**CS 307 Team 3 – Kitchen Support**

**Design Document**

**Garrick Buckley, Mason Herhusky, Jack Horton, Garrett Lewis, Graham Park, Darrell Shi**

**Table of Contents**

|  |  |
| --- | --- |
| **Purpose** | **3** |
| **Design Outline** | **3** |
| **Design Issues: Functional** | **5** |
| **Design Issues: Non-Functional** | **6** |
| **Design Details** | **7** |

**Purpose**

Novice chefs tend not to know what they can make with the ingredients that they have. This leads to wasted food and more money spent on eating out. Our system will be different from existing products by focusing on finding unique dishes very quickly based on what you have in your house right now.

The following are the core functional areas of our system:

1. Users should be able to find recipes easily based on…
   1. Manually input ingredients and cuisine preferences
   2. Existing ingredients in the user’s kitchen
   3. Suggestions based on the user’s previous activity and input
   4. Limiting ingredients based on allergies
   5. Recipe difficulty, cost, and time required to prepare
2. Users can ‘swipe left’ on a recipe that they are not interested in, or ‘swipe right’ on a recipe that they do find interesting.
   1. Recipes that the user finds interesting are saved per-session, and can be reviewed once the user feels that they have looked through enough recipes.
   2. 0The way that the user rates recipes will be tracked and saved, and will influence what recipes will be shown to them in the future.
3. Users can keep an inventory of the food that they have in their house by either manually inputting ingredients or by scanning each ingredient’s barcode
4. Users can manage their account and personal preferences in either the app or the website, and can sign up using social services such as Facebook and Google
5. Users can view their recommended recipes on dedicated Android and iOS apps, as well as online at <https://www.kitchen.support>

**Design Outline**

**General Priorities**

1. Portability: We would like this app to be able to be used on multiple platforms. With the help of Xamarin, it will be easy for us to reuse large portions of the mobile app code. We will also structure the web app to use the same API as the mobile apps, which will keep us from writing two different server backends.
2. Usability: Because this application will be entering an already competitive recipe app marketplace, making the application easy and fun to use will be a key differentiator between us and the competition.
3. Performance: Our priority with this app is to connect users with recipes in the quickest and simplest way possible. We need to make sure that our request processing is fast and that our apps are responsive to allow the user to see the information that they need as soon as possible

**Design Decisions**

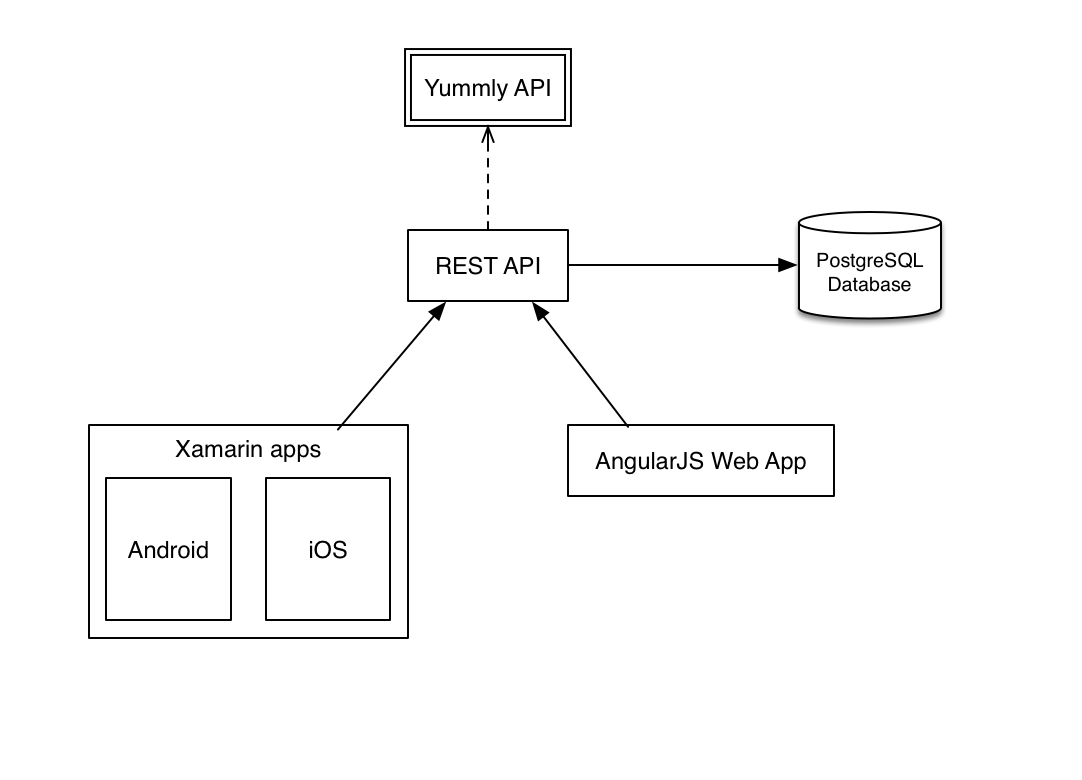
Our service will be focusing mainly on mobile users, but we will provide a web interface for those who are unable to access it through a mobile device. We’re focusing on the mobile application because it allows us to take advantage of the camera for scanning barcodes on ingredients, and it’s easier for our end-users to use a mobile application while you are cooking in the kitchen. Our application will use the Client-Server model where our API allows clients on any platform to interact with our application. Our API will be connected to a database that stores user information and cached recipe data. Our web application will be a Single Page Application written using AngularJS.

