# Sequence Diagram Design Document

**Introduction:**

This project is to create a Recipe Manager application that allows users to create and view recipes. They should be able to import and export recipes as files as well, and there should be a shopping cart they can add ingredients to. This document aims to outline the interactions between a user and the application, as well the expected behavior of the application during this interaction. This design should cover a high level overview of the key user flows in the application and the interactions between our frontend and backend. Details regarding implementation, interactions between components within the application, and potential issues are not within the scope of this document and will not be covered.

**System Overview:**

We intend to create two subsystems to accomplish this project. One will be a user interface and frontend written in Javascript using the React framework. The other will be a Node.js server and backend used for storing and retrieving recipes for the user. Most view-related functionality will be implemented on the frontend. The key features for the frontend are as follows:

- The ability to create a recipe
- The ability to edit a recipe
- The ability to delete a recipe
- The ability to search for recipes
- The ability to organize recipes into categories
- The ability to import a recipe from a properly formatted JSON file
- The ability to export a recipe as a properly formatted JSON file
- The ability to add ingredients to a shopping cart
- The ability to remove ingredients from a shopping cart

Any adjustments made to the content of a recipe or organization of recipes should be sent to the backend using a combination of POST, PUT, and DELETE requests. All recipes should be displayed by utilizing the GET method. Therefore, the following features should be implemented for our backend service:

- The ability to create a file for a new recipe
- The ability to edit a file for an existing recipe
- The ability to delete a file corresponding to a recipe
- The ability to create a folder for a category of recipes
- The ability to create a folder for a User
- The ability to move files into folders for organization
- The ability to return an organized hierarchy of recipes to the frontend
- The ability to add ingredients to a shopping cart file corresponding to a User
- The ability to remove ingredients from a shopping cart file corresponding to a User
- The ability to return a list of shopping cart ingredients to the frontend

For the completion of this project, we are aiming to finish a prototype of this application in which you would run both the server and UI locally, and calls would be made from the UI to the local server. However, this can easily be ported from a locally run prototype to a full application in a

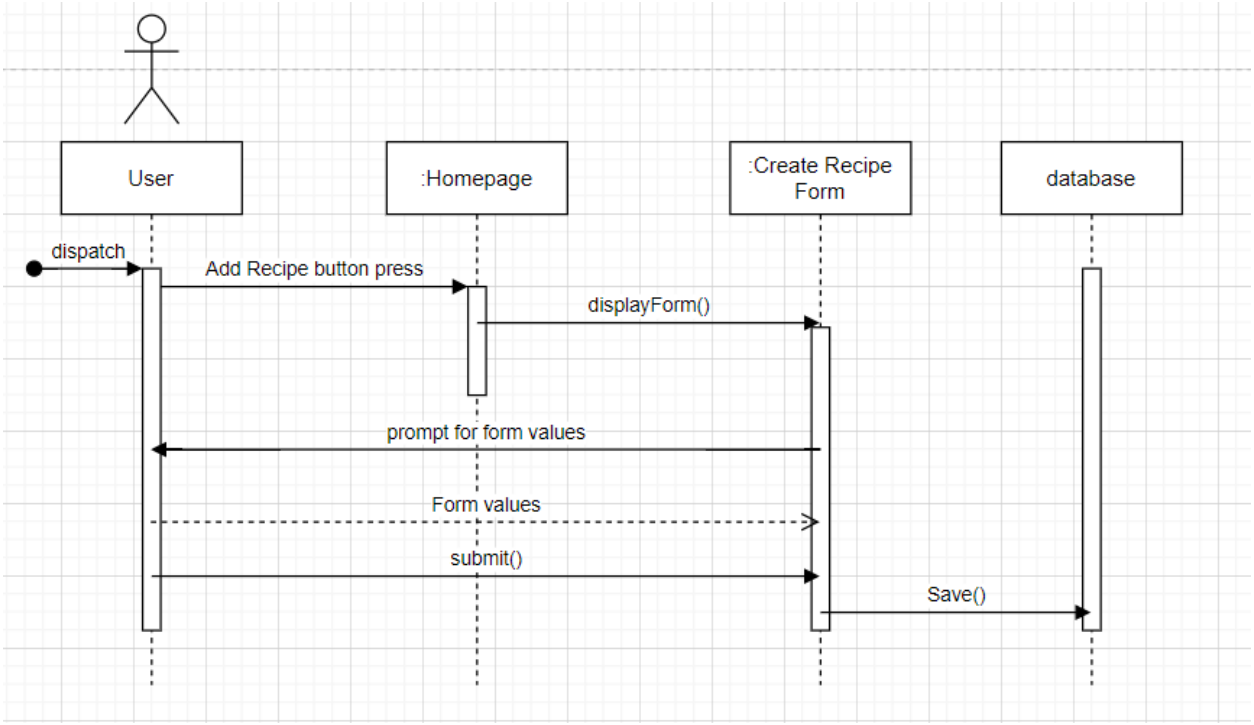
few simple steps. The UI and server are hosted fulltime on a dedicated computer or server, the request URLs within the UI are changed to reflect the new location of the server, and any CORS policies are properly updated to avoid bad requests.

