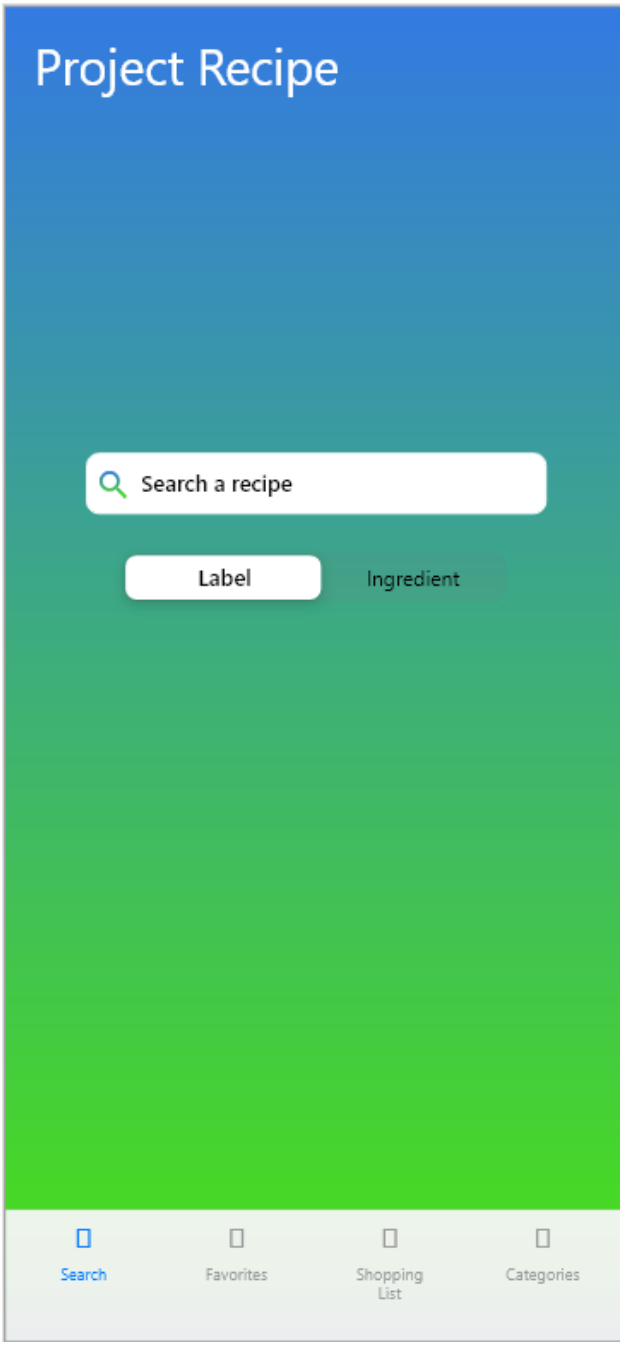
Duy Nguyen, Nafi Mondal, Tahmid Hannan

Group 6

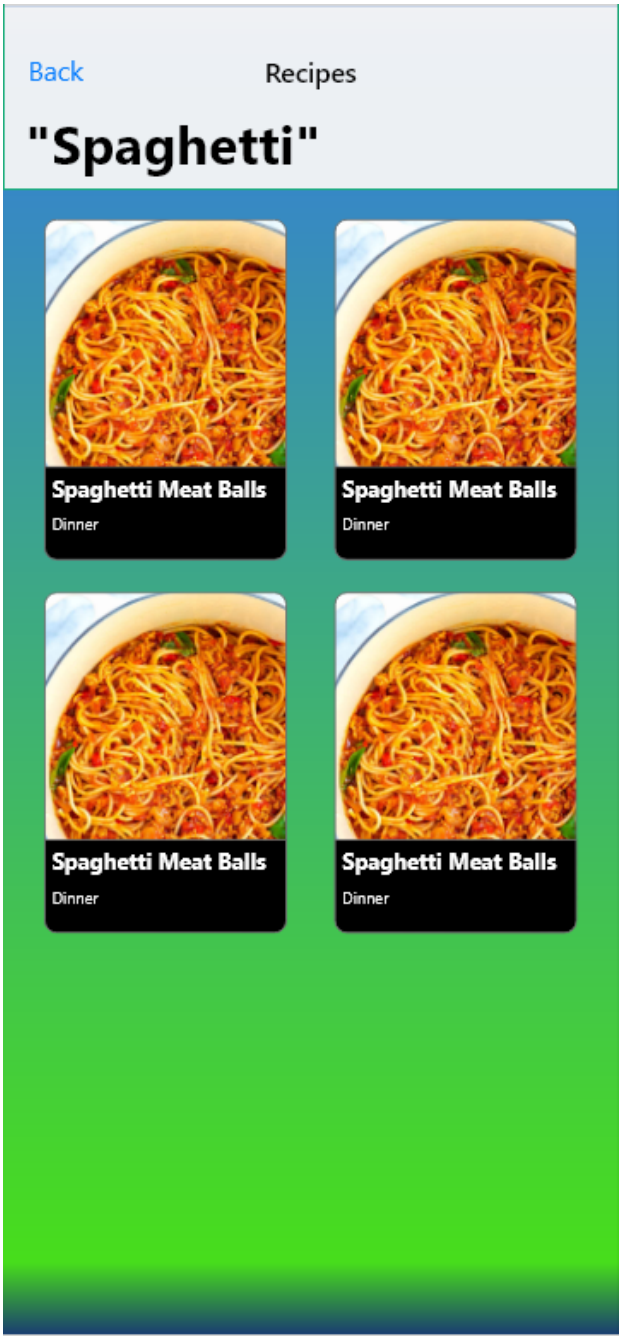
Recipe App Report

User-Interface (Duy Nguyen)

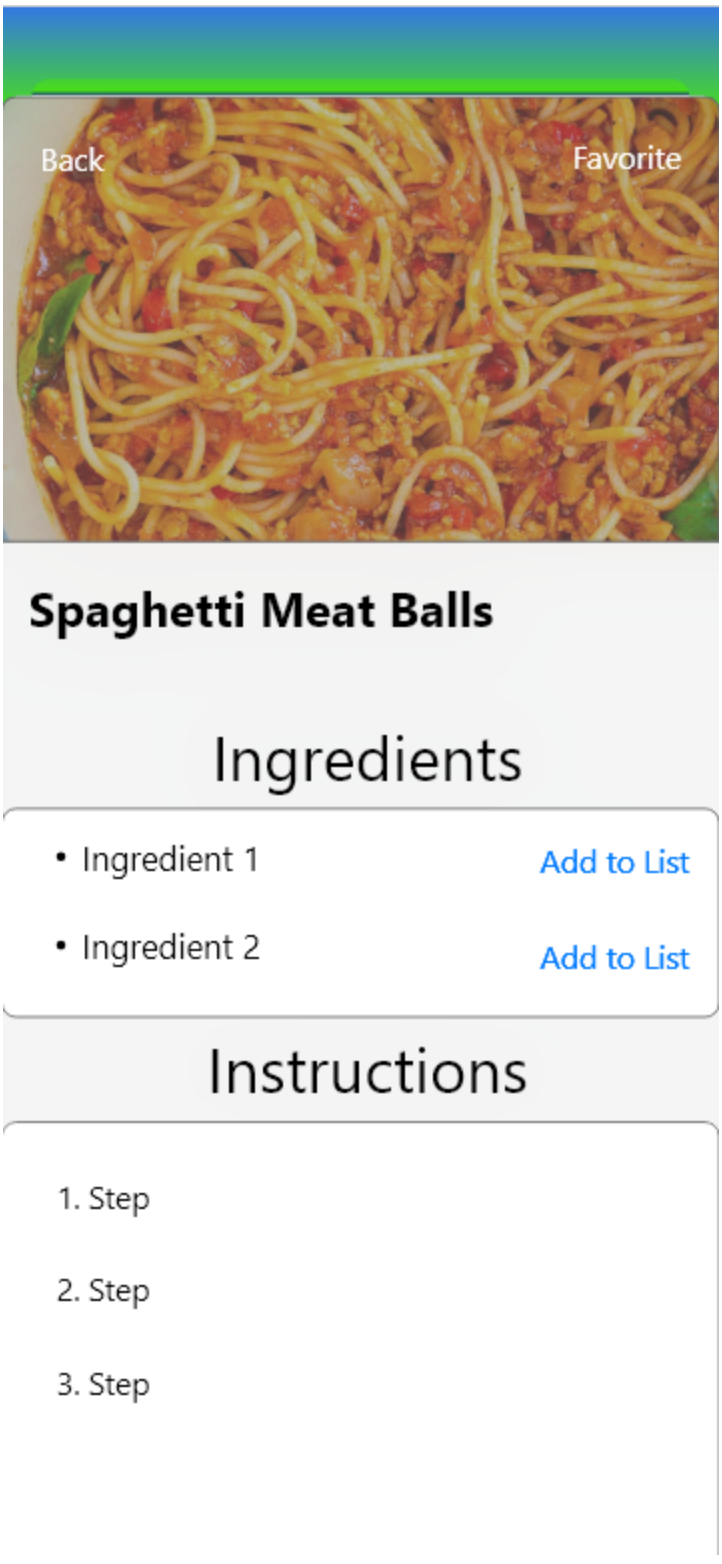
When designing a recipe app, we needed to lay out basic features that every recipe app needs. We first began with the search feature which is standard for our recipe search app. From this, we wanted two modes for searching: search by recipe and search by ingredients.



