Happie: Project Final Deliverable

the meal planning application using a microservice technology

FLORIS VOSSEBELD, University of Twente, Netherlands
EVAN FRANCISZOK, University of Twente, Netherlands

Introduction

Welcome to our detailed report on the ongoing development of our Intelligent Recipe and Meal Planning Application. This document provides an in-depth look into how we've built our application using microservices architecture.

Project Description

Our project revolves around creating a smart meal planning tool. The goal is to make it easier for users to plan meals, manage their pantry, and create optimized shopping lists that minimize the cost of the groceries. We also give the users the possibility of assigning preferences and diet restrictions. We realised this by using a microservice architecture. Services included: user profile management, recipe suggestions, inventory tracking, and shopping list optimization.

Report Structure

This report is organized into several main chapters, each focusing on a specific aspect of our project:

- Motivation: We explain why we have made a meal planning application and what we aimed to achieve.
- Business Process: Here, we explain the main functions that our application performs and how they benefit potential users.
- Architecture: This chapter breaks down how we designed our application using microservices architecture.
- Design Decisions: We talk about the technology and architecture choices we made and why we made them.
- Validation: We explain how we tested our application to make sure it works well.
- Relevant Information: Finally, we cover any other important details about our project.

Table of contents

Motivation

Our motivation for this project has come forward as we both are students that like to sport a lot and don't want to spend a lot of time having to think about what, how and with what ingredients we need to cook. Added to this, one of us does have a food restriction (vegetarian) that makes finding good recipes harder. As both of us are students that don't have a large budget to live with. We would like to minimize the cost of the meals that we will have to make, certainly as we live with other people and we will need to make meals for multiple people.

As normally meal planning can be a time-consuming task that requires a lot of comparisons, searching in books or on the internet. Especially when you want to cook a new meal with some dietary restrictions. Because there is no fun in cooking and eating the same meal over and over again. This resulted in our idea to make a meal planning application that will help with these problems and streamline the experience of cooking.

Our goal is to develop an Intelligent Recipe and Meal Planning Application that addresses these pain points and helps users to make informed decisions that align with their preferences, dietary constraints, while being possible with their lifestyle.

Our objectives include:

- A personalized page containing all the features that will help with streamlining the meal planning and cooking experience. Based on their dietary preferences, diet restrictions, allergies and cost of the meals.
- Tracking of the users Inventory and generating shopping lists. This enables users to efficiently manage their pantry inventory and supermarket trips.
- These features should be implemented in a flexible and scalable designed application that allows ease of integration with external services, future improvements and expansions.

With these objectives we will try to create an application that will solve our problems with meal planning, while being a good submission for the SOA course.

Business process

Our meal planning application supports several functionalities aimed at the simplification of meal planning and the thereby required activities. These processes are designed to streamline the meal planning experience for users and facilitates decision-making by providing suggestions.

- *Recipe Suggestions*:
  One of our primary business processes will be the recipe suggestion. This service will use AI, magic algorithms or other advanced methods to generate recipes based on the user's dietary restriction, allergies and preferences. This will help decision making by only providing possible recipes for the

user. It has to be said that for this submission we used a database with some test recipes that will be cross referenced with the user preference to generate our suggestions.

- *Inventory Management*:
  Also implemented in our application is inventory management. This will allow users to track their pantry items, quantities and the expiration dates. Currently can this information only be used as insight and for the meal planning and indirectly by the shopping list services. In the future this could be extended for other uses. For example, usage patterns.

- *Shopping List Optimization*:
  This business process enables users to automatically retrieve (optimized) shopping lists based on their meal planning and their pantry inventory. This will also show what supermarket to go to in order to get the best price for the groceries, taking promotions into account. This business process will ensure that the user only purchases items that the user needs and avoid unnecessary purchases.

- *Meal Planning*:
  The meal planning connects all the other business processes together and makes the application complete. The meal planning puts the chosen suggestions together and presents it to the users. The user is able to change some specific suggestions if they are not wanted.