**Use Cases and Sequence Diagrams:**

Recently we planned out Use Cases to define our User Acceptance Criteria. Below is a refined list of these Use Cases for the purpose of creating thorough but not repetitive Sequence Diagrams.

1.1. Use Case 1

- Title: Add a recipe
- User Goal: To create and save a new recipe to the app
- Preconditions: User has access to the “add recipe” functionality
- Steps:
    1. User presses the associated “add recipe” button in the home page.
    2. The user enters the name of the recipe and at least one step into the text entry fields.
    3. The user uploads an image of the completed recipe.
    4. The user may enter other optional information such as required cookware or additional notes.
    5. The user clicks the “save” button to add the recipe into the app.
- Priority: High

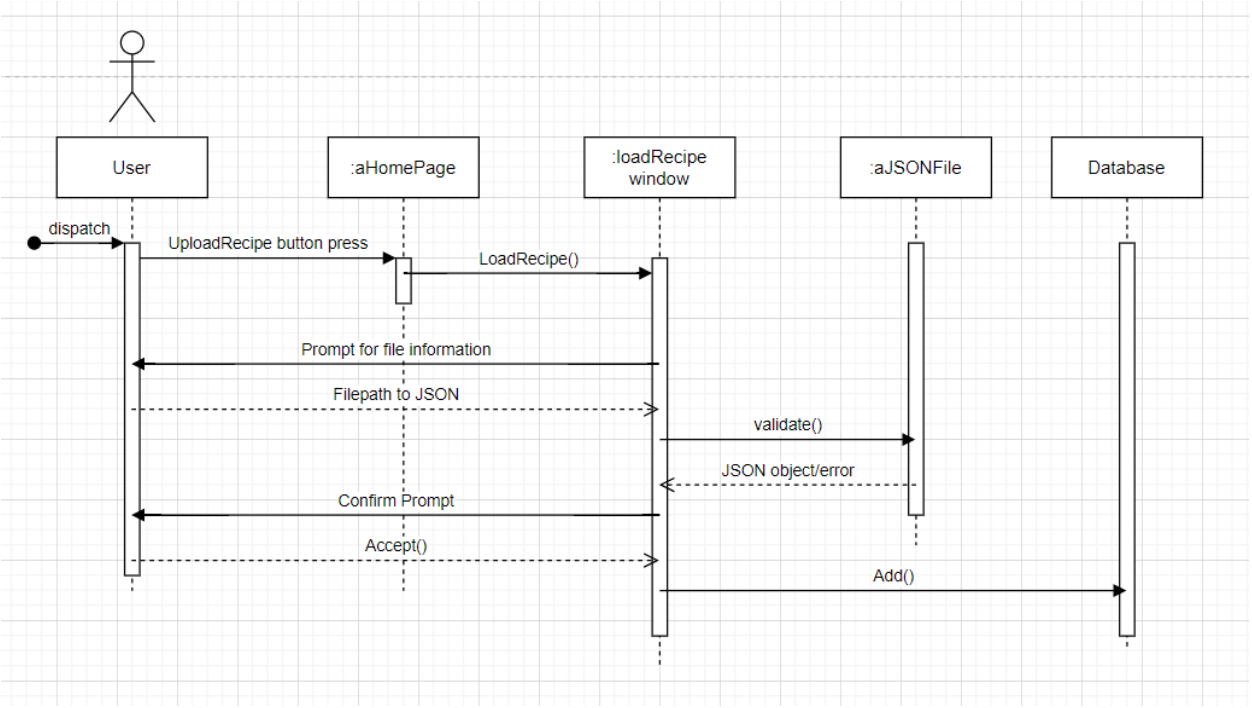



1.2. Use Case 2

- Title: Upload a recipe

- User Goal: To upload a pre-existing file containing a recipe to the app and have it be displayed.
- Preconditions: The user has access to the “upload” functionality, and has a .json file containing the recipe information.
- Steps:
    1. The user presses the associated “upload recipe” button in the home page.
    2. A popup window appears prompting the user with the proper format for information in their .json recipe file and a field to upload a file.
    3. The user uploads a .json recipe file.
    4. The app validates whether the .json file is properly formatted, and returns an error message if it is not.
    5. User presses the “submit” button to add the recipe to the app.