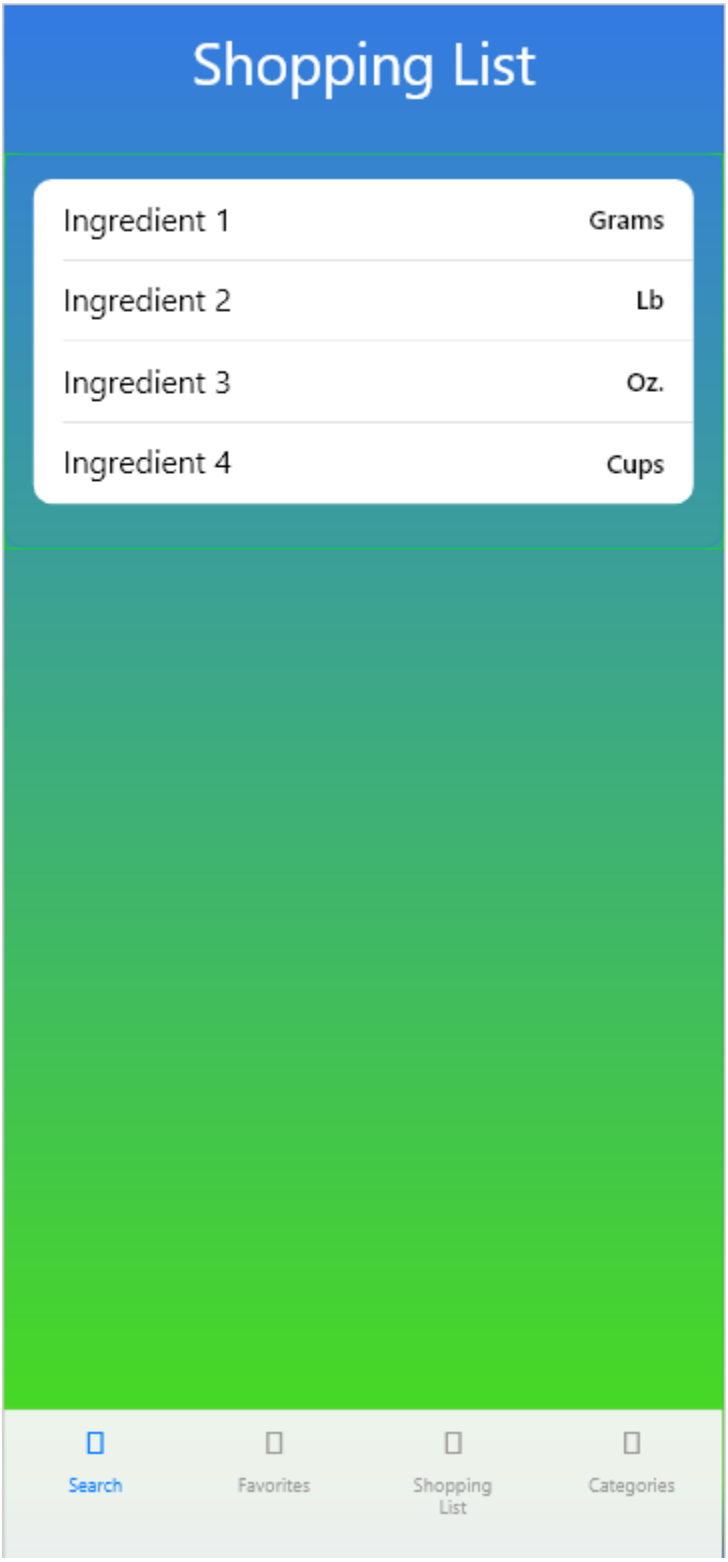
This is the prototype of our home/search screen for our app. The search bar is in the middle since it is the most important feature in this view. Next, the picker is placed to add the option of searching by recipe and searching by ingredient. When the user is finished typing the search query, the view needs to go to the search results. We decided to use a navigation view for this after the search is finished.