These business processes together help achieve the overarching goal of our Meal Planning Application: to simplify meal planning, enhance user satisfaction, and streamline the whole process.

Architecture

Our Meal Planning Application is designed with a microservices architecture, this ensures scalability, flexibility, and modularity. because we use this architecture we are able to integrate various services, each responsible for specific functionalities. In this section, we will go over the architecture of our solution, describing the services defined and how they collaborate with each other.

In our application we implemented the following microservices:

- *Meal planning User Interface:*
  Even though we do not call it a service, it is. This service provides the user with the UI and represents the data from the other services so the user can easily view and interact with the data.
- *Meal Planning Service:*
  Facilitates meal planning by providing the best fitting personalized recipe suggestions, taking into account the current inventory availability.
- *Recipe Suggestion Service:*
  This service provides the Meal Planning Services with the meal suggestions based on the user preferences.
- *Inventory Management Service:*
  Tracks pantry items and their quantities, ensuring accurate inventory management and helps to facilitate informed meal planning and shopping list generation.

- *Shopping List Optimization:*
  Manages the generation and optimization of shopping lists based on user meal plans, pantry inventory, and current supermarket prices for the items.
- *User Profile Service:*
  Responsible for managing user profiles, including storing and retrieving user preferences, dietary restrictions, and allergies.
- *Jumbo Price Service:*
  Retrieves price information from the Jumbo supermarket, enabling cost-effective shopping list generation.
- *Albert Heijn Price Service:*
  Like the Jumbo Price Service, this service get the price information from the Albert Heijn supermarket. Getting prices of multiple supermarkets makes price comparison possible.

Architecture Explanation:

Because we have made a micro-service architecture, our solution exists out of several interconnected components. Each of these services is fulfilling a specific role in the meal planning application.

The user interface serves as the origin of all processes that will be started in the solution and as the only way for an user to interact with the application. To be able to represent the information to the user it needs to have access to data. This data is managed by multiple different other services. So the user interface has connections with some other services in order to start processes, retrieve data and update this date.

One of the services that the user interface will interact with is the meal planning service. This service will in turn act as a controller, like in an orchestration pattern. This service will orchestrate when and what the other services will have to do in order to generate the meal planning. This has to be done in order to fulfill the primary business process of the application: the meal planning. The planning is then stored in the associated database for retrieval by the user (interface).

Once there is a new meal planning available by the meal planning service, this is passed to the message queue from the shopping list optimization service. In this request all the required ingredients are sent that are still needed to complete the recipe.
Once this service has received this request to generate a shopping list it will go and retrieve the current prices of the connected supermarkets and generate a shopping list. The list will be optimized on the price. Such that the user will spend as little as possible.

There are some other services in play. These services and their respective relation with other services are represented in the figure below. In this diagram you can see that some of the services communicate synchronously and others asynchronous. Later on in this report we will clarify which ones and why we have chosen for this type of communication.

