**RECIPE CHATBOT IMPLEMENTATION: Culinary Assistant**

1. **INTRODUCTION**

The goal of our project is to implement a chatbot that helps the user to prepare certain recipes. The idea is that the user asks the chatbot what he/she can cook with certain ingredients, or to get inspiration, etc. In addition, if the user has been lost during the procedure the chatbot repeats the step, the chatbot also asks about how it is going during the conversation, provides the original source (link) or search for similar recipes if the user specifies it.

1. **INSTRUMENTS NEEDED**
   1. **RECIPE DATASET**

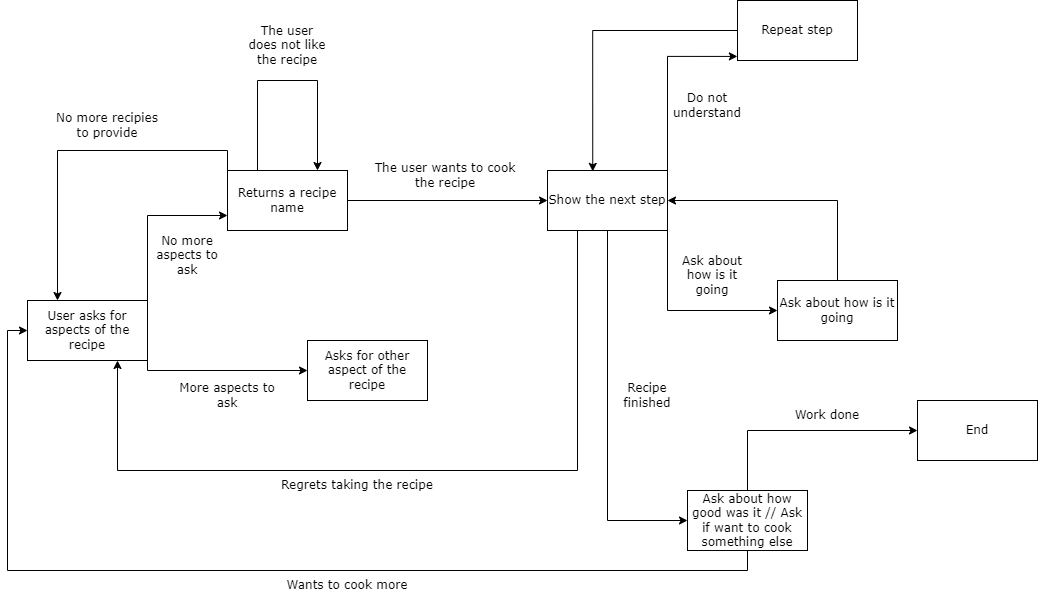
The dataset used in this project is called ‘RecipeNLG (cooking recipes dataset)’ from kaggle.com. It consists of a Cooking Recipes Dataset for Semi-Structured Text Generation with the title, the ingredients, the directions, the link, the source and the keywords of 2,231,142 different cooking recipes:

<https://www.kaggle.com/datasets/paultimothymooney/recipenlg>

* 1. **DIAGRAM**

The first part of the elaboration of the project consisted of preparing a visual scheme to have clear the actions that we wanted the chatbot to carry out.

The following image shows the diagram of the desired flow of the conversation according to the user's intentions.



**Fig 1:** Conversation flowchart

* 1. **SENTENCE SIMILARITY DATASET**

Furthermore, in order to have clear what we want the model to perform, we created tables with possible statements examples that user could ask, in what type can be classified, and what is going to response or do the chatbot:

Enumerated types of user input:

| don't understand other step | 1 |
| --- | --- |
| don't understand this step | 2 |
| don't want to continue (change to other recipe) | 3 |
| don't want to continue (stop) | 4 |
| keep going | 5 |
| ask for link | 6 |
| repeat this step | 7 |
| end the conversation | 8 |
| question for recipe (given ingredients) | 9 |
| question for suggestion | 10 |
| without certain ingredients | 11 |
| new recipe | 12 |
| stop | 13 |
| keep recommending | 14 |
| reestart / want to give ingredients | 15 |

We use sentence similarity to “classify” the inputs of the user to be able to carry out the actions desired and know what to answer.

We have attached the dataset to the .zip

* 1. **LIBRARIES, PROGRAMS AND MODELS**

To implement our chatbot, we have used several natural language processing tools:

**WordNetLemmatizer()**

Used to group together the different inflected forms of a word so they can be analyzed as a single item. We applied it when classifying the user’s input.

**SentenceTransformer()**

Used to compute sentence embeddings, which can then be compared with cosine-similarity to find sentences with a similar meaning. Basically, we have used it to compute sentence similarity. We have used this for all the user inputs that are not a yes/no response.

**DialogTag()**

We used this python library essentially to do dialogue act classification. Dialogue act classification is the task of classifying an utterance with respect to the function it serves in a dialogue, i.e. the act the speaker is performing. We have used this for all the user inputs that are a yes/no response.

1. **IMPORTANT FUNCTIONS OF OUR CODE**

To implement the chatbot in such a way that the conversation would have a flow, we have designed 11 different modes that we explain in the following section. In each of them, we use auxiliary functions implemented by us- In this report we explain the most relevant ones:

**ingredient\_check()**

This function consists of returning the lists of ingredients (keywords) where the input is found. The input passed as a parameter is cleaned and splitted, and for each of the resulting words it is checked if it is part of the list of ingredients. If true, it adds it to the returned list.