This shows the results screen which displays the search query at the top and a list of recipe cards in the view. The recipe card displays the recipe title and the type of recipe. When the user taps a recipe card it opens the detail of the specific recipe.



This view uses the .sheet() modifier on each recipe card which pulls up a detailed view of the recipe. In this view, the ingredients and a set of instructions are displayed for the user. Additionally, each ingredient under “ingredients” can be put into a shopping list which stores ingredients the user can view later for shopping.



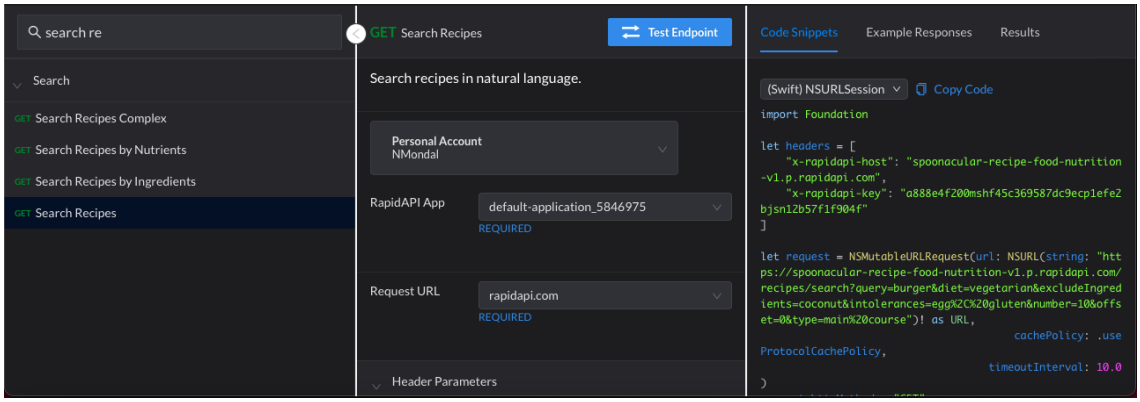
The prototype user interface contained the main features that we wanted to implement for our app and outlines the overall appearance of the app. As the app was developed, there were many elements that were changed to fit with SwiftUI.