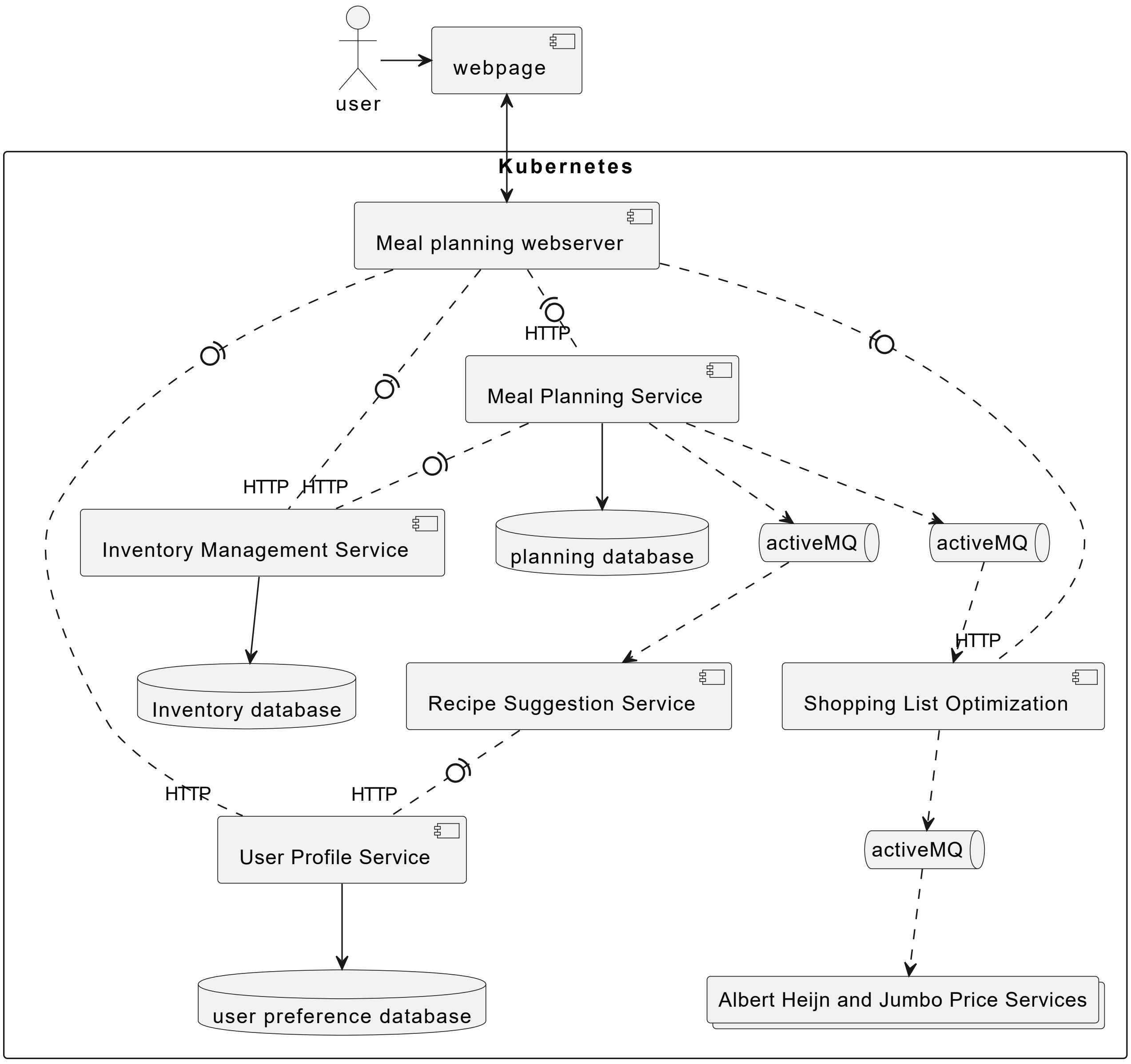

Figure 1: Component diagram

Synchronous vs. Asynchronous Services:

Synchronous Communication:

We opted for synchronous communication between all communication with the user profile and the inventory service. These two services are both services that do not have external dependencies and are only responsible for a small part of the data management of the application. We will not work with a dataset so big that this will even slow down the application to justify asynchronous communication. Most of the interactions with these services will be posting and retrieving data to be stored and doing minor alterations and transformations to this data.

The shopping list will be accessible via a message queue, but also with normal synchronous communication. We will argue why we have used a message queue later in this report. The synchronous part is only addressed by the user interface for retrieval of the shopping lists. As this is just a database query to retrieve any possibly generated list and can be done very quickly, we decided to use synchronous communication. This will also ensure efficient and responsive user interactions.

Asynchronous Communication:

In contrast, we employ asynchronous communication between the List Service and the Supermarket Service, as well as between the Planning Service and the Inventory and List services. This decision was driven by the desire to avoid blocking the application while fetching price information from third-party APIs and to streamline the meal planning process. By utilizing message queues, we can submit price inquiries in bulk and continue processing other tasks while awaiting responses, thereby enhancing the overall efficiency and responsiveness of our application.