- Priority: Medium

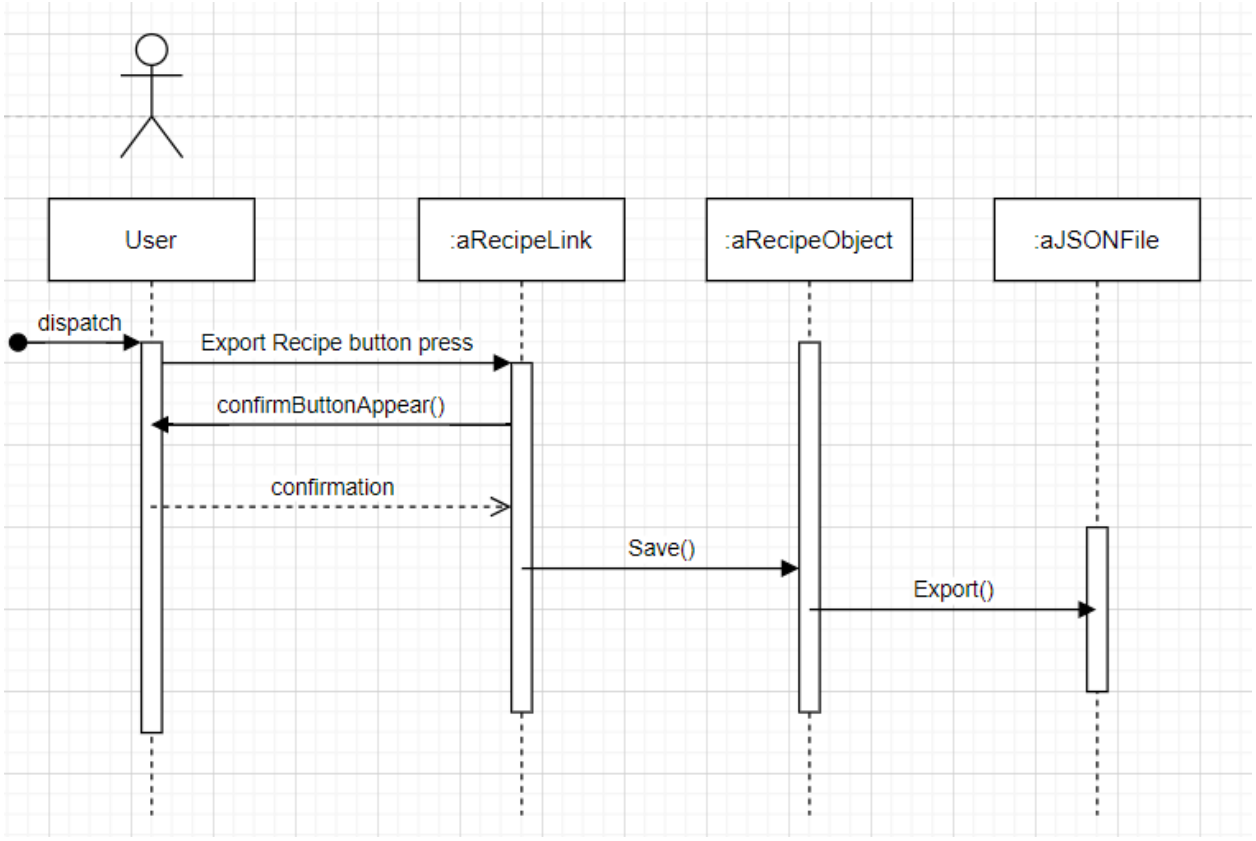



### 1.3. Use Case 3

- Title: Export a recipe
- User Goal: To export a locally stored recipe to a file that they can send to other users to get that recipe.
- Preconditions: The user has access to the “export” functionality and has at least one recipe stored on their application.