First, the recipe results view and the recipe detail was changed to have style. The recipe results view is much bigger and easier to see details of the recipe before tapping it. The recipe detail view has more color and has a "favorites" button to let users add their favorite recipe.

API, Map, and Data (Nafi Mondal)

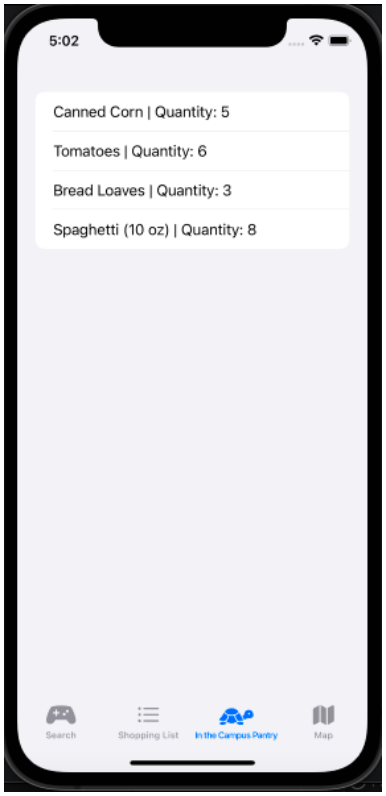
For this project we used the Spoonacular API to perform recipe searching and ingredient finding. In order to do this, we used a free account on RapidAPI to subscribe to the Spoonacular API before using it. Using NSURL functions given to us automatically with the API, we perform our search functions by inputting the queries from our search bar into the function parameters with some slight tweaks to make the queries run in regular SwiftUI at the top level.

The main functions we used to make this part were the search recipes, search recipes by ingredients, and get recipe functions. "Search recipe" takes a name of a recipe and returns several possible options of the recipe based on what is in the database (e.g. "spaghetti"). "Search recipes by ingredients" allows the user to input a string of different ingredients before performing a database search for recipes containing said ingredients (e.g. "apples, flour, sugar"). Finally, "Get recipe nutrition by id" allows the user to input a numerical ID of a recipe in the online database, immediately returning the matching recipe (e.g. 1003464). These inputs would be made within our search bar, and would return based on the type of search specified.

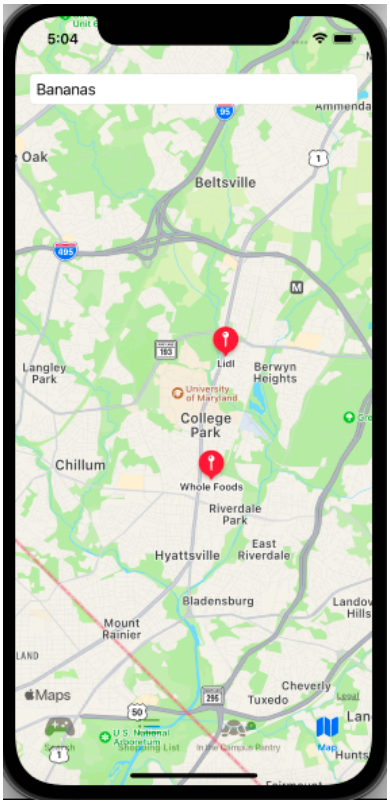


Since we made a recipe app, we wanted to store favorite recipes for the user to have in handy. We attempted to use Coredata to store favorites from Spoonacular onto the mobile device. However, there were issues that came up when we tried to implement it. Part of the reason was possibly due to the fact that we didn't start with Coredata at first, instead opting to retroactively fit it in once we got our functionality to work. However, since we had multiple files having to use Coredata, it was ultimately too complex for us to track how it worked, and therefore we weren't able to fully implement it on our Favorites tab. Despite that, it does still have some functionality due to what we were able to accomplish.