In Contrast, we use asynchronous communication between the meal planning and the shopping list optimization service. Even though we have enabled synchronous communication between the user interface shopping list optimization service as mentioned above. When the meal service wants to communicate with the shopping list optimization service it does not require an immediate response. This is also not really possible as the shopping list service is dependent on external services that might delay a response.

The external services used by the shopping list optimization are all the price services of the supermarkets. For now there are two but this can become more in a possible future. These price services are directly communicating with API's of supermarkets and we assume these to be unreliable with regards to speed. For this reason we have also made this communication asynchronous. We can then submit multiple price inquiries and continue other processes.

The last service that uses asynchronous communication is the recipe suggestion service. We assume that our magic algorithm, AI or other method could take some time to generate the best suggestions. For now it will be very quick but in the future this could change.This could block other processes in the application. To prevent this we have made communication with this service asynchronous.

Justification of Message Queues:

We have chosen to implement message queues for all of our services that will use asynchronous communication. We have done this as all services have to perform tasks that take some time that could prevent the handling of new requests.

For instance the suggestion service could take a relatively long time to generate a new recipe. In that time there could be a new request to generate suggestions. To prevent this new request from being lost, we implemented a message queue when the meal planning communicates with the suggestion service. Then once the request is handled, the next task can be started. This is also the case for the price services. These services could potentially take more time as we cannot assume that the API of the supermarkets will respond quickly.

When the meal planning application sends a new list with ingredients to the shopping list optimization also with a message queue. This is done because we do not require a response for this request. And the price optimization service could potentially take a long time to finish the retrieval of all the prices. Also we do not want to lose the ingredient lists due to a HTTP timeout when this is not required in this case.