- Steps:
    1. The user presses the associated “export recipe” button on an existing recipe.
    2. The file is downloaded to the User’s downloads folder.

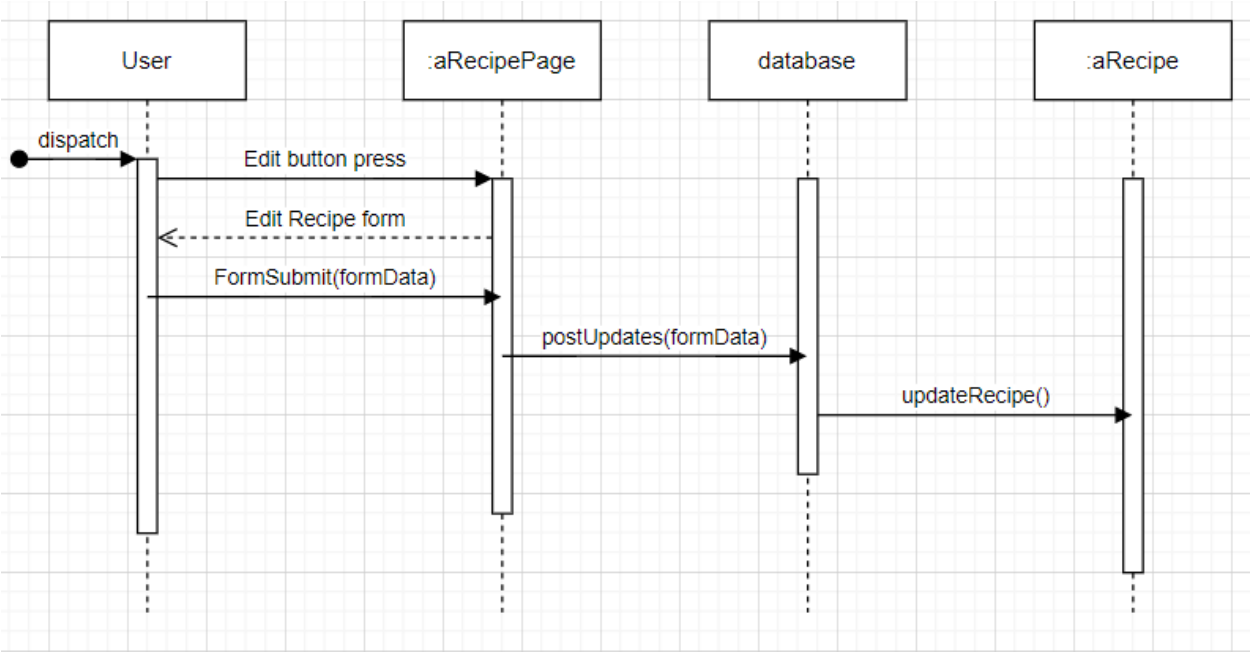
3. The user can send this file to a friend or to another instance of their application.