1. Back-end
   1. Purpose: act as a RESTful API so that clients on any platform can get generic data from our service
   2. Implements as much app logic as possible so that the experience is similar across platforms and we do not have to write custom, duplicate code for each client.
   3. Stores user preferences and generates customized suggestions that travel with the user across platforms.
2. Front-end
   1. Web
      1. Purpose: allow users who do not have or are otherwise unable to use their mobile device to still use our service.
      2. Acts as a Single Page App, where all of our app logic will be held on the client in order to eliminate our need for a separate web server and API server
      3. Uses Material Design to complement the styling of our Android app and give users a cohesive user experience.
   2. Mobile
      1. Purpose: allow users to access our service through their iOS or Android phones.
      2. Built with Xamarin to reach as many platforms with as little code as possible.
      3. Native styling will make our application feel as if it were custom-made separately for both iOS and Android.
   3. Database
      1. Purpose: allow users to save their pantry, suggestions, and preferences in the cloud to easily be accessed on any device.

**System Interactions**

Our application has an almost entirely uni-directional flow. Clients make requests to our REST API, which in turn will request information from Yummly’s REST API. We do not plan on using any socket communication, nor do we plan on implementing two-way data binding within our applications, as these both make the overall system much more complex. Authentication is provided by our REST API using a token scheme, with user accounts backed by our PostgreSQL database. In addition to holding user information, our database will cache the items that we fetch from Yummly, which will let us more easily provide suggestions to our users.

**System Overview**



**Design Issues**

**Functional**

**Issue**:How should our mobile applications be developed?

1. PhoneGap/Ionic/Titanium
2. Xamarin
3. Native

**Decision**: We decided to use Xamarin. Option 1 was ruled out due to our team’s lack of experience using Javascript, and option 3 was ruled out due to concerns over code duplication and our lack of knowledge of Swift or Objective-C for iOS development. All of our team members felt comfortable using C#, and as such, Xamarin was the best fit.

**Issue**: When should we to require users to log in?

1. Require login for any interaction
2. Require login only for user-related actions, such as saving a recipe
3. Hybrid solution

**Decision**: We ended up going with a hybrid solution, where mobile apps require login for any interaction, while the web app will allow the user to browse the site and search for recipes without having to log in. The web app will still require an account for pantry management, recipe saving, and personalized recipe suggestions. Our experience is vastly superior on mobile when you have an account, because entering search parameters for recipes manually on each search is very hard on mobile devices. As such, we felt it was worth forcing users to sign in before browsing to eliminate these manual processes as much as possible. On desktop, thanks to Google searching and the ease of entering search parameters with a mouse and keyboard, we opted to not force logins for basic interactions. For instance, it is very likely that someone could be looking for a recipe on Google or Bing and land on our site, and we could lose that user if we force them to sign up just to view the recipe – this is not a concern for mobile apps.