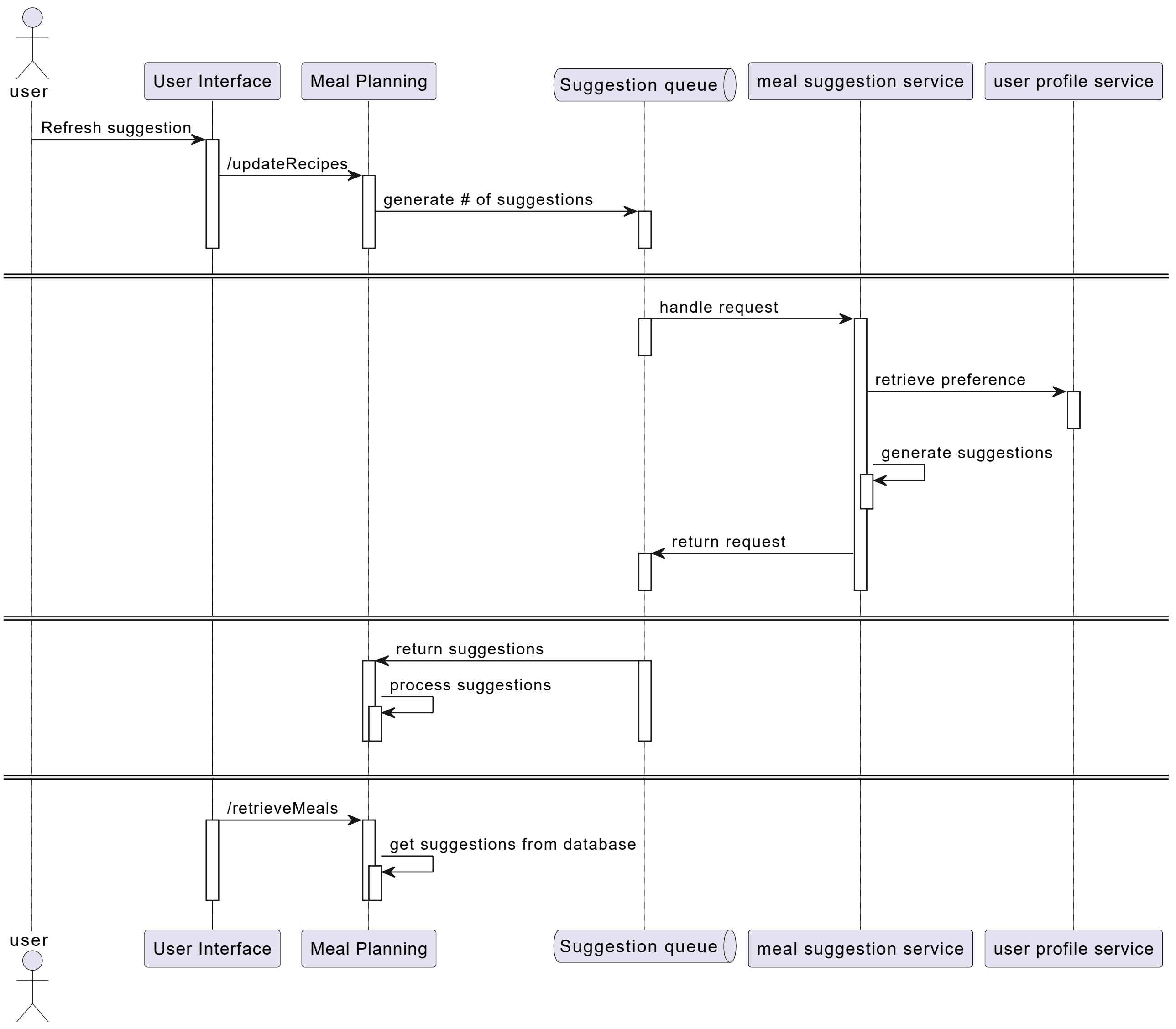

Figure 2: Sequence diagram

In the diagram above (Figure 2) we have shown a diagram to present the process flow for the generation of the recipes. This action can be started by the user. This request is then forwarded by the user interface to the meal service. The user ID is also passed along to the meal planning. When this information is passed to the suggestion services that can then in turn retrieve the user preferences from the user profile service. The "magic" algorithm can then generate the recipes.

This is then via the message queues passed back to the user interface through the process shown in the sequence diagram.

Design decisions

- The more concrete/specific design decisions, by explaining why you did (and didn't) use different technologies such as SOAP, REST and Message queue. Consider giving a summary table listing the services with the information about why you used (or did not use) these technologies and communication styles.

In the development of our meal planning application, we had to make several design decisions. Each of these decisions was based on the scenario where our application would be used by many people around the netherlands. Here the application should be able to run smoothly and be scalable while utilizing a micro-service architecture. So all of our design choices are aimed at the performance, scalability, and maintainability of the application. In this chapter we will outline these decisions, including the rationale behind each choice.

Architecture Patterns used in our architecture:

Decision: We used the orchestration pattern mainly for the Meal Planning Service so it could dictate the flow of the data while planning the meal. However the whole application utilizes more of a hybrid of the orchestration and choreography patterns.

Rationale: The orchestration pattern allows for centralized control and coordination of the interactions between the services. This simplifies the implementation and maintenance of the processes within the Meal Planning Service. This is not needed for the other services. These services can be more reactive and to their process independently. This makes the hybrid pattern easier to implement and more intuitive.

Synchronous and Asynchronous Communication:

Decision: The architecture in our application uses both asynchronous and synchronous communication between the different services. Synchronous communication is used where real-time data retrieval and immediate responses are more important. The places where asynchronous communication is used are at the places where we do not require an immediate response or where this can block the process for a long time.

Rationale: The rationale behind the decisions is better explained in the architecture chapter above.

Communication protocol:

Decision: We have chosen to use the REST protocol of SOAP for the communication between the services in our application. This decision was based on our experience with the REST protocol and allows for ease of implementation without a lot of overhead and configuration. Another reason is that we do not share sensitive data and therefore do not require the need for the extra security that can be offered by SOAP.

