if a user has an inadequate amount of a certain ingredient, they will not be able to actually cook the suggested recipe. We intend to support the feature of recipe search based on ingredient amount.

Concepts

Scavenge

Purpose: Scavenge allows users to search for recipes given ingredients they already own, eliminating the need to go and buy more ingredients.
Operational Principle: The search takes into account all of the current ingredients in the Pantry (see below), their respective amounts, and a user-specified serving size. The results will be recipes containing a subset of Pantry ingredients with the constraints of ingredient amounts and serving size desired.

Flexible Search

Purpose: This gives the user more flexibility in terms of recipes that the user could make, and the user is not solely constrained by ingredients they already own.
Operational Principle: If no recipes containing only the Pantry ingredients exists, users can also specify how many “extraneous” ingredients (ingredients the user doesn’t currently possess) they allow in the recipe. The search will return recipes that include up to the max number of allowed “extraneous” ingredients.

Pantry

Purpose: The pantry items represent the items the user currently owns so they can be used to search for recipes.
Operational Principle: The user inputs ingredients they currently own into the Pantry through an autocomplete list that the user must choose ingredients from. This mitigates the risk of the user typing something incorrectly or inputting an ingredient that is too generic to be found in a recipe (e.g. cheese versus feta cheese or blue cheese).

Pantry Sharing

Purpose: Pantry sharing allows users to combine their pantry ingredients with other users’ pantry ingredients in order to cook together.
Operational Principle: The user goes to his list of Sous Chefs (see below) and chooses who to send a Pantry Sharing request to. If the Sous Chef accepts, the user will be able to view all of the Sous Chef’s ingredients in his own pantry and use them to Scavenge for recipes.

Sous Chefs

Purpose: Becoming Sous Chefs with other users allows Pantry Sharing. This allows users to only share Pantries with people they know.
Operational Principle: The user sends a Sous Chef request to another user who he would like to share pantries with.