**Non-functional**

**Issue**: What database should we use to store the user and recipe information?

1. MySQL
2. PostgreSQL
3. MongoDB/other NoSQL

**Decision**: We chose to use PostgreSQL. MongoDB or any other NoSQL database (like CouchDB or DynamoDB) all would have worked well for our application, especially because they would have allowed us to easily store the JSON response that we were getting back from Yummly. However, we have a lot of relational aspects to our data that link users and recipes, and NoSQL databases have varying levels of support and easy of implementation of true relationships. MySQL would have forced us to map out all of the relations presented in the JSON response from Yummly if we wanted to be able to query that data, and this would have involved making many more data and relation tables. PostgreSQL was a very good middle ground, as it allows us to form proper relationships between our user and recipe data, while also allowing us to store, index, and query the raw JSON we get back from Yummly without having to map it to SQL tables.

**Issue**: Where would we pull recipe information from?

1. Yummly
2. Spoonacular
3. Open Source Food
4. Food2Fork
5. Input recipes manually into our database
6. Hybrid approach

**Decision**: The Yummly API was the best fit for our application. It pulls recipes in from many different recipe sources, including large, well-known food sites like AllRecipes.com and Epicurious. They also have generous academic plans that gave us plenty of access to their service for free. Overall, we found that Yummly had more features and a more robust API than all of the other options. We chose against maintaining our own database of recipes due to the difficulty and tediousness of building up a large-enough catalog of recipes. Finally, we considered pulling recipe information from multiple sources, however Yummly already has so many upstream sources itself that we believed we would run into a lot of duplicate recipes, where two or more APIs would have the same recipe listed. These duplicates could also be difficult to detect, depending on the structure of the responses from each different API.

**Issue**: How do we scan bar codes?

1. Use online API
2. Use platform-specific libraries
3. Use a pre-built Xamarin component

**Decision**: A pre-built Xamarin component best fits our desire to reduce code duplication and keep the app performant. Using an online API would require sending the picture of the bar code to an online service, which would take a significant amount of time (especially over mobile networks) and would quickly consume end-user’s data, many of whom likely have plans with data limits. Platform-specific APIs may have very good user experience and performance, but would result in significant code duplication. We were able to find a pre-built Xamarin component that encapsulates all of the functionality we need into a cross-platform, performant solution, which provided us with the best of both worlds.

