- **Who was on your team?**
    - Ningyu Chen, Azad Ebrahimian, Cheng Xi Tsou, Sukhman Singh
- **What's the URL of your deployed app?**
    - https://cooking-app.wumbo.casa
- **What's the URL of your github repository with the code for your deployed app?**
    - https://github.com/azadebrahimian/WebDev_FinalProject2
- **Is your app deployed and working?**
    - Yes
- **For each team member, what work did that person do on the project?**
    - Ningyu Chen:
        - Integrating the Socket/Redux
        - Worked on adding owned ingredients tab/recipes tab/Grocery stores tab
        - Wrote the controllers/routes
    - Azad Ebrahimian:
        - Generated the idea for the project
        - Designed the database and relationships between tables
        - Successfully deployed web application
    - Cheng Xi Tsou:
        - Worked on adding new ingredients tab
        - Worked on Grocery stores tab
        - Wrote backend code for api calls
        - Wrote some of the controllers
    - Sukhman Singh:
        - Worked on the live feed on the home page and navigation bar
        - Helped design UI and layout of the site
        - Worked on "Virtual Fridge", searching and managing ingredients list

- **What does your project 2 app do?**

Our Project 2 app is a web application that allows for users to virtually store their real life refrigerator and pantry ingredients and research possible recipes that they can make with them. Every user will have their own virtual inventory, and have the chance to add new items when they acquire them or remove items when they run out. Every ingredient that can be interacted with contains details about itself like cost, location in store, and nutrition details. Whenever a user's ingredient inventory changes, their personal list of recipes will update to directly match what they currently own. If a user

needs an ingredient that doesn't exist in the main list of ingredients, they can search for it by directly going through our API's search feature.

Whenever a user updates their personalized set of ingredients, or finds a new ingredient from the API, a live feed will be updated and displayed to all users that are online at the time. This feed will display that a user has identified a new ingredient/recipe, and allow for other users to see this and do research on those items if they are interested in them.

Finally, our project also includes a section that allows for users to discover grocery stores near their area, in case they need to make a run to the store and pick up some ingredients that they are missing in a recipe they found. If a user allows for the site to access their location, our project displays grocery stores that exist within 20 kilometers of the user. Each grocery store contains details about its name, address, price level, and rating.

- **How has your app concept changed since the proposal?**

We changed many different small features of our app, but overall our app is the same. First, we slightly altered how recipes would be sent to the live feed. Instead of users having to click on recipes to show that they are "using" them, we instead made it so any recipes that a user finds will show up on the live feed. This gives the main user a way to look back and see all the recipes he could have made at the time he was searching instead of just the ones he clicked on. Additionally, it gives other users more recipes to look at, making it more likely they will find one that they are interested in. Next, we removed favorites because the new live feed accomplished what the favorites feature aimed to do. This also pushes the concept that our app is for discovering new recipes instead of saving old ones. After much deliberation, we decided to remove posts and its related functionality including comments and likes. The main reason for this was it deviated from the purpose of our app and it seemed like it was a forced addition. The main purpose of user to user interaction on our site is to give ideas on what recipes could be used or what new ingredients could be bought. Because this is accomplished by our live feed, the posts feature would be redundant and unnecessary. Another adjustment we made was removing friends. In parallel to the goal of the app described above, we wanted users to see the cooking activity of many different people and restricting interaction to only friends would be counterproductive. We also removed the cooking status of users because it would not only be a hassle to keep track of, it would also not add much benefit to other users. Even if users manage to maintain an accurate cooking status, other users would not benefit from seeing their friends status. With a similar thought process as this, we decided to change our "neat" feature because our

original idea would not be very useful. Initially, our idea was the ability to look at other users' fridges and request items. We think other users' ingredients should be private, so we would not want users to have the ability to look at others' fridges. Requesting would also not be helpful because virtually requesting items would have to be paired with meeting in person to trade the physical items anyways. The "transfer" of items can be done with a simple search for the receiver and a removal for the sender, so this functionality is covered. We replaced it with the useful functionality of finding nearby grocery stores so you can actually buy the food you are missing. This pertains to our app much more than the previous idea. It is expanded upon later in this report. Our final change to our app was an added mini-live feed to our navigation bar so all users could see the live updates on any page on the site. This should make users more comfortable while using the site because they will be able to see when other users are active.

- **How do users interact with your application? What can they accomplish by doing so?**

To interact with our application, users must first register an account. This requires the user to input a first and last name, a username, an email, and a password (two inputs for this to confirm the password). After logging in, users can see the live feed of what other users have been adding to the ingredient database and what recipes they have recently found. For some users, this could be all the functionality they need as it allows them to get some new ideas of what they can cook. For others users, the next would be to click on the "Virtual Fridge" tab and add any ingredients they have access to. These ingredients will be used to see what possible recipes the user could make or almost make. This virtual fridge is meant to be kept up to date, so as ingredients are used or bought, they should also be deleted and added to this database. Users can also scroll through this list to get an idea of what they want to make or what ingredients they want to buy. While searching for ingredients, there is a chance the user has a unique ingredient that is not currently in our database. To solve this issue, they can go to the next tab titled "Explore New Ingredients." Here, they can search for more ingredients in the Spoonacular database to add to our database. This would be printed on the live feed. After adding all these ingredients to their virtual fridges, users can use the "Recipes" tab to search for recipes they can make or almost make. The provided recipes state what ingredients would be used, what ingredients, if any, are missing, what ingredients would be unused, and a link to the recipe itself. This list of recipes would also be printed out on the live feed. The final interaction users could do on our app is on our "Grocery Stores" tab. On this tab, users can see information on all the nearby grocery stores and supermarkets, including their names, addresses, price levels, and

ratings. This gives users a way to acquire ingredients to add to their virtual fridge and complete more recipes.