- Priority: Medium



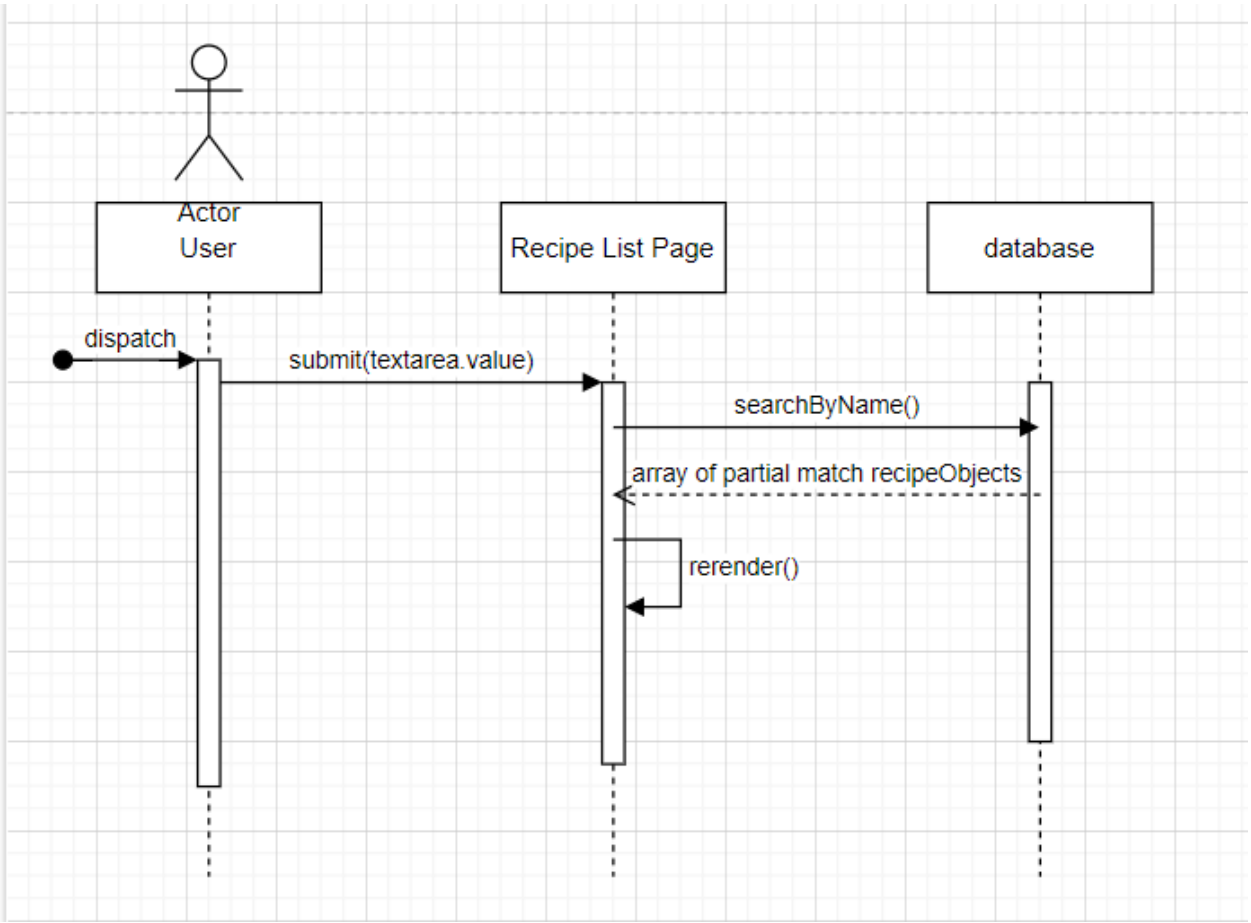

1.4. Use Case 4

- Title: Edit a recipe
- User Goal: To take a recipe in their application and edit the fields to add or remove information.
- Preconditions: The user has at least one recipe stored and it is open.
- Steps:
  1. The user clicks on the edit button on an existing recipe.
  2. The application opens a form that is filled in with all existing information on the recipe.
  3. The user enters new information and hits the save button.
  4. The recipe will switch out of edit mode and the new information is shown.
- Priority: High