In an effort to make the project more UMD-oriented, we reached out to the Campus Pantry to see if we can get access to their inventory sheet to read in available ingredients for students on campus, however we never received a reply back. Therefore, we instead made our own CSV data sheet using dummy data with items the Campus Pantry would probably have (tomatoes, bread, etc.) consisting of the food name and the amount. The PantryView file reads the input from the CSV and separates the newlines and commas into data, which is then processed and stored in a list on a separate view. In the real world, we would expect this to be updated weekly or daily so students will have the most up to date information on what they can get.



For the MapView, we used a different sort of map setup from what we learned in class using a public data template from Paul Hudson (Hacking with Swift). This was because we needed to do a search query on ingredients that could be found in stores near College Park. Therefore, we used a UIViewRepresentable MapView centered around the College Park area to update where the annotations would be. We were successful in setting up the actual map and search bar, but the search calls using the natural language query option would not return any objects, whether it was food or the actual locations. This was puzzling because it did not act how we would have expected based on online tutorials and separate testing on Apple Maps. In the end we weren't able to fully solve the issue, but we were able to simulate some functionality using some dummy variables to showcase how it would have looked if the searching had fully worked.



Backend (Tahmid Hannan)

Initially, our main focus was on using an API to grab data, and have our app basically act as a visualizer of that data. The main stages were ‘Request -> Call to API -> Parse the JSON data to Swift Objects -> Present the data’. As Duy took care of the frontend UI related tasks, I worked on the backend that was related to handling requests, parsing the data for the API call, and then parsing the API call returns into Swift objects for visualization. For this project, we decided to create a similar structure as we did with the 2048 game projects - we had the frontend, and an observable class that acted as our backend. Our application is heavily dependent on the internet, as every single functionality makes API calls, and due to that, we had to handle every single request accordingly.

As mentioned above, we had to use Spoonacular API, and with that, we had to attempt to be as conservative as possible. Spoonacular API is a ‘freemium’, and only allows 50 requests on

the free subscription, and that led us to be as efficient as we can be with the API calls. As we slowly explored the Spoonacular API, I realized not all recipes returned were uniformly-structured as the others. For example, an API call to get bulk information (such as multiple recipe details) returned a recipe structure that was different from the results we got for the search API calls. Due to this, we have a lot of different Swift structs to deal with each different functionality, and as a consequence of this, a lot of views that we wanted to reuse for multiple purposes had to be heavily modified for compatibility. The biggest issue we faced was using API call results before the whole API call finished. This resulted in a lot of frustrations, and hindered the progress of our application by a lot more than we expected.

However, in the end, we were able to get the backend inline with our primary goal - a simple recipe application. Due to the difficulty getting the API objects to conform with our Swift objects, I, personally, couldn't pursue any additional functionalities to include in our app. The app seems a bit sluggish, due to the API response time being high - on average, RapidAPI says Spoonacular has an average latency of > 500ms.

Conclusion

Overall, despite some setbacks and some unfinished parts, we were ultimately successful in making a recipe finder app. While we are aware there are many apps like ours out there, we believe our app is unique due to it using data from the UMD Campus Pantry and having an in-app map to locate groceries in the College Park area, therefore making it a very useful app for UMD students and faculty to use.

Some of our stretch goals were not accomplished, the big one namely being using the camera to read recipes/ingredients and finding them using Spoonacular. If we had more time and more experience in using all the SwiftUI tools, we believe we would have been able to

accomplish it. In terms of things we would have done differently now that we have hindsight and experience using Xcode, we should have started our project with Coredata first so we wouldn't have had to deal with making sure lots of little parts would have worked perfectly together after adding it. We could have also started our Map tab earlier since we did not realize how much debugging and building there would have to be when dealing with map updates and querying. However, these mistakes and tough lessons taught us to be better coders and how to work together as a team to solve these kinds of issues, which is invaluable experience to have as we go into our careers.