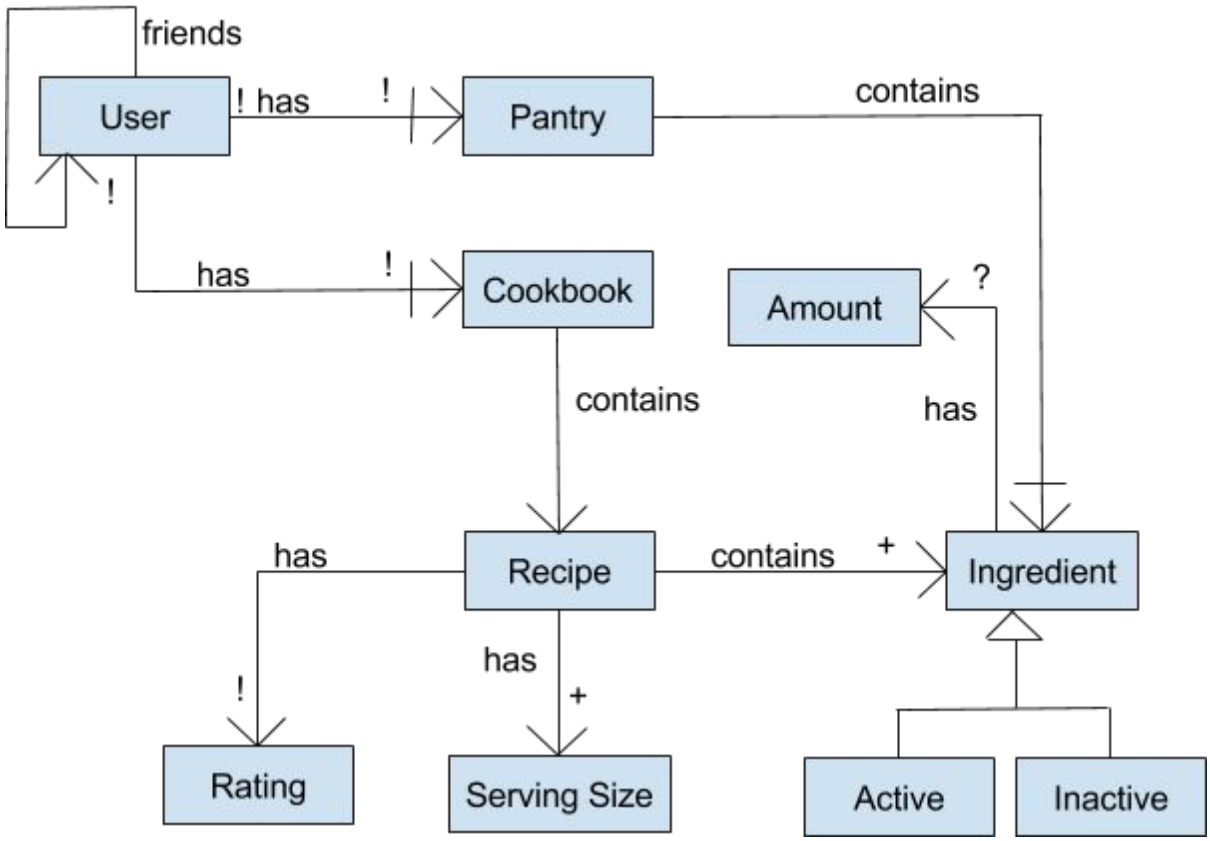
Rating Recipes

Purpose: Ratings allows users to make a more informed decision about which recipe they choose.
Operational Principle: Users rate recipes, and the recipes will be displayed in order from highest rating to lowest rating.

Cookbook

Purpose: The Cookbook allows users to view recipes they enjoy at a later time without the risk of losing it.
Operational Principle: The user clicks the heart by the recipe to save it to his Cookbook.

Data Models



Textual Constraints:

1. An ingredient with no specified amount is assumed to have an unlimited amount.
2. The amount of an ingredient that is in a recipe depends on the serving size specified for the recipe.
3. Users cannot be friends with themselves.
4. The amount and the state of an ingredient are pantry-specific. Pantry-specific ingredients have an ingredient field, an active/inactive field, and an amount field.

Explanations:

1. Ingredients contained in recipes are always active. This is due to the fact that the active/inactive field of a recipe only matters when users are searching for recipes
2. A recipe has both a serving size and all ingredient have amounts. However, we scale all recipes to conform to a custom serving size when searching. It is easier to think of these two fields as a ratio (how much of each ingredient per one serving size).

Security Concerns

- Security Policy for ImHungry.com
    - Users can only search recipes if they are logged into an account
    - Users can only favorite recipes if they are logged into an account
    - User can only view their own favorite recipes in their “Cookbook”
    - User can view the ingredients in their own “Pantry”
    - User can view ingredients in another user’s pantry only if they are currently sharing their pantries
- Threat model

    - What can an attacker do?
        - an unauthenticated user could issue a request to either add, remove, or edit ingredients in another user's "Pantry"
        - an unauthenticated user could issue a request to favorite or unfavorite recipes in another user's "Cookbook"
        - an unauthenticated user could view and search recipes without logging into an account first
        - an authenticated user could edit/add/delete recipes that can be searched for in our system.
    - Combat Strategy!
        - use hidden tokens so a malicious site cannot use a user's cookies to access and mutate another user's account. This protects against CSRF attacks.
        - manually escape text content that is presented to users in our application (such as recipe instructions) so that javascript scripts can not be injected and the malicious scripts executed on the client side. This would protect against XSS attacks.
        - sanitize all text inputs by casting to string to prevent hacker from issuing requests to our system/database. This would protect against code injection attacks.
    - Worst case scenario
        - if a user hacked into ImHungry.com, they can edit the ingredients in another user's Pantry
            - This threat is not critical because there is no sensitive information that is lost or changed
        - if a user hacked ImHungry.com, they can delete/add favorite saved recipes in another user's Cookbook
            - This threat is also not critical. Although the user's whose account was hacked may lose their saved recipes, these recipes can be found again and no sensitive information is stolen by the hacker.
        - if a user hacked ImHungry.com, they can view recipes without being logged in or authenticated
            - This threat is not critical because no other user is harmed or affected by a hacker viewing the recipes. No user information is gleaned by the hacker.
        - if a user hacked ImHungry.com, they could change the recipes (by adding new recipes, deleting or editing existing recipes)
            - This threat is moderately minor. Regular users would be inconvenienced because they could receive faulty recipes, or be no longer able to access certain recipes. However, no sensitive information is lost and there is no irrevocable damage done to the users or system.