**recipes()**

This function is used to get all the information of the recipes where the ingredients list input appears. To get the ids with the function recipes\_ids(), we randomly shuffle this list and select all the necessary information from the first 5 recipes.

**response\_yes\_no()**

This function is quite simple, it just uses the dialog model to get the class type of the input passed as a parameter to detect if it's a yes a no, or something else input (using a dialogue classifier).

**sen\_type()**

This last function is used to get the type of the question (using sentence similarity). It consists of iterating over all the elements of the dataset created by us, comparing the similarity sentence of the input with each one of the sentences, saving the result in a new dataframe with columns ["Sentence", "Type", "Score"] and when finished, sort this dataframe in ascending order, and selecting the type with the highest score. It is important to mention that to check that the input makes sense, we only accept scores higher than 0.3. If the first score is less than this number, a -1 is returned, indicating that something is not working as it should, the user input is not as expected.

1. **FLOW OF THE CONVERSATION AND FEATURES**

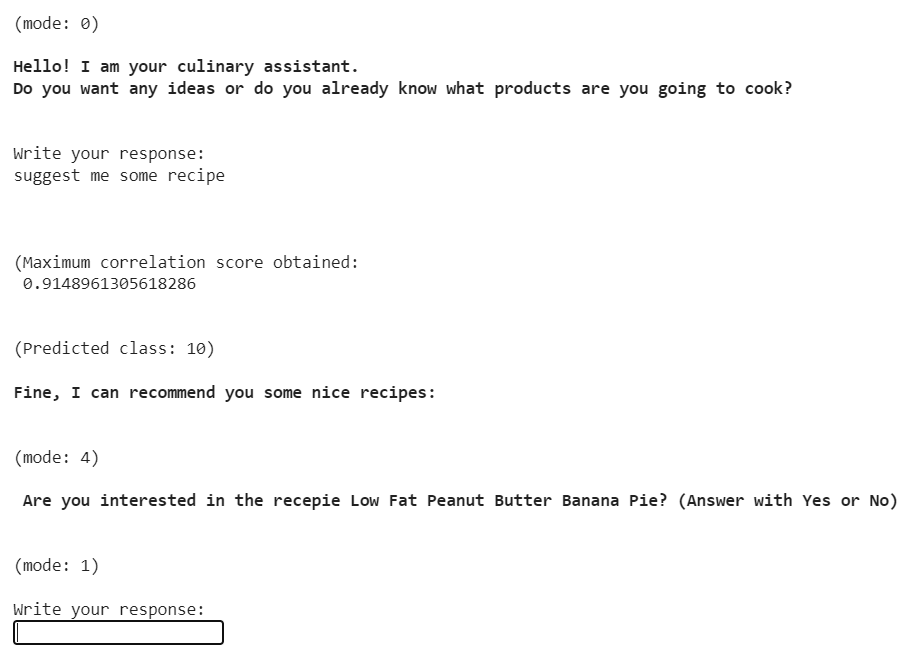
To keep a flow of the conversation we divided the states of the conversation into modes, where each mode represents different ways of behaving. In all the modes we have implemented two general features.

We keep track of the consistency of the user input (as can be seen in figure 4). Furthermore, if the number of consecutive incoherence inputs from the user is higher than 3, the bot stops the conversation.

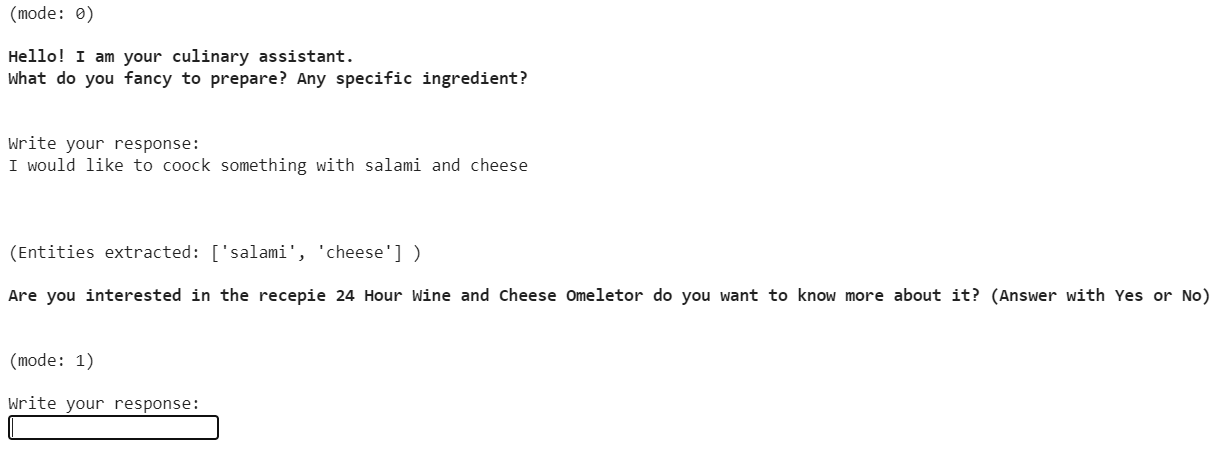
* 1. **Mode 0**

In this mode, the conversation begins and the user needs to decide what he/she wants. To get a suggestion or a recipe with different ingredients.