Rationale: RESTful API’s are lightweight and can do everything we need. Because of our experience with implementing this protocol, this reduced the development time of the application. REST simplifies the communication between the services.

Usage of Message Queue:

Decision: We implemented message queues for asynchronous communication between services, particularly for processes involving external integration of API’s and possibly long-running tasks.

Rationale: Message queues can provide a way to prevent blockage of processes and do not require a system to give immediate response when a request is done. So by implementing these queues in our application we prevent long loading times and potential blocking of the application.

Data storage:

Decision: To store data in our system we have chosen to implement persistent volumes with MongoDB. This data will be stored in the volumes of docker.

Rationale: We have chosen a persistent volume so we can restart the application and the information will still be available when it is opened up again. This is needed in our application because all the information that the user wants has to be available for longer periods of time. At this time we do not want to “forget” that data.

Validation

In order to ensure that the application is working and that is correctly performs all the business processes that we have stated, we have conducted some tests. These tests include walking through our application (by using the user interface) and validating the communication between the services for some processes. For this second type of testing we have concrete evidence included in the attachments.

Testing:

We have tested the application by interaction with the user interface and the features and processes contained in this application. Testing included, creating a new user by logging in with a new account, adding allergies to the user preference, adding diet restrictions to the user preference, requesting new suggestions, retrieving the generated shopping list and some more features that our user interface offers.

This method of testing makes sure that the whole application under normal conditions is working. This has all been done while the application is running in Kubernetes. Because all our tests were successful, we can conclude that all of the services are running and communicating with each other successfully.

Usage Information:

In addition to the test method we have just described. Have we also conducted usage analysis by walking through specific processes within the application and recording the communication between services. This provides a more detailed view in the dataflow while these processes are running. A full transcript of this test has been included in the attachments. We have tested the following two processes, the Refreshing of the suggestions and the Fetching of these suggestions. These are the same processes that are modeled in the architecture chapter.

Example Validation Scenarios:

Refreshing Suggestions:

- Step 1: The user presses the refresh button to update meal suggestions. The frontend sends a message to the backend.
- Step 2: The backend forwards the message to the Meal Planning Service.

- Step 3: The Meal Planning Service sends a message to the 'recipe-queue' message queue.
- Step 4: The Recipe Suggestion Service processes the message from the 'recipe-queue' and sends a response to the 'recipe-response-queue'.

Fetching Suggestions:

- Step 1: The meal planning UI attempts to refresh meal suggestions by fetching from the client side to the backend for frontend.
- Step 2: The backend sends a request to the Meal Planning Service.
- Step 3: The client side receives the response from the Meal Planning Service.

With these tests and usage analysis cases we think we have verified enough of our application to say with a high degree of certainty that the application behaves like expected and works. Also with this method of testing we can provide the reader of concrete evidence of the working of the meal planning application.

Relevant information

In this chapter we will go over some additional extra information and insights about our meal planning application. To make sure all relevant information is contained in this report.

One of the aspects that is not covered before in this report is the integration with an authentication service. We have integrated oauth2 support, for now we have chosen to use the oauth2 from GitHub. We have implemented oauth2 to enhance security and the user experience. The user or tester of our application can just log in to their (currently) GitHub account and a new user is automatically created. This streamlines the login process and does not require the need to create a new password and other necessities when creating your own login.
We have chosen for GitHub because of the high likelihood for the current users and testers to have a GitHub account.

Conclusion

In conclusion, in the development of our meal planning application we have tried to reduce the issue of the time consuming traditional meal planning methods (especially when there are diet restrictions or allergies). And created a micro-service architecture that should be a good submission for the SOA course.

We have created a relatively complex architecture that can manage the meal planning in a flexible, scalable and maintainable manner. With the usage of different communication protocols and methods we can keep the application feeling responsive. Even when some of the services will take longer. The communication in our project is tested by walking through the process flow and looking at the responses and requests of each of the services. Hereby confirming that it works as intended and verifying the workings of all the business processes.