1.5. Use Case 5

- Title: Search for recipe by name
- User Goal: To use a search bar to find a specific recipe
- Preconditions: The user has the desired recipe stored
- Steps:
    1. The user clicks on the search button on the home page
    2. The user enters the name of a specific recipe
    3. The user presses the enter key on keyboard
    4. The application brings the user to a page with all the recipes associated with the name entered
    5. The user selects the desired recipe
    6. The application opens the recipe selected
- Priority: High



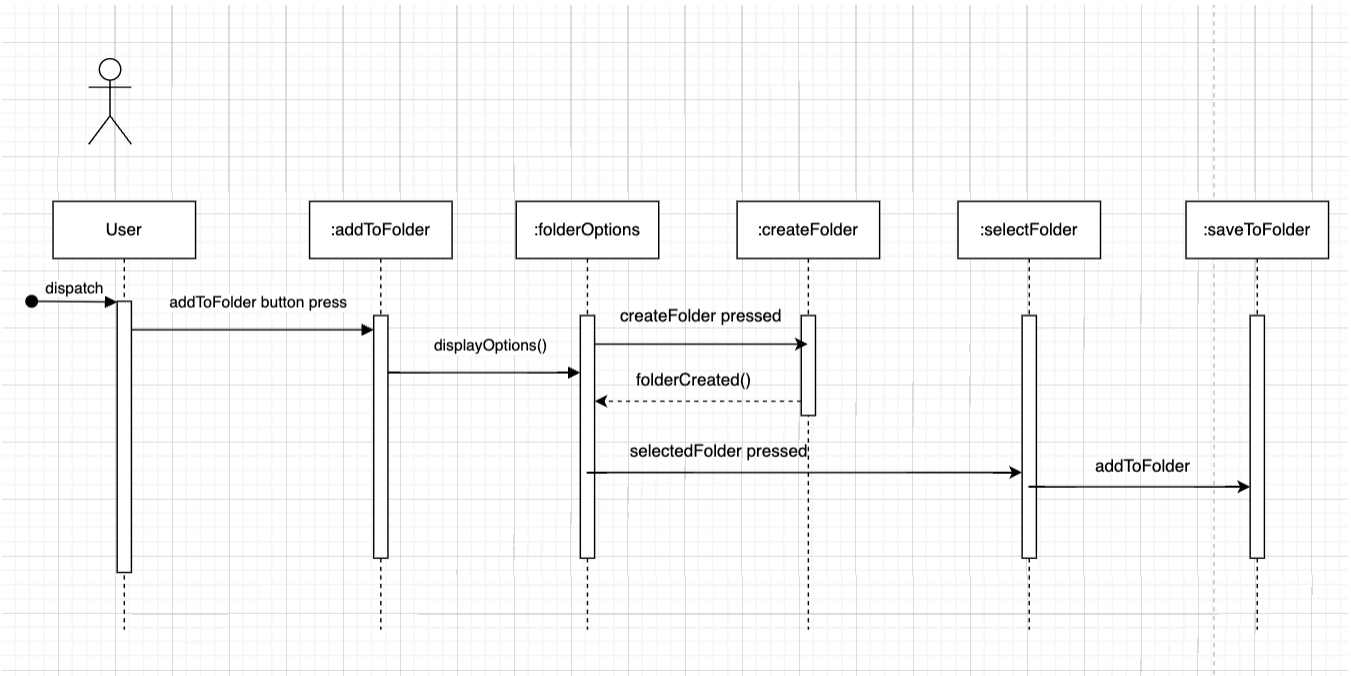

1.6. Use Case 6

- Title: Categorize a recipe
- User Goal: The user can sort saved recipes into groups
- Preconditions: The user has all required areas of a recipe filled out but has not saved
- Steps:
    1. The user clicks on an “add recipe to folder” button
    2. The application brings up a pop up with two options
        a. If the user wants to save in an already existing folder
            i. User selects desired folder
            ii. User clicks on an “add” button
        b. If the user wants to create a new folder
            i. User clicks on an “add folder” button
            ii. Application brings a pop up that prompts user to input folder name
            iii. User clicks on field
            iv. User inputs folder name
            v. User hits save button
            vi. Folder is added to list of folders
            vii. User selects desired folder