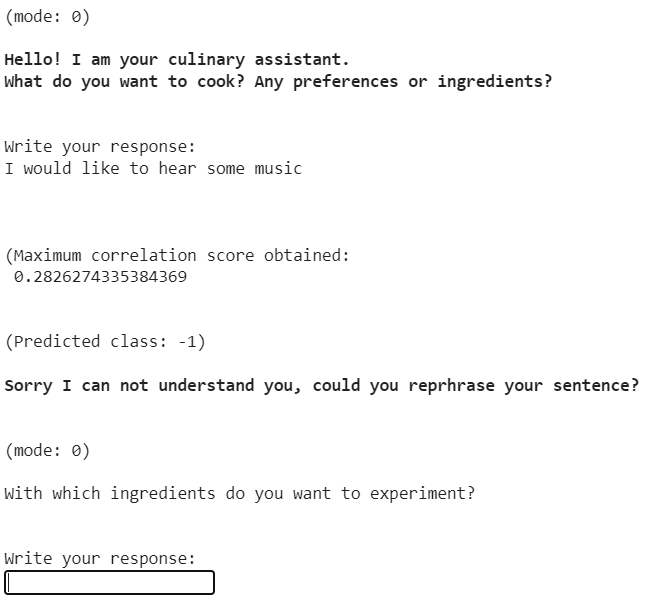
Furthermore, at the beginning of the conversation, the user can only be asking for two types of information: asking for a recipe with given ingredients or needing a suggestion. Therefore, it checks if there is any ingredient within the input, if so, the chatbot proceeds to look for recipes, if not, it proceeds to suggest recipes.



**Fig 2:** Example of user asking for a suggestion.



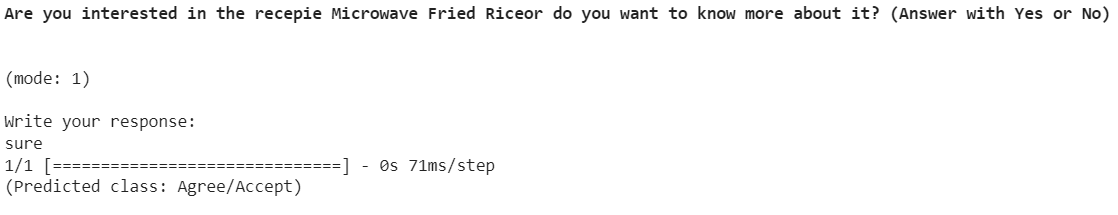
**Fig 3**: Example of user asking for a recipe with certain ingredients



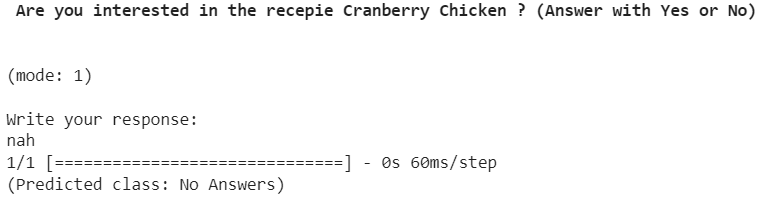
**Fig 4:** Example of user asking something with no sense, score lower than 0.3.

* 1. **Mode 1**

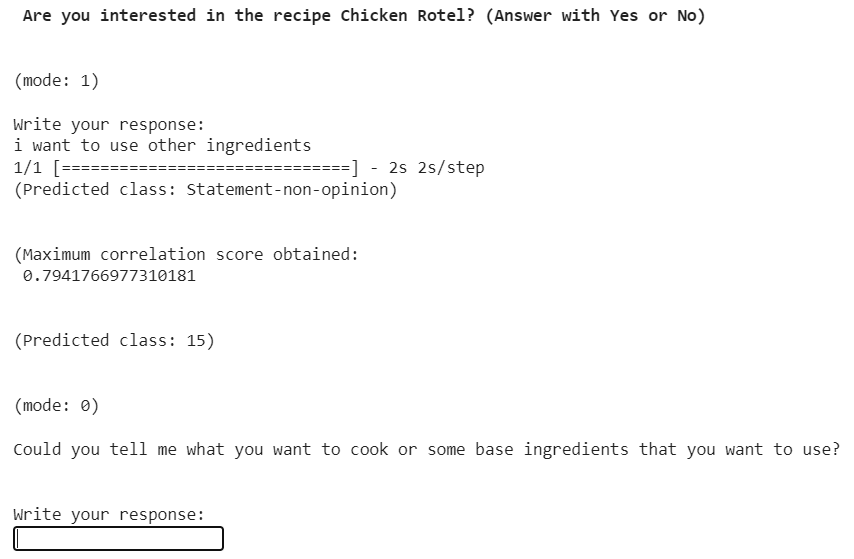
We use this mode as an auxiliary method to check in some cases if the user is trying to say yes or no. To achieve this we use a dialogue model that tells us the type of the user entry. It doesn’t work well enough for the rest of the user entities.