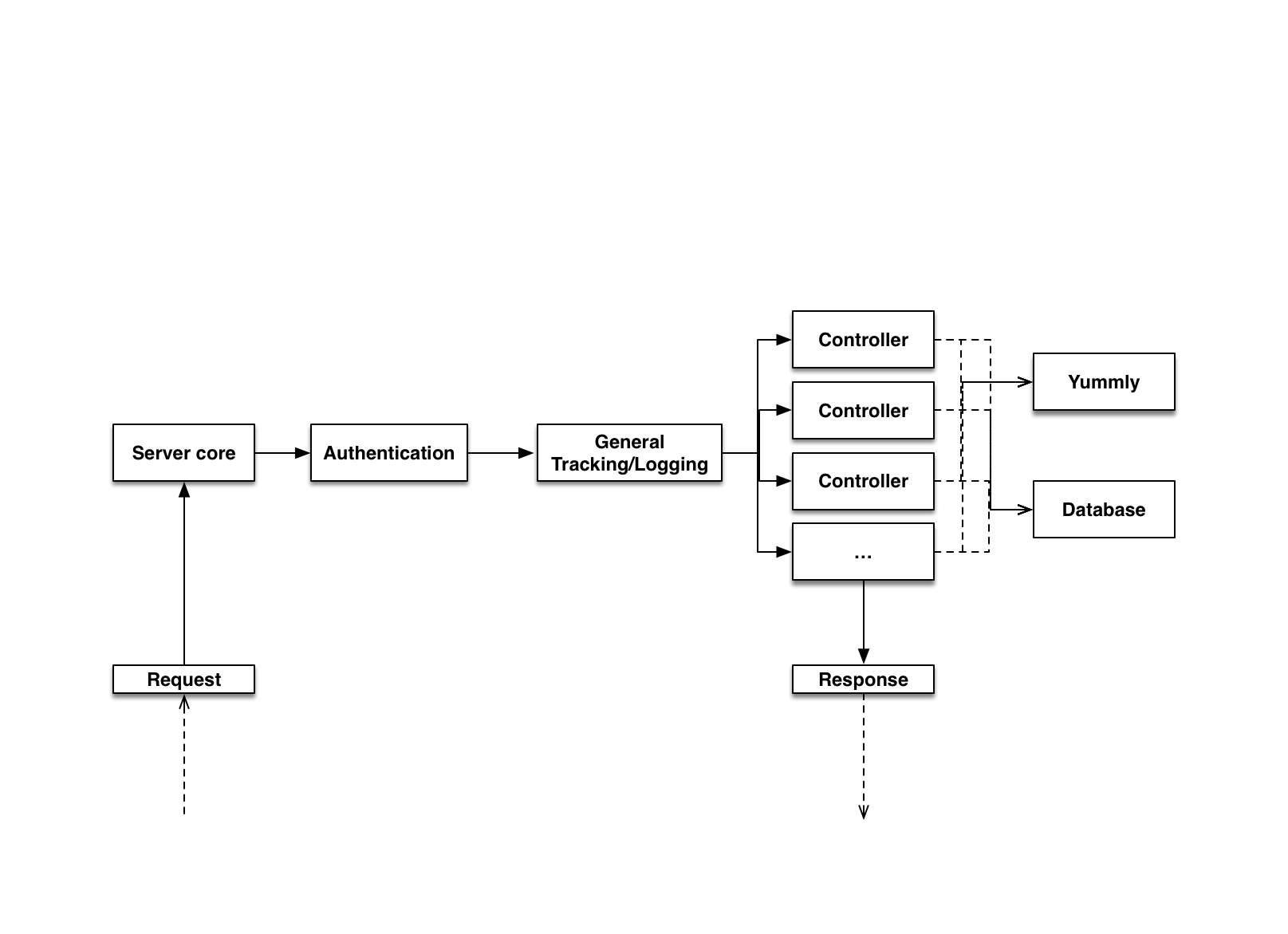
**Issue**: What language should we use to make our REST API?