- **For each project requirement above, how does your app meet that requirement?**

Our final project has much more functionality than our events app and bulls and cows app. More than just sessions and events, users have their own inventory of ingredients, and are able to use them to interact with a database to create personalized lists of recipes based on their input. We also access two different APIs in our project, and combine everything with a live feed for all online users.

Our web app is indeed built as two separate components: one for the back-end and one for the front-end. Our back-end is built using Elixir and Phoenix similarly to how was done for our previous assignments. A large portion of our app logic exists within the back-end, such as our database design/schemas, our item controllers, and as well as a JSON API to retrieve data based on data from our users. The front-end is a React app, and structured as a single page application. It can communicate with our back-end via JSON API to send/receive data, and sockets and channels to continue our live feed.

Our entire application (front and back end) is currently deployed on one team member's VPS. The two external APIs that we use for our application are not hosted locally however.

Our web application uses user accounts to keep track of each person's personalized inventory and recipes. Every user's password is securely hashed in our database to avoid any possible security threats. To improve the trust for user security, our website uses HTTPS to ensure full safety of user's information and passwords.

The whole application uses a postgres database to store all its user and app data. It also stores user data with sessions, so that a user can move around the web application without having to lose any data or be signed out of their account.

In our project's back-end we utilize two external APIs to help our application be fully functional. The first external API used is Spoonacular. Spoonacular is used to retrieve recipes based on a list of ingredients provided by a user, as well as discover new ingredients that might not currently exist in our provided list of ingredients. The second external API used is the Places API by Google. This API is used to list the existing grocery stores near the user's location.

Our project uses Phoenix channels to push data to a live feed that is visible for all concurrent users of our application. This live feed is updated based on user action, whenever someone discovers new recipes or ingredients that other users might be interested in. The moment any new recipe or ingredient is discovered, the corresponding feed will be updated instantly for everyone to see.

Our application's "neat" feature is using location services to display the nearest grocery stores from a user. We decided to use a second API that isn't required from the project details to accomplish this task. In order for the Places API to acquire the location of a user, we had to establish an HTTPS connection for our website as well. We thought this extra feature would stand out from the rest of the assignment, since the project doesn't focus at all on a user's location.

As a team we thoroughly tested our web application for bugs, UI mistakes, and anything else that we felt was wrong in the project. Of course we debugged heavily while creating the project, but once it was completed we spent a day looking for edge cases and finding slight errors that shouldn't have existed. With this group testing, we discovered numerous issues and were able to eliminate them one-by-one. One issue we found with our socket code was how our live feed would not update until the user refreshed their page. After looking into it, we realized that this was due to socket and tokens being created before a user joined the channel. A UI issue we discovered was how an error message would not leave even after the issue was resolved, causing an annoying message being displayed and confusing the user. We simply fixed this by ensuring that the error stored in our redux store was cleared after the issue was resolved.

- **What interesting stuff does your app include beyond the project requirements?**

An interesting part of our project that was beyond the requirements was the use of the HTML Geolocation API and Places API. We had a grocery stores tab where users find supermarkets near them up to a 20 km radius. When the user clicks on the grocery store tab, the HTML Geolocation API grabs the coordinates of the user, feeds it to the Places API, then queries a list of places with the tag "supermarket" in the Places API. If the user does not turn on their location in the browser, nothing will be displayed. If it is turned on, we display the list of supermarkets in the frontend in a table. The HTML Geolocation is also deprecated on an insecure website, so we had to establish a HTTPS connection for the feature to work. We thought by using the HTML Geolocation and Places API, this feature is beyond the project requirements.

- **What's the complex part of your app? How did you design that component and why?**

The complex part of our app is how we manage our ingredients list on the server. When designing how to implement this aspect of the site, we decided not to allow users to add arbitrary ingredients to the database because we wanted to ensure that all ingredients were valid and were compatible with the Spoonacular API. Therefore, we decided to preload the top 1000 ingredients on the API to our server database. We implemented a search bar that filters the possible ingredients as you type. For the cases that the ingredient users want is not available in our database, they search through the Spoonacular database for new ingredients and add them to our database. This action is also posted on the live feed so other users could see how the database has been updated. This design is the most beneficial for both the users and developers.

- **What was the most significant challenge you encountered and how did you solve it?**

The most significant challenge was designing our live feed functionality. Our first plan was to have every user join its independent topic and register its topic by username, and the handle_in functions in the channel module will be responsible to broadcast live feeds and call the database base, which I think it's the responsibility of the controller. Then we decided to have every user join a single channel regardless of topics. We implemented a live feed process(genserver) that will be instantiated along with the application to track the live feed data and handle broadcasting to the users. When the user makes an api request to the controller, and the controller will decide whether to broadcast the event, if it does it will push updates to the process and instruct a broadcast, and send back a json response.