**Fig 5:** Example of user saying sure, the predicted class is from the set of ‘yes’ responses.



**Fig 6:** Example of user saying nah, the predicted class is from the set of no responses

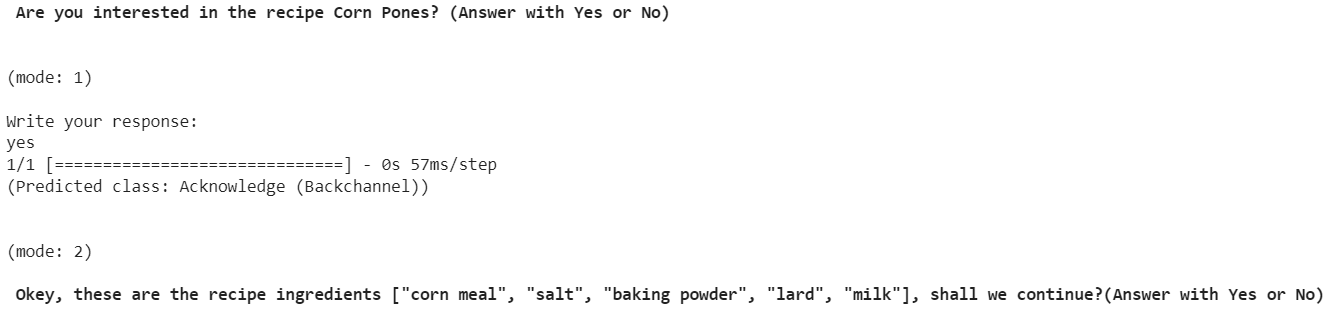


**Fig 7:** Example of when the user is bored from seeing the recipes and wants to select by himself some ingredients. User said “ i want to use other ingredients”.

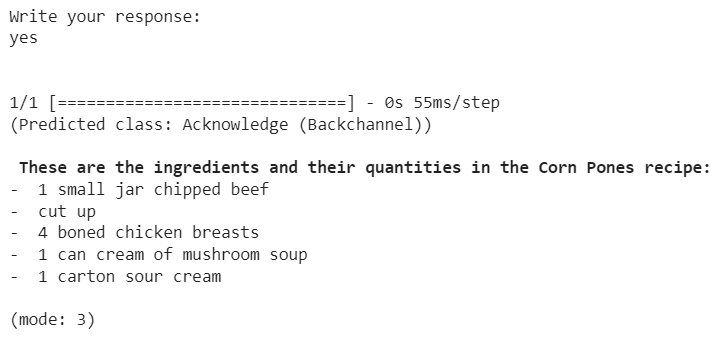
* 1. **Mode 2**

We use this mode to give the ingredients/quantities of the chosen recipe and to know if the user is okay with it. If the user is not satisfied with the ingredients of the recipe chosen, is asked to change the recipe or to provide new ingredients.

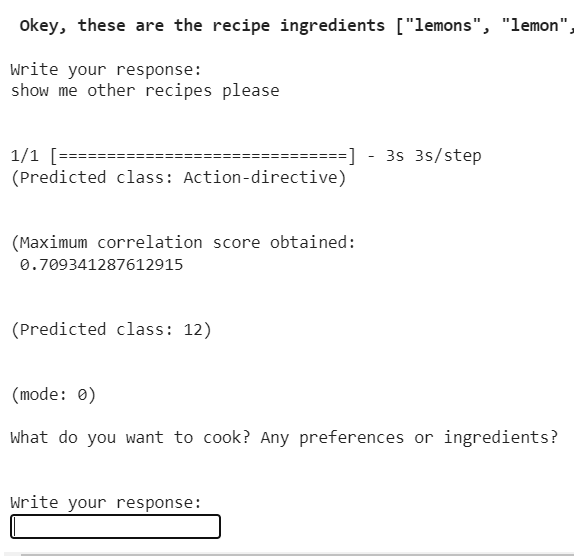
Moreover, in this module the exact quantity of ingredients are given before starting to provide the steps. We have to take in mind that the user can decide at any point of the conversation to change the recipe or to stop (example in fig 10).



**Fig 8:** Example when the user likes the recipe, we show the full list of ingredients



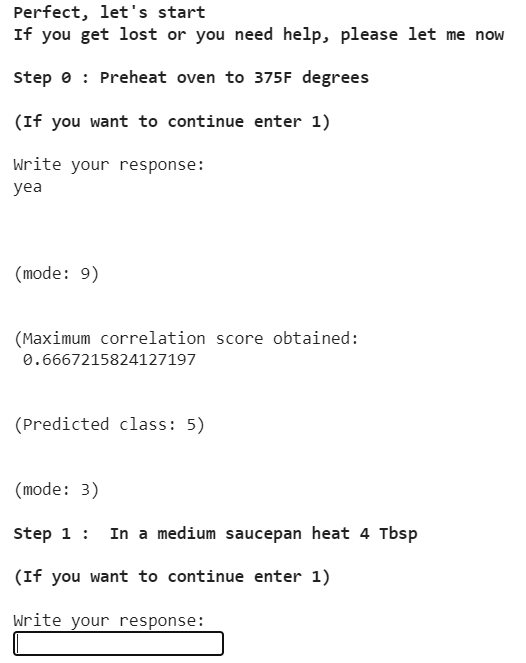
**Fig 9:** Example, when the user likes the ingredients, we show them the quantities and we go to mode 3, where we start with the first step.