1. Node.js using Express and Knex.js
2. C# using ASP.NET Web API and Entity Framework

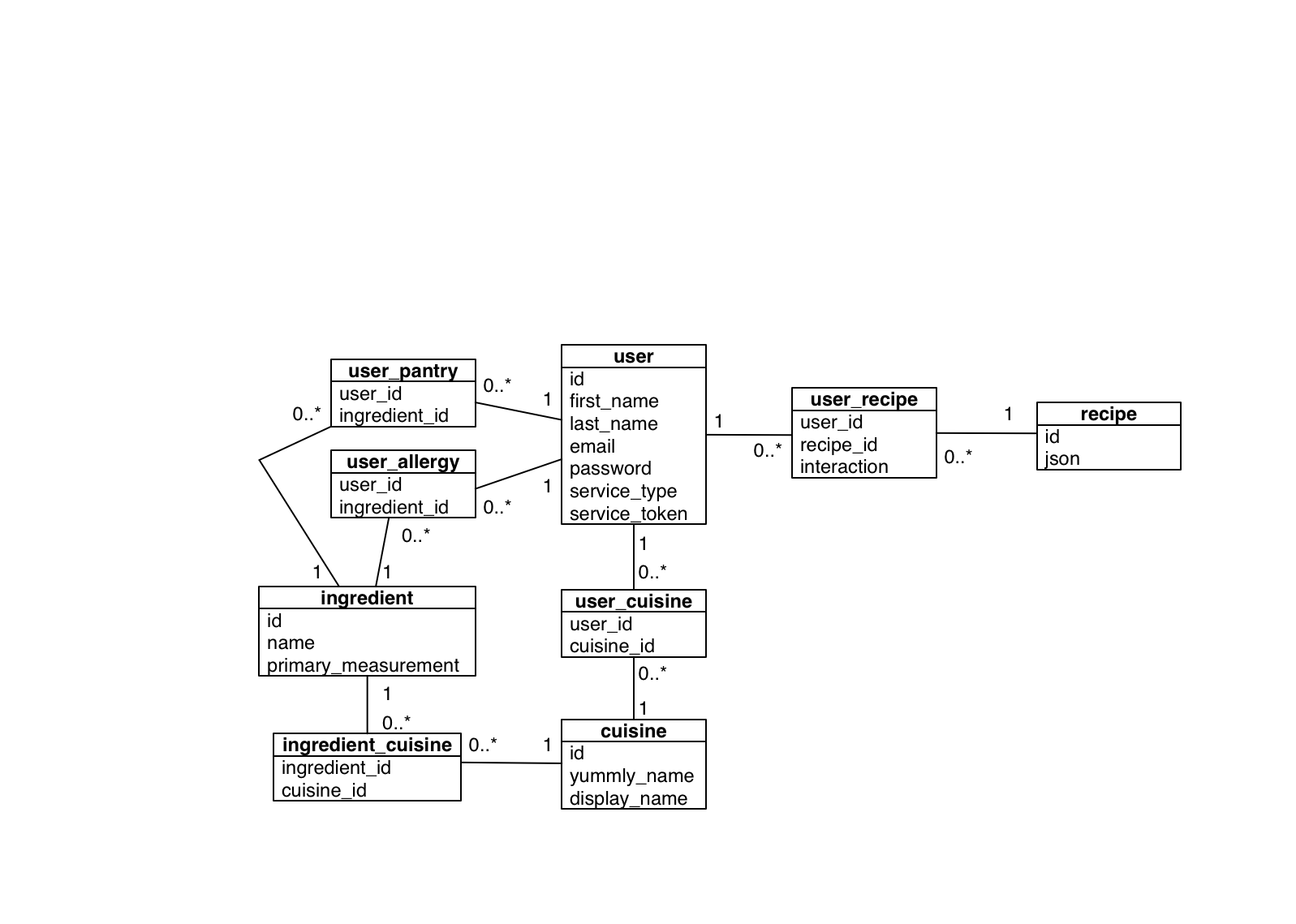
**Decision**: We ended up selecting Node.js for our REST API platform. Both options were very good and provided a lot of benefits, however we have two team members with extensive experience using the Node stack, and none who had ever used Web API or Entity Framework.

**Design Details**

**REST service component diagram**



**Data diagram**



**Class Descriptions**

**User**

A user represents the data related to the user of the application, including name, username, email, password hash, etc.

**Cuisine**

The style of dish (American, Italian, Chinese, etc). This is provided by the Yummly API and will be another parameter to be used when searching for recipes.