viii. User clicks on an “add” button

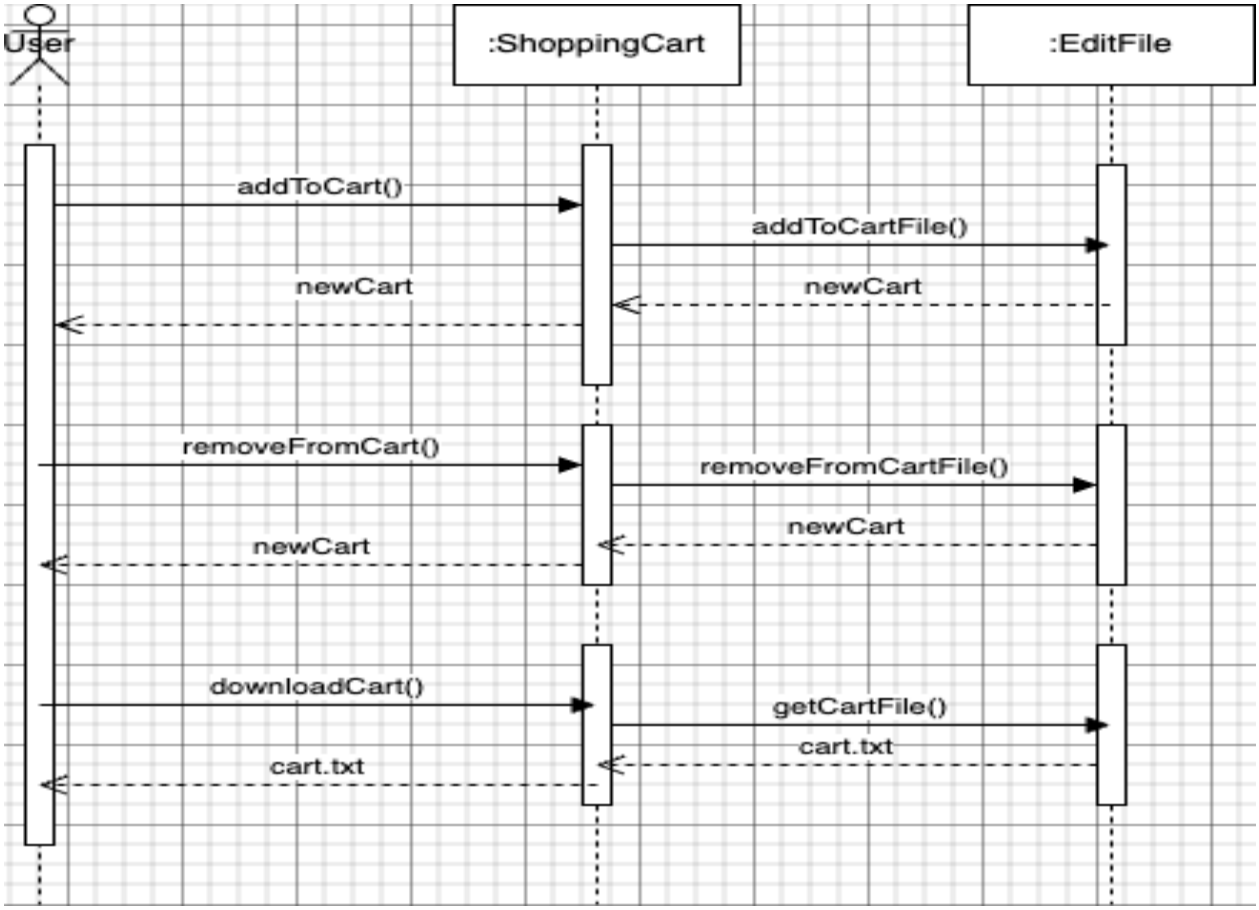
3. User clicks “save” button

- Priority: Medium




### 1.7. Use Case 7

- Title: Manipulate shopping cart
- User Goal: The user can add and remove ingredients to a shopping cart, and download the cart as a .txt file
- Preconditions: The user has at least one recipe in their application that contains at least three ingredients and the user has an empty cart
- Steps:
    1. Select a recipe with at least three ingredients from the home screen
    2. Click on the add to cart button next to three ingredients from the recipe
    3. Return to the home page
    4. Click on the shopping cart button
    5. The three ingredients should be displayed in the cart
    6. Click the delete button next to one of the ingredients
    7. That ingredient should no longer be displayed on the cart
    8. Click the export button in the shopping cart
    9. A .txt file should be downloaded to the user’s computer
    10. That file should contain only the two remaining ingredients




**Design Principles and Guidelines:**

When creating sequence diagrams for our Recipe Manager application, we will be using the following design principles and notation conventions:

- Modularity: Our sequence diagram will exhibit modularity by breaking down the interactions between the frontend and backend into smaller, more manageable steps. Each step will represent a specific action or behavior of the system.
- Abstraction: Our sequence diagram will focus on representing interactions between different components or actors without going into implementation details. They will provide a high-level overview of the system's behavior.
- Notation Conventions: UML notation will be used for creating our sequence diagram. This includes representing actors, lifelines, messages, and method calls using standard UML symbols.
- Lifelines: Lifelines represent the participating objects or components in the sequence diagram. Each lifeline represents an instance of a class or a component involved in the interaction.
- Messages: Messages will represent the communication or method calls between lifelines. They can be represented by arrows with labels indicating the method name and any relevant parameters.

- Activation Bars: Activation bars represent the time period during which an object or component is actively executing a method call. They will be shown as a box extending from the lifeline.