**Fig 10:** The user don’t like the recipe ingredients and decides to start again

* 1. **Mode 3**

Here is where the enumerated steps of the recipe are given. There is also a follow-up to identify if the user does not get lost, wants to repeat a step or end the recipe.



**Fig 11:** Similar example of mode 1, the user can say yes in several ways here too.

Depending on the input user, the modes will change. But mode 3 only takes charge of yes or no questions, when the input isn’t identified with any of these classes we send the result to mode 9.

* 1. **Mode 4**

This mode is for suggesting random recipes. The recipes are chosen with the getRandomRecipes() function and it is asked to the user if the recipe proposed is fine for him/her.

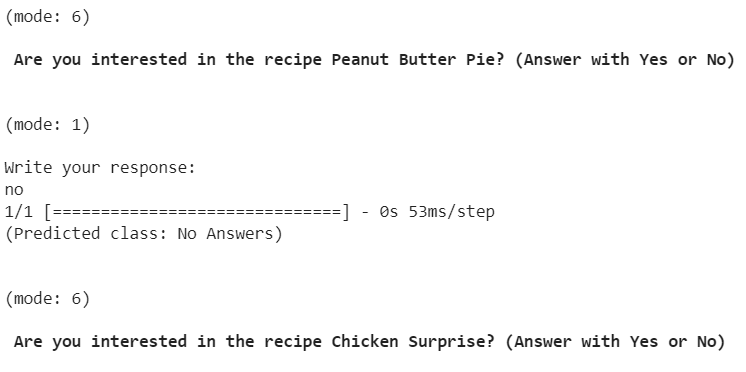
We can see an example in Fig 1. When the user is in this mode we also use mode 1 to know if the user likes or dislikes the presented recipe.

* 1. **Mode 5**

This mode is used to provide the link of the original source of the recipe. We use it when the user gets lost or wants to end the conversation but after it, saving the recipe link. In figure 10 can be seen an example of this functionality.

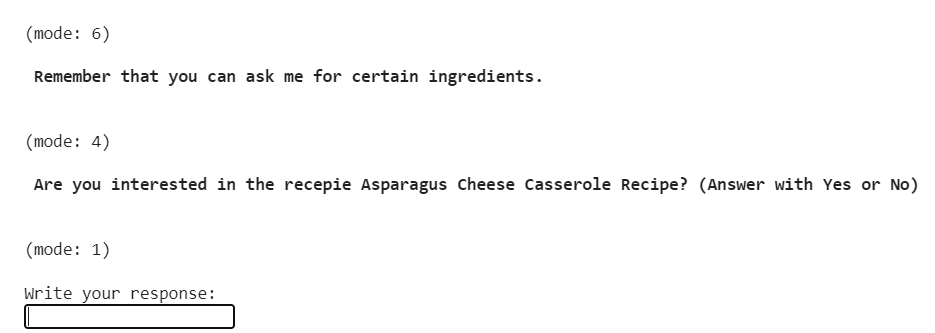
* 1. **Mode 6**

If the user wants to change the recipe, it is sent to this mode, where the list of possible recipes to offer is iterated and if this list is finished, we inform the user.



**Fig 12:** The bot shows recipes but the user rejects them

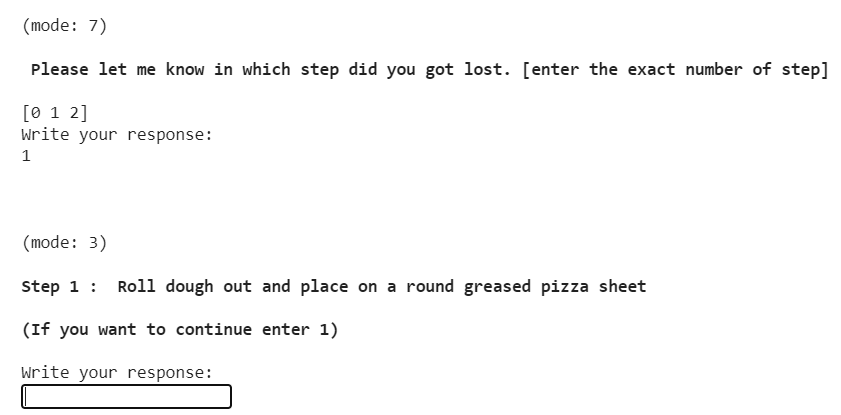
There is also a feature that shows a message to the user when he has skipped the recipe 5 times, reminding that he can choose the ingredients for the recipe.



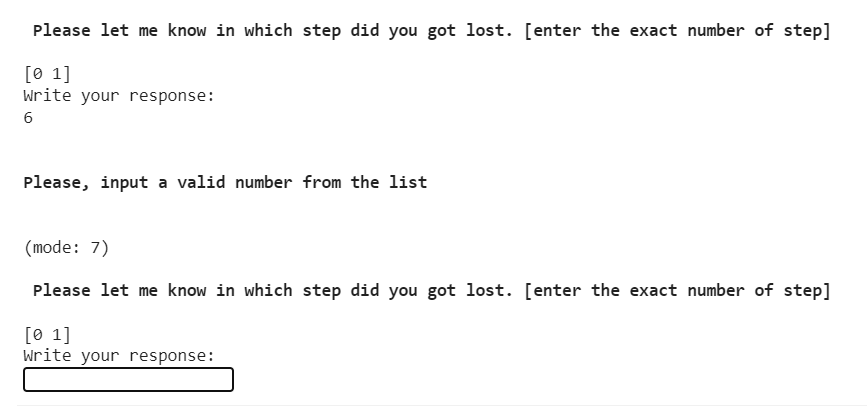
**Fig 13:** The user has rejected the recipe 5 times so the bot remembers that can choose ingredients for his recipe.