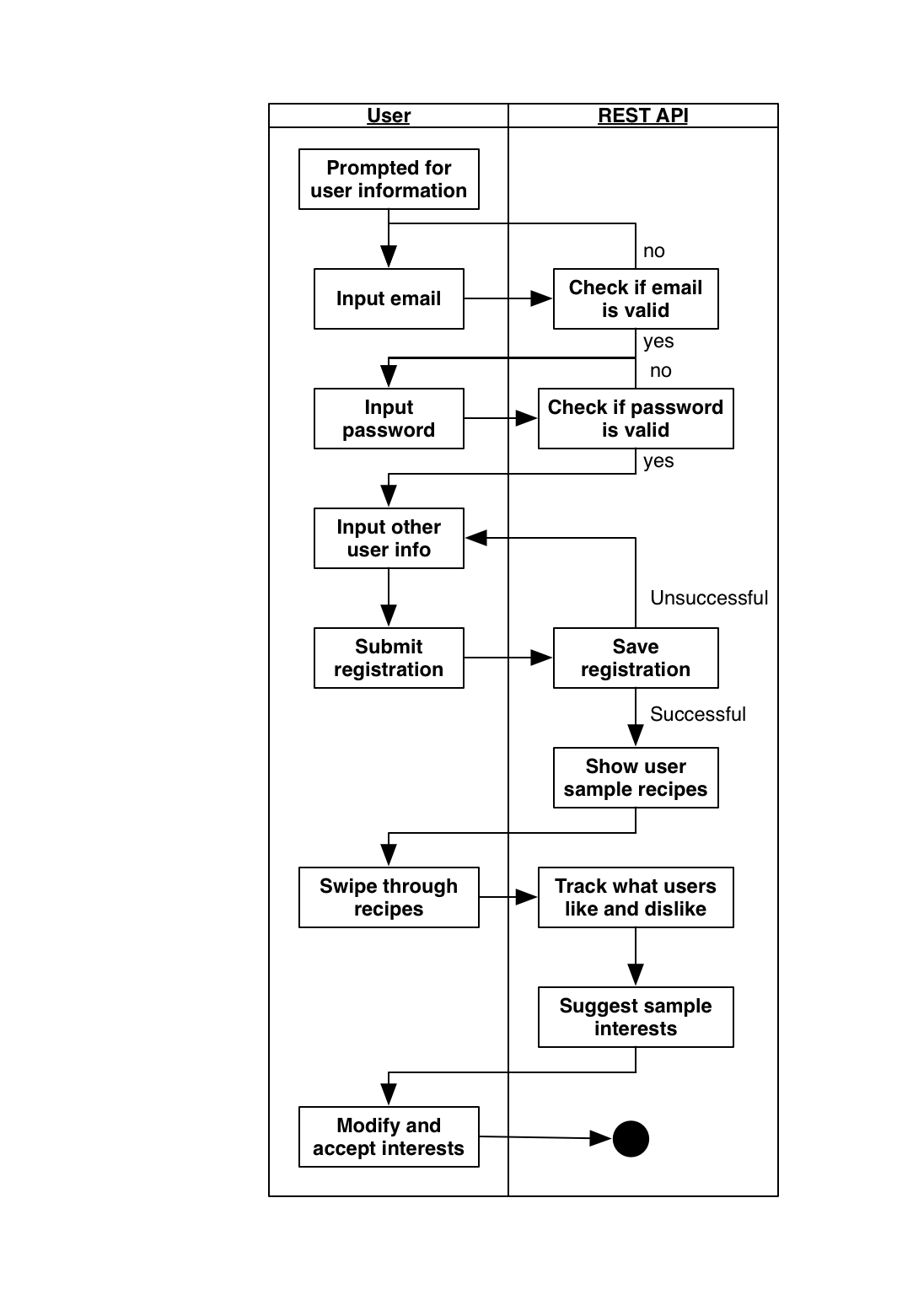
**Ingredient**

An ingredient represents the data related to the user’s pantry items and allergies that are used to whitelist and blacklist ingredients, respectively.