Standard Web Attacks and How to Mitigate

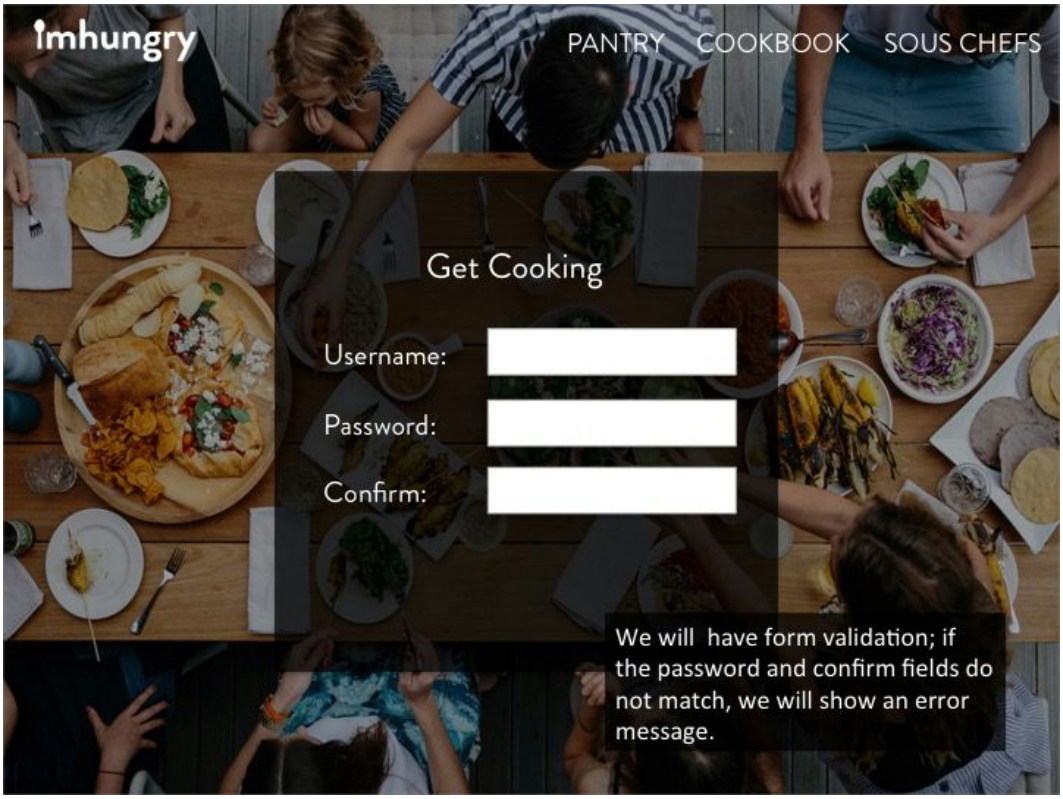
- Code Injection

    - Attack description
        - instead of submitting text into a text input area, a hacker inputs a JSON object. When the JSON object goes to the server side, instead of being treated as a string, it is treated as a database request. This allows the hacker to access your database, viewing, mutating, or deleting data to which the hacker should not have access.
            - ex: hacker inputs a JSON object in the field where you are supposed to type an ingredient
    - How to address
        - sanitize all user inputs by casting (forcing) the input to be treated as a plain string.
            - For example, in your code, you can use text = JSON.stringify(inputText)
- Cross site scripting attack - XSS
    - Attack description
        - hacker inserts malicious content, or malicious scripts, that is delivered to the client-side web browser. Then this malicious javascript is executed, giving the hacker access to user browser data.
    - How to address
        - escape all string inputs in our application so that all inputs will be treated as text.
            - Again, we can use JSON.stringify()
            - if using ejs, can use <%= >
            - if using handlebars, can use {{{...}}}
- Cross site request forgery - CSRF
    - Attack description
        - hacker uses browser trust to transmit their own requests to a web site or application.
            - for example, when you have two window tabs open, one with your ImHungry.com account and another with a malicious site
                - the malicious site uses the cookie from your logged in account to access your Pantry and change the ingredients in your pantry
    - How to address
        - Use hidden tokens by adding a new field or token that is generated every time the page is requested
        - associate the token with the specific session
        - middleware checks that csrf token matches request and session