* 1. **Mode 7**

If the user has been lost during the preparation of the recipe, it is sent to this mode. Here we provide the possible steps where he/she got lost and the user has to choose where he/she wants to return.

****

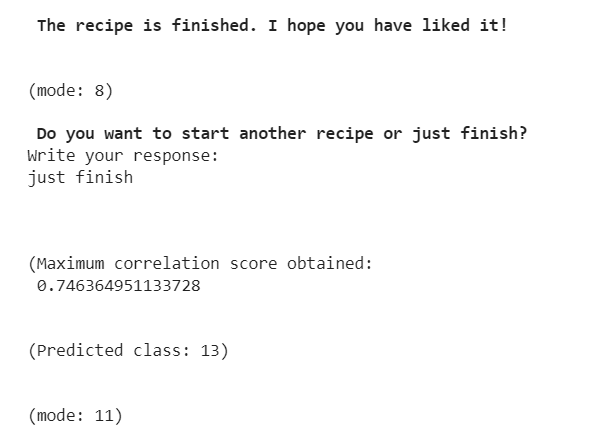
**Fig 14:** The user selects in which step he/she got lost and the bot returns to mode 3.

****

**Fig 15:** The user input’s a invalid number so we repeat the question.

* 1. **Mode 8**

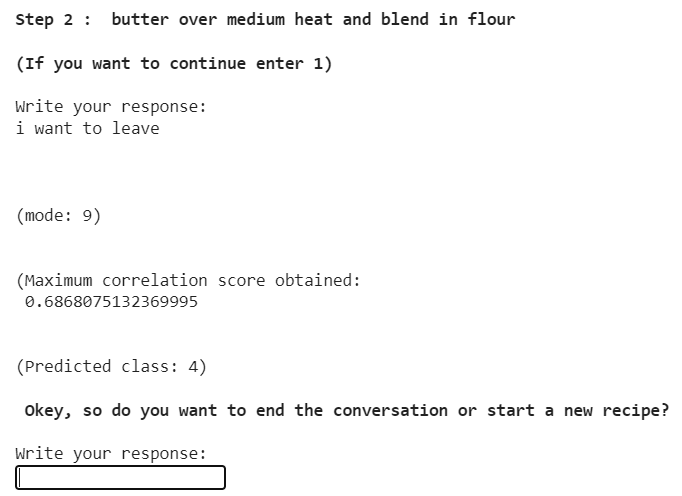
When the recipe is finished, the chatbot asks the user if they want to try another recipe or just end the conversation using this mode.



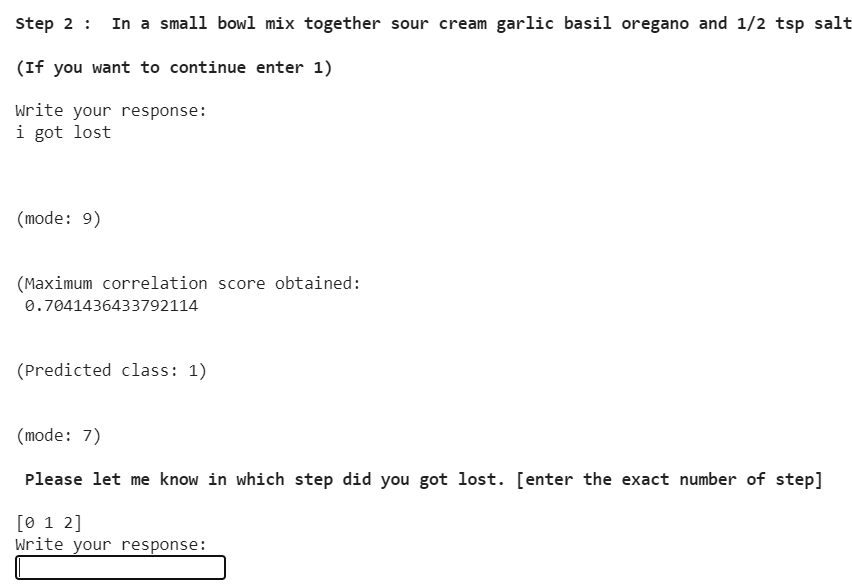
**Fig 16:** The user wants to leave (goes to mode 11)