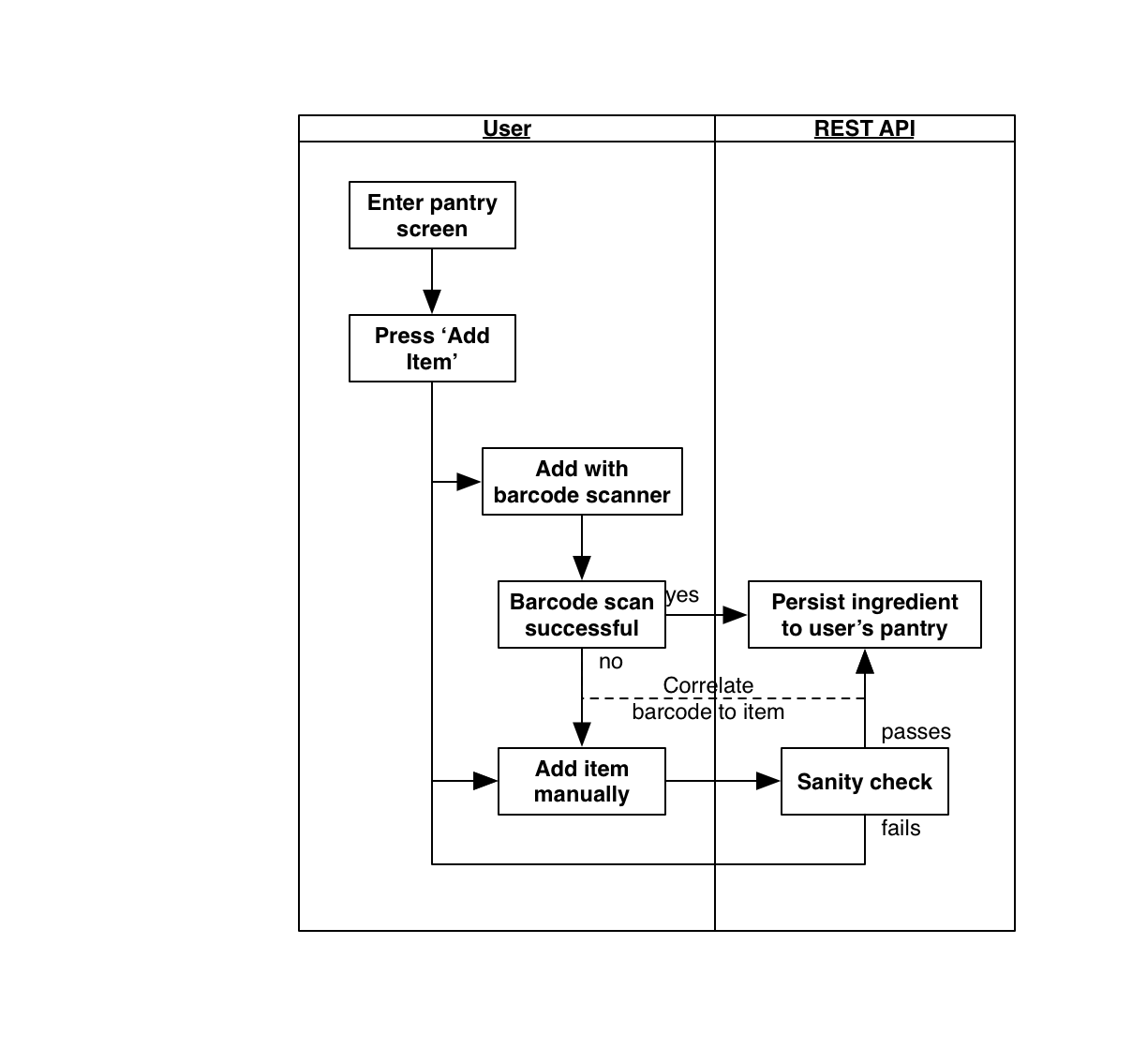
**Recipe**

The response from the Yummly API for a given Get Recipe call. This data is in an unstructured JSON format that we are able to query against thanks to PostgreSQL’s JSON support.