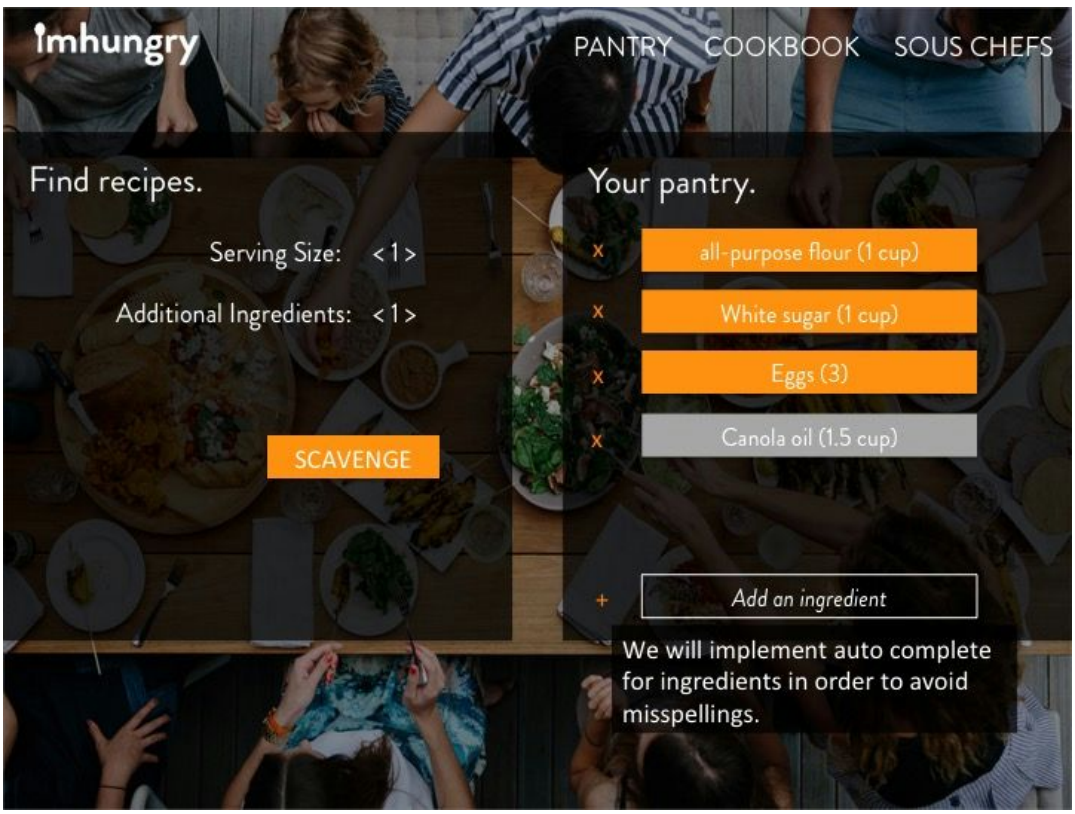
User Interface

Wireframes

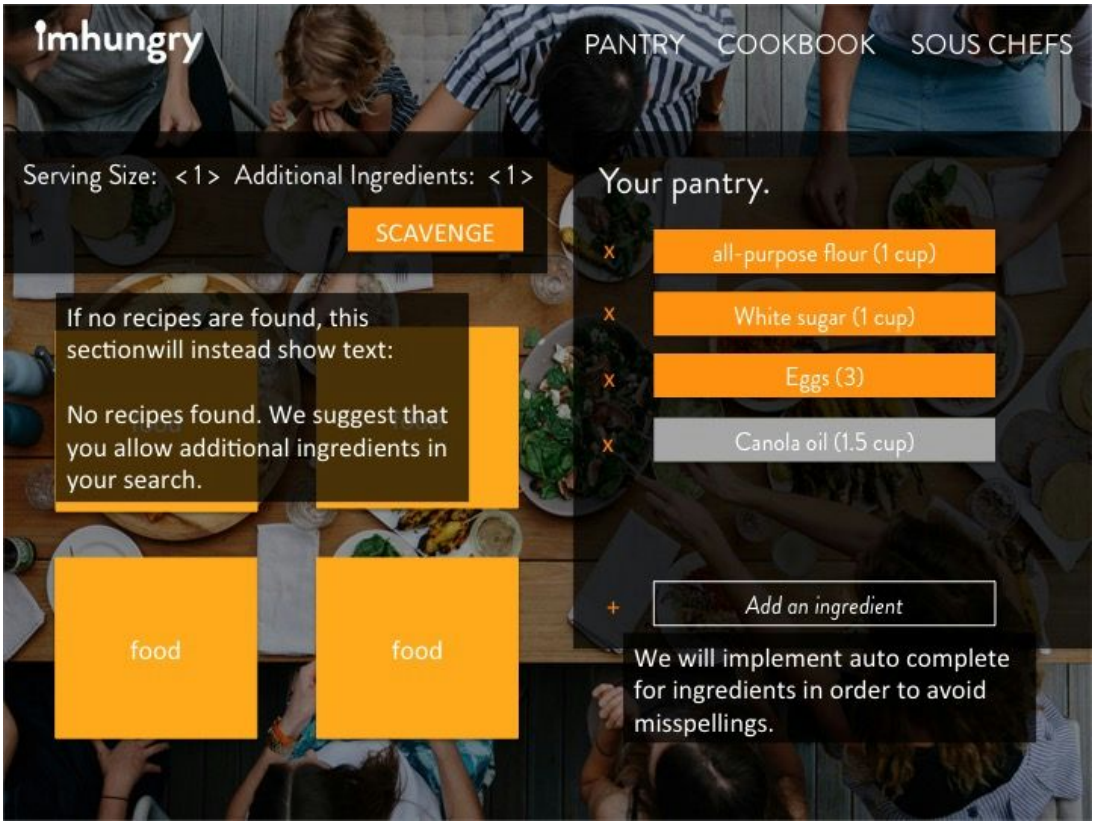
Login / Sign in



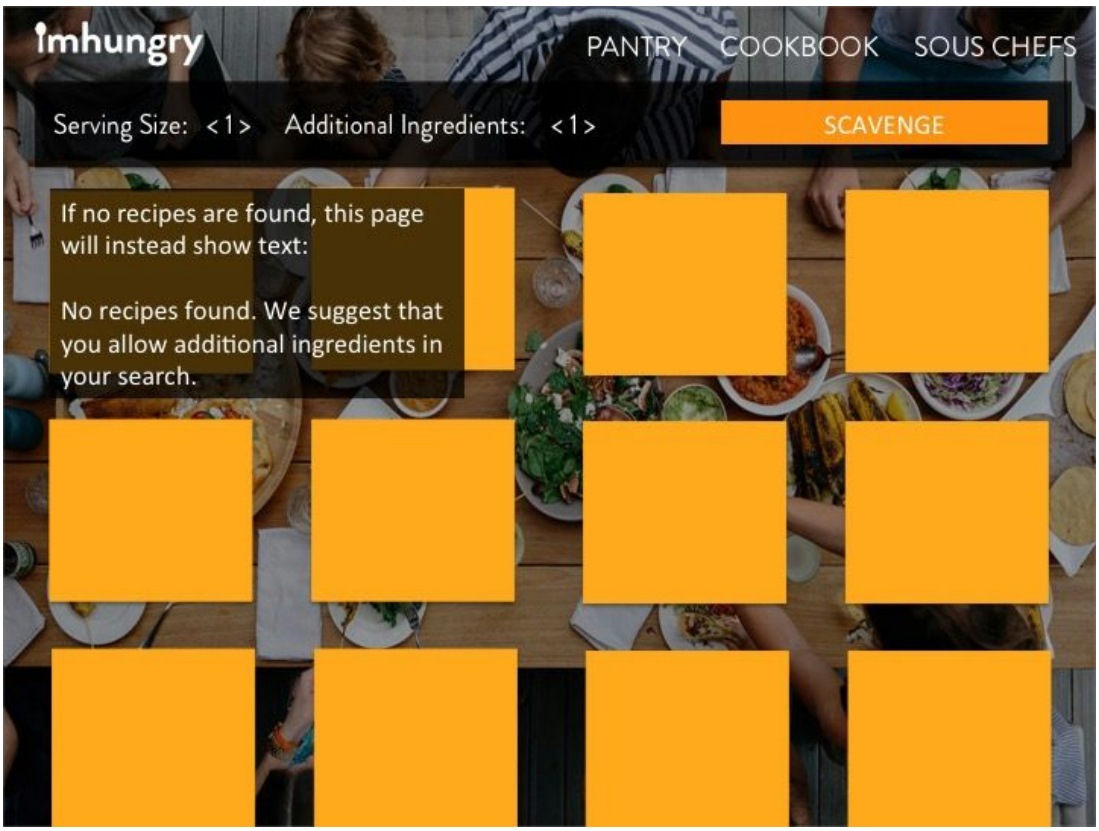
Logged in Homepage



Home page with search results

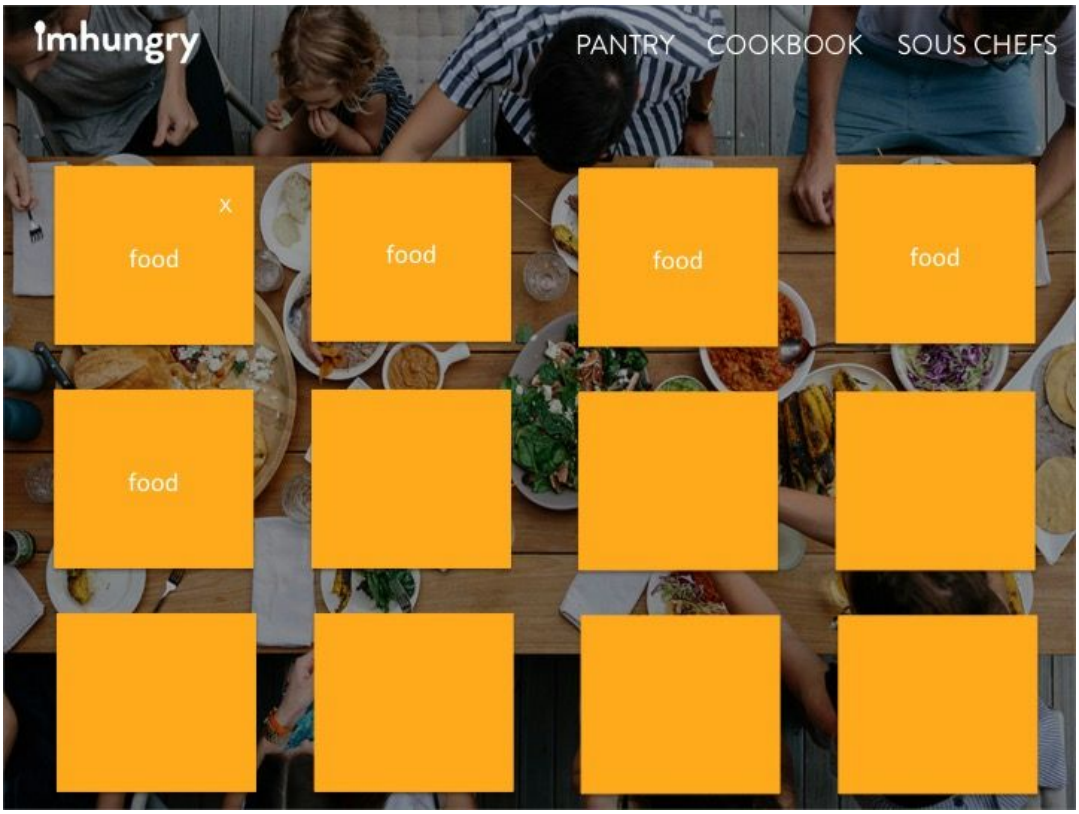


Home page with expanded search results

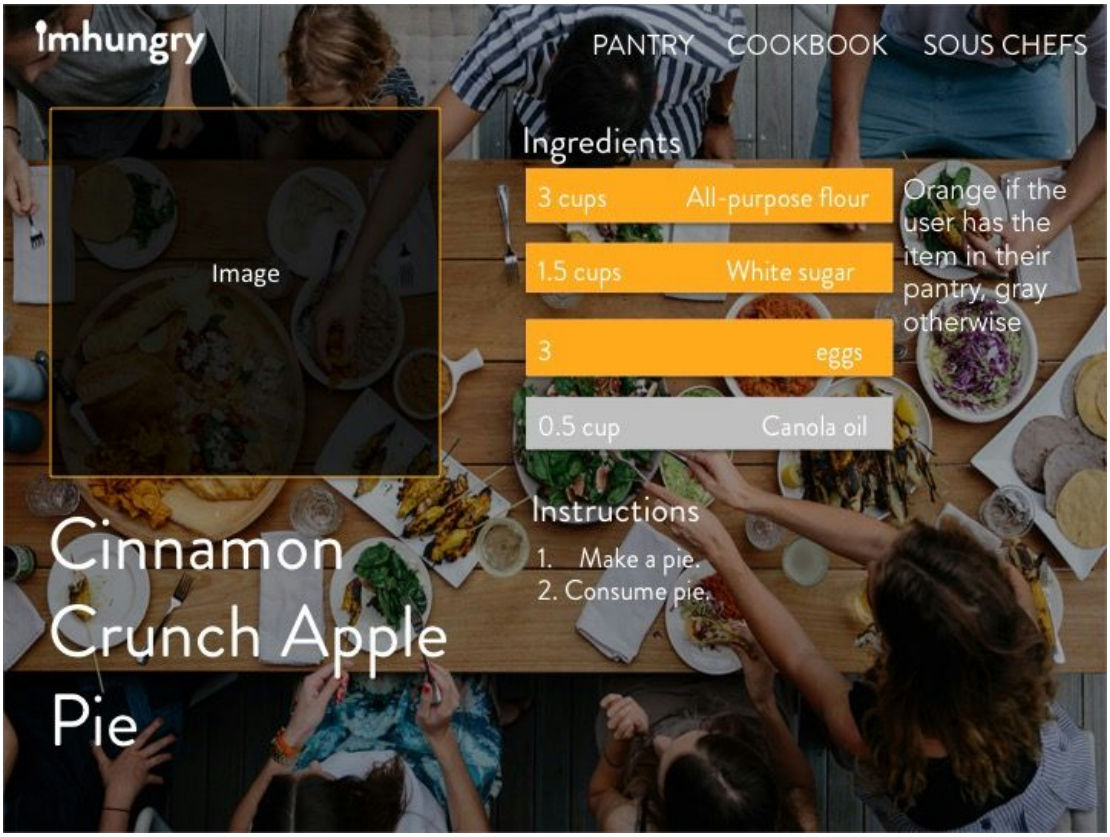


Sous Chefs

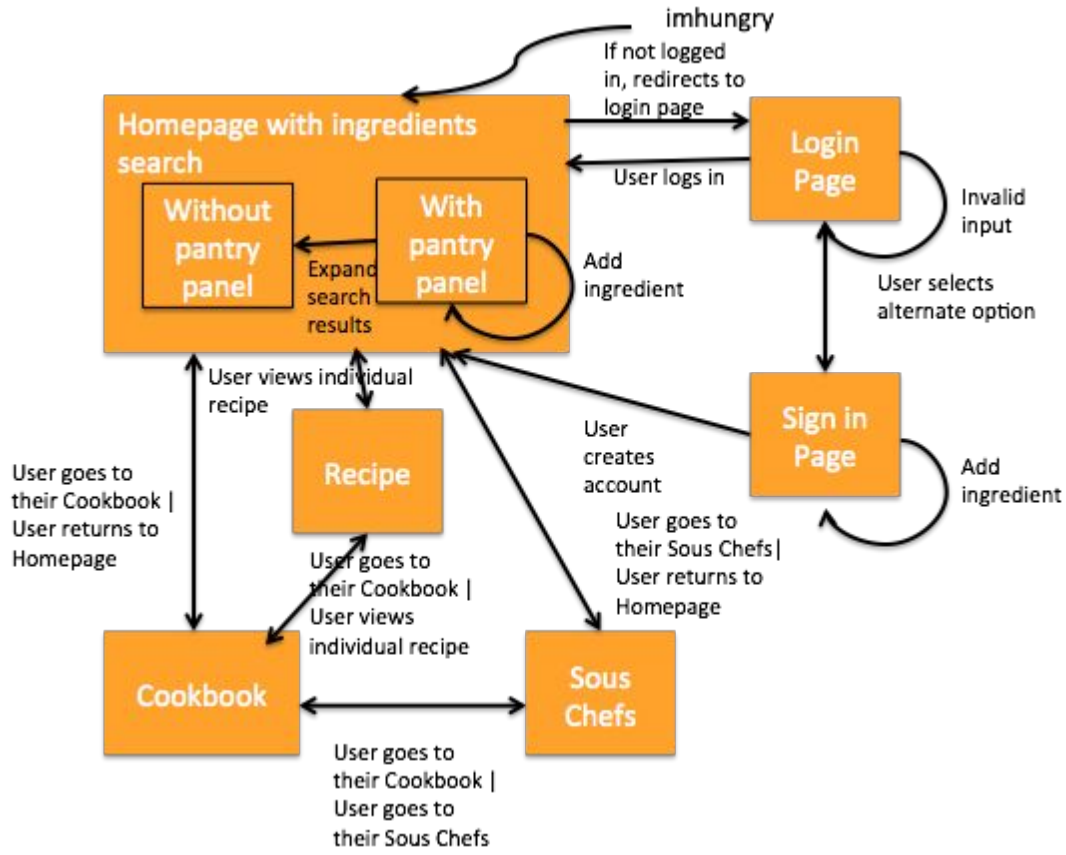
Cookbook



Recipe view


Imhungry
PANTRY
COOKBOOK
SOUS CHEFS
Image
Ingredients
3 cups All-purpose flour
1.5 cups White sugar
3 eggs
0.5 cup Canola oil
Orange if the user has the item in their pantry, gray otherwise
Instructions
1. Make a pie.
2. Consume pie.
Cinnamon Crunch Apple Pie

Transition Diagram



Design Challenges

One of the challenges we faced was the problem of users misspelling an ingredient or perhaps inputting an ingredient that was too general for a recipe (e.g. “oil” instead of “olive oil” or “coconut oil,” which are used in very different ways). A misspelled ingredient would definitely not return anything from our database. Furthermore, a generic search for “oil” may not return any recipes either though there are many recipes using different types of oil. This challenge was factored into our design such that each ingredient present in the recipes is an ingredient object in the database. All ingredients that the user inputs must match to an ingredient object, and this is completed using autocomplete. When adding ingredients, the user must choose an ingredient from our autocomplete list. This will eliminate mismatched string errors that would hinder a user from finding the optimal recipe in the search process.

Another possibility is that the search returns results where the recipes require a higher amount of an ingredient than the user owns, and the user won’t be able to make the recipe. We came up with two design ideas to mitigate this. First, the user can specify the amount of each type of ingredient he owns. If no amount is specified, the amount is assumed to be unlimited. This will allow the search to only return recipes where the user has enough of each ingredient. However, there’s the problem where a recipe calls for a higher amount of ingredients because it serves ten people. In this case, this recipe won’t be returned because the user technically doesn’t have enough ingredients. To account for this,

we also decided to allow the user can specify a serving size. This way, the search will return recipes that would have been otherwise not included in the results, and the returned recipes will be scaled down to match that serving size.

We wanted users to be able to view Pantry items while making their search, but we also wanted to make sure there was a clear separation of concerns between Scavenge and Pantry. We first had the two concepts in the same view, but that made it confusing whether the search bar was the search for recipes or to add ingredients to the Pantry.

We then decided to split the two into separate web pages. However, with this design, users would not be able to see Pantry items when conducting the search, and it also gives the user less flexibility in terms of choosing which ingredients from their Pantry they would like to use.