* 1. **Mode 9**

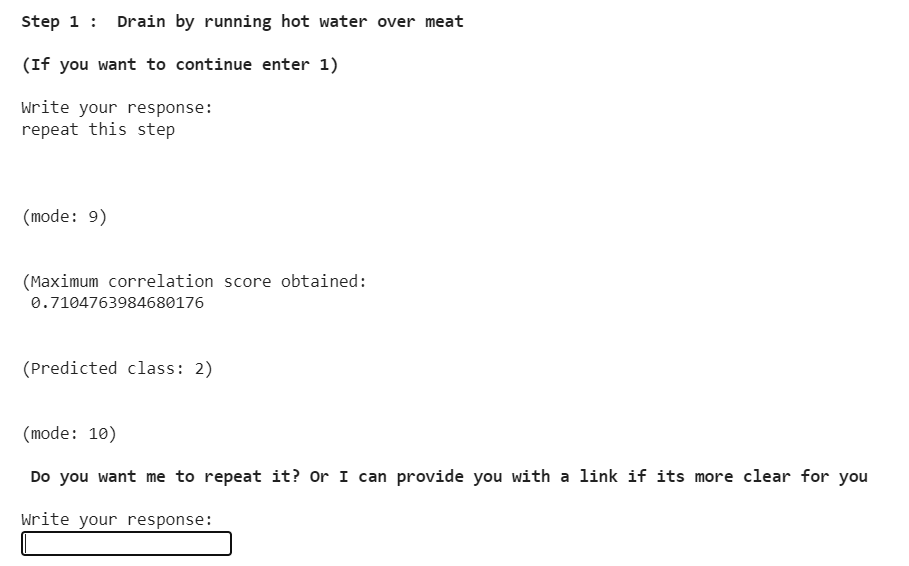
This mode is to have control of the user's intentions according to their input. If you want to continue, if you are lost, if you want to finish, if you don't put a coherent input, etc. And according to the result, it sends the user to one mode or another.



**Fig 17:** The user is done and wants to leave, before it we ask if he/she wants to start again or end. If the user wants to start again we reset all the variables and we begin again.



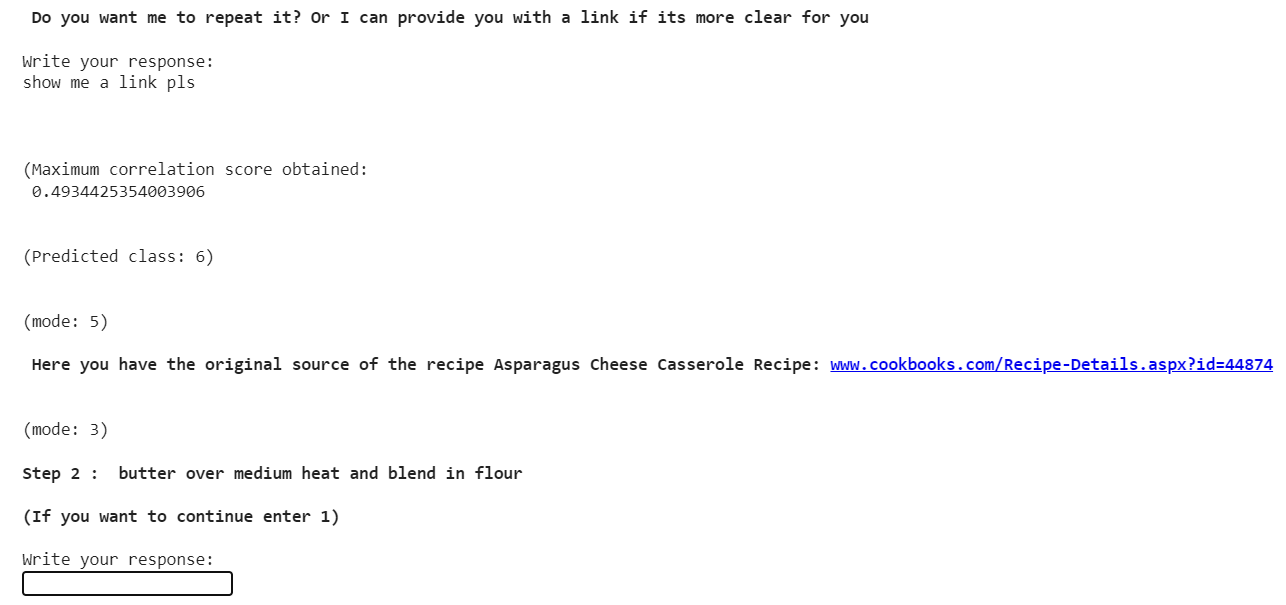
**Fig 18:** The user gets lost, and we ask in which step (we also show the list of steps). If the user doesn't put an intended number we proceed as always( repeating the question 3 times).



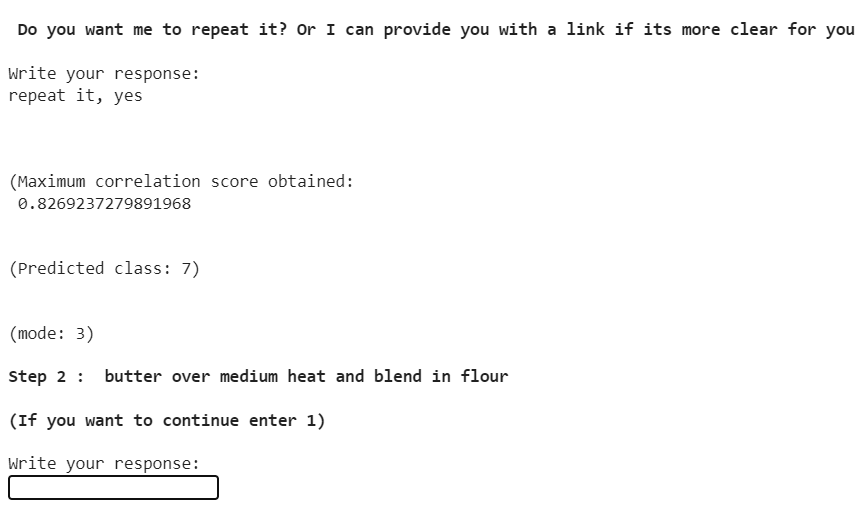
**Fig 19:** User wants to repeat this step

* 1. **Mode 10**

When detecting that the user has been lost in the last step provided, we check in this mode if he/she wants the chatbot to repeat it or to give him the original link. Using sentence similarity, and as always we do, if the user input has no sense we take it into account.