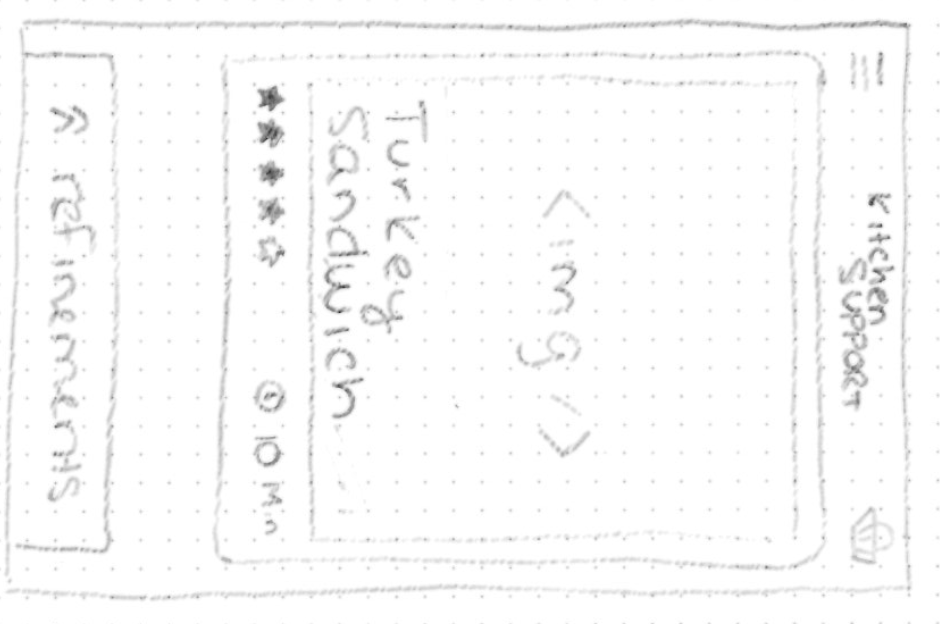
**Register Flow**



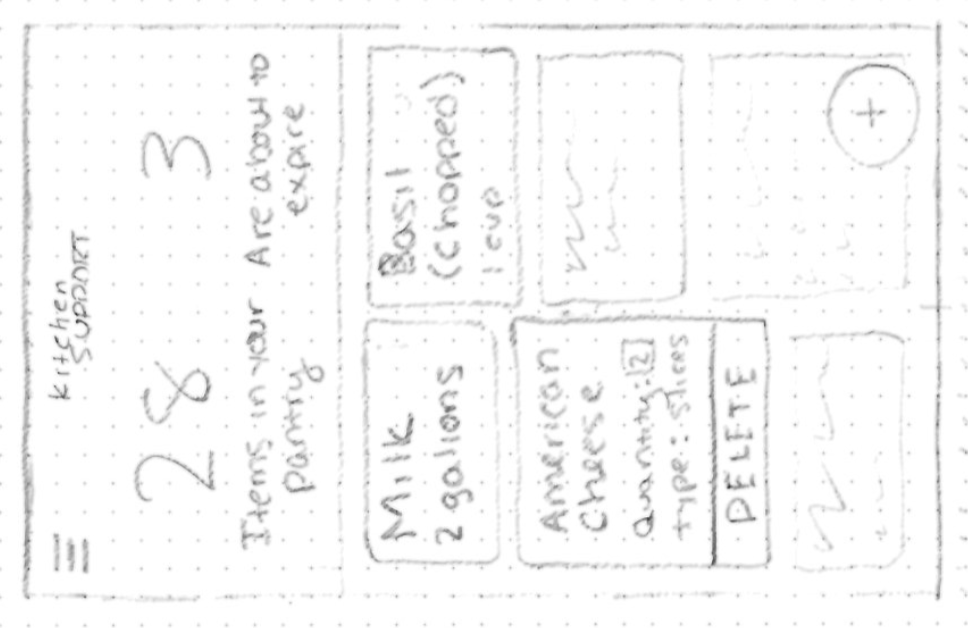
**Pantry Management Flow**



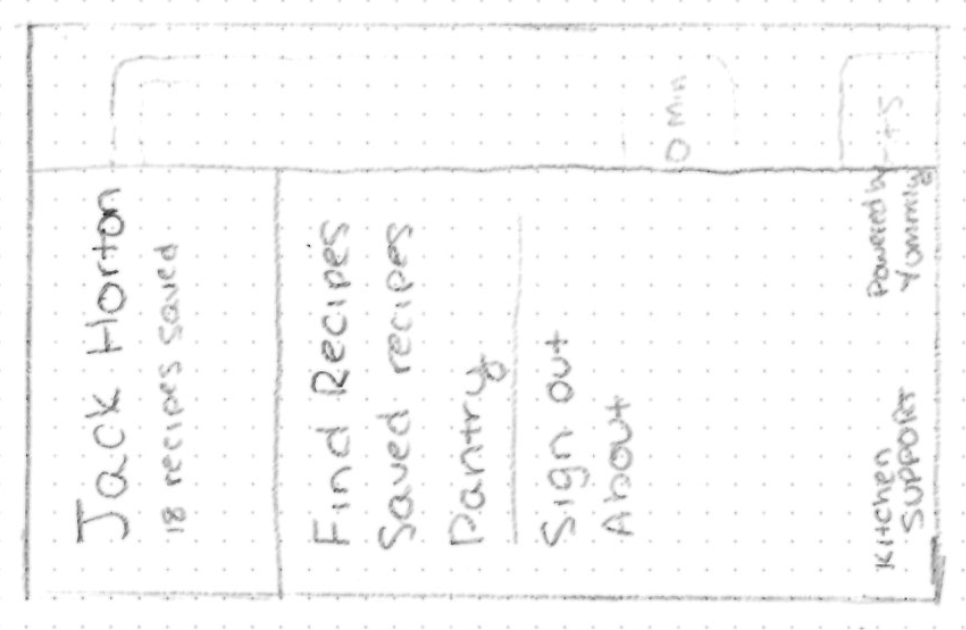
**Starting screen after login**



**Pantry management**



**Hamburger menu**



**Web home page, before logging in**