- Return Messages: Return messages will be used to indicate the flow of control and the return value from a method call. They are represented by dashed arrows or lines.
- Conditions and Loops: If there are any conditions or loops involved in the sequence of interactions, they can be represented using combined fragments, such as alt (alternative), opt (optional), and loop (looped) fragments.
- Annotations: Annotations will be added to the sequence diagram to provide additional explanations or clarify specific steps or conditions.

By following these design principles and notation conventions, the sequence diagrams for our Recipe Manager application can properly represent the interactions between the user interface and the backend pieces, as well as show the expected behavior of the system during various use cases.

**Design Validation**

One of the main validation techniques we used was peer review, as we all went through each of our sequence diagrams and use case scenarios to make sure that they appropriately fit what we were trying to achieve. In order to do this, during our weekly meeting, we went through each use case and sequence diagram individually, making sure that they fit our requirements. We found a few tweaks that were necessary in multiple sequence diagrams, and changed them to ensure that they were robust and representative of our use cases.

**Design Evolution**

For the future, as more use cases may be added we will add more sequence diagrams to represent them. In addition, as our processes and methods used to program the different features evolve, the sequence diagrams will have to evolve with them. While our current sequence diagrams represent what pieces we currently use, there is great potential for them to be updated in future versions of the software as the program needs themselves evolve. In making our sequence diagrams we made sure to implement them in ways so that they have a lot of ability to evolve and change in the future, making sure that relationships between the different pieces are able to be changed if needed. With this, by using sequence diagrams, we were able to modularize our program in a way that we’re able to adapt to any future changes that may be necessary in a relatively short time frame.