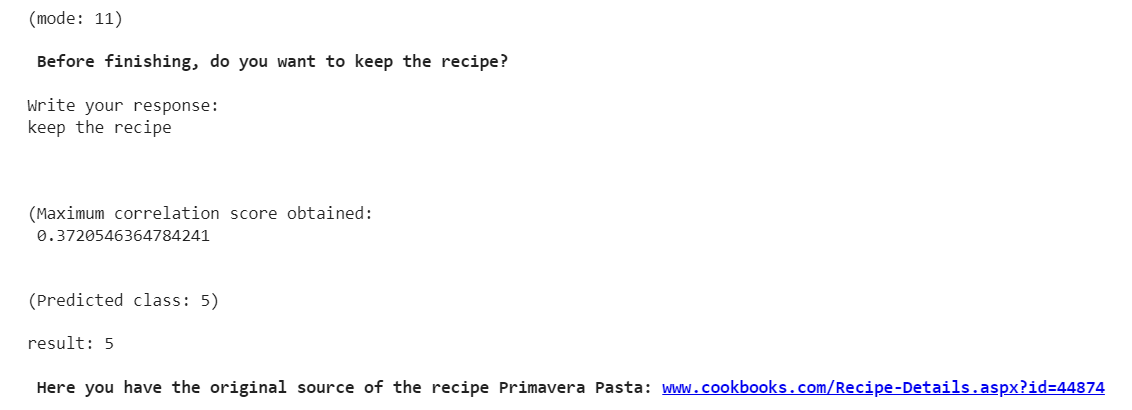
**Fig 20:** Example when the user says that he wants the link. Once the user has the link the conversation returns to the next step.



**Fig 21:** Example when the user says that he wants to repeat the step.

* 1. **Mode 11**

Finally, mode 11 is used to end the conversation, but before ending it, the user is asked if they want the original source of the recipe.



**Fig 22:** The user asks for the recipe before leaving.

1. **POSSIBLE IMPROVEMENTS**
   1. **Upgrade the recipe dataset**

The problem is that the lists of ingredients of some recipes make no sense (for example : random words like “something” or kitchen tools like “spoon”). Due to this we needed to find a smaller dataset (only 500 ingredients) to be able to get the only real ingredients. And for this reason, many ingredients cannot be identified by the system.

For that reason, a possible upgrade would be to clean the recipe dataset from those non-ingredient words, so we can get rid of the smaller dataset to identify only real ingredients.

* 1. **Fixing repetition recipes problem**

When we show the recipes to the user we are not taking into account the repetition of the recipes, i.e, when the user rejects a suggested recipe, the system could show the user the same recipe again at some point.

So having a register of the seen recipes could improve this aspect. We haven’t taken into account because, for recipes that match one or two ingredients, the output number of recipes is too big to really take it into account (the probability of seeing two recipes it's very low).

* 1. **Improving sentence similarity dataset**

The truth is that the context to be able to classify the dataset using sentence similarity is too poor. This impacts directly on the scores that we are getting in sentence similarity by missing the meaning of some user’s inputs.

So we could improve it by adding more sentences with the same idea and having people with more knowledge to write the sentences (We have to take into account that we are the ones writing those sentences and also the ones testing them, so this directly impacts on the results).

* 1. **Asking for more details to the user**

When we did the diagram we thought about asking many questions to the user. As an example, requesting for ingredients that the user doesn't want in his recipe, requesting for kitchen tools…

With more time that can be implemented, also there are datasets and models that can help with this task. For example, detecting if the recipe is for breakfast, for dinner …

1. **LAST WEEK CHANGES**

For last week Alexander told us few things to add to our code, so we implemented:

* 1. **Adding dialogue model**

As he told us, we implemented this model where the user needed to respond with a yes or no. Using this we can improve and take as a yes/no some similar words.

It works quite well, but the “yes” class does not. Some similar words that we would like to be accepted are rejected because they belong to other classes. So to adjust this, for the yes class we take into account 3 classes:

****

However, this does not happen with the no class.

* 1. **Fixing when the user says something with no sense**

We were not contemplating if the user said something with no correlation with the questions.

So to fix this we look if the max score of the best match for sentence similarity is higher than 0’3. If it's lower we ask for a rephrase and we count the number of times that the user said something with no sense. This way we can filter a lot of text and improve the conversation.If the user says something without sense 3 times, the chatbot stops the conversation. We decided to take 0’3 after testing some inputs.

* 1. **Shuffle the proposed recipes**

Another problem of the old code was that we were only taking 5 possible recipes to recommend, and once the recommendations finished, we did not restart them. This could cause the user to see the 5 already recommended recipes again.

To solve this problem we simply apply shuffle from the random library to the ids of the recipes to recommend, inside the recipes() and getRandomRecipes() functions.