We then settled on creating a Pantry sidebar on the main search page where users could see their ingredients and specify which ingredients from the Pantry and in what amounts they would like to use them. However, this rendered the separate Pantry web page obsolete as almost all functionality except adding and deleting Pantry items was now able to be completed from the sidebar. As a result, we eliminated the separate Pantry webpage and settled on a Pantry sidebar on the main search page. The sidebar can be collapsed in the case the user doesn't want to see his Pantry all the time. This way, all Pantry functionality is centered in one location, and the user will easily be able to see and choose exactly which ingredients he wants in the search, giving the user much more control.

Another challenge was to figure out how to incorporate the concept of Pantry Sharing. One on one Pantry Sharing was straightforward enough, but the main challenge was the case when User A shares their Pantry with multiple people. If User A shares with User B and User C, should B and C automatically be shared as well? We decided against this because this web can escalate very quickly, and there's a chance that User B and C are not Sous Chefs, which would violate our concept of requiring users to be Sous Chefs before being able to Pantry Share.

Another design idea we considered was to create "cooking groups" where users can create groups to join and all Pantry Share together. However, this can also get complicated in terms of which group the user is currently active in because technically, a user can't cook with two separate groups of people at once.

We finally settled on the most straightforward version of multiple Pantry Sharing. If User A is sharing with User B and User A is sharing with User C, User A's Pantry will display all of User A, B, and C's Pantry ingredients. However, B's Pantry will only display A and B's Pantry, and C's Pantries will also only display A and C's Pantries. B and C have no connection through A, so if B and C want to view each other's Pantries, they must share with each other as well. This simplified design eliminates all the complications caused by the previously considered designs.