Further improvements to the application could be a completely working method of recipe suggestion service. And the integration with multiple supermarket API's. Also some of our logic is currently not production final and just imitates expected behavior. Overall does the application achieve what is wanted from the application for the "service oriented architecture" goals.

Acknowledgements

In this project we have used ChatGPT 3.5 and 4.0 as a supportive tool, providing structure to our paper and aiding in the identification and correction of spelling mistakes. We did also use ChatGPT for debugging and understanding errors while we were developing the application.
It is important to note that all content generated by ChatGPT underwent thorough review and validation by us. We take full responsibility for the final output.

Attachments

Refreshing suggestions

After pressing the refresh button to refresh the meals Frontend sends message to backend. Both in Next.JS

```
POST to http://localhost:8085/ with body generated by nextjs:

Body : ["66192bb8c7bd0a6d4be881e8"]
```

Backend for frontend sends message to the mealplanningservice:

```
POST request to http://localhost:8082/updateRecipes?
userId=661d3f6f1d0684e3cc565e1a
```

Message queue message to ‘recipe-queue’ from the mealplanning service

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<RecipeRequest xmlns="http://www.example.com/api/events/reciperequest"
xmlns:ns2="http://www.example.com/api/events/shoppinglistrequest"
xmlns:ns3="http://www.example.com/api/events/recipesresponse">

    <userId>66192bb8c7bd0a6d4be881e8</userId>

    <numberOfRecipes>7</numberOfRecipes>

</RecipeRequest>
```

Message queue message to ‘recipe-response-queue from the recipe suggestion service

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ns2:Recipes xmlns="http://www.example.com/api/events/reciperequest"
xmlns:ns2="http://www.example.com/api/events/recipesresponse">

    <ns2:Recipe>
        <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
        <ns2:id>66240c45475b5a21c48ba43d</ns2:id>
        <ns2:title>Apple cinnamon granola bars</ns2:title>
        <ns2:ingredient>2 cup old fashioned oats</ns2:ingredient>
        <ns2:ingredient>3/4 cup wheat germ</ns2:ingredient>
        <ns2:ingredient>1/2 cup ground flax seed</ns2:ingredient>
        <ns2:ingredient>1/2 stick butter</ns2:ingredient>
        <ns2:ingredient>2/3 cup Brown sugar</ns2:ingredient>
        <ns2:ingredient>1/2 cup honey</ns2:ingredient>
        <ns2:ingredient>2 tsp vanilla</ns2:ingredient>
        <ns2:ingredient>1/2 tsp kosher salt</ns2:ingredient>
        <ns2:ingredient>8 oz dried and chopped apples</ns2:ingredient>
        <ns2:ingredient>1 dash cinnamon</ns2:ingredient>
        <ns2:NER>oats</ns2:NER>
        <ns2:NER>ground flax</ns2:NER>
        <ns2:NER>butter</ns2:NER>
        <ns2:NER>Brown sugar</ns2:NER>
        <ns2:NER>honey</ns2:NER>
        <ns2:NER>vanilla</ns2:NER>
        <ns2:NER>kosher salt</ns2:NER>
        <ns2:NER>apples</ns2:NER>
        <ns2:NER>cinnamon</ns2:NER>
    </ns2:Recipe>
    <ns2:Recipe>
    <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
    <ns2:id>66240c45475b5a21c48ba2d4</ns2:id>
    <ns2:title>Hibiscus Sangria</ns2:title>
    <ns2:ingredient>1 cup Grapes, Washed And De-stemmed</ns2:ingredient>
    <ns2:ingredient>1 cup Boiling Water</ns2:ingredient>
    <ns2:ingredient>1/4 cups Dried Hibiscus Flowers (also Called Flor De
Jamaica)</ns2:ingredient>
    <ns2:ingredient>1 Tablespoon Heaping, Sugar</ns2:ingredient>
    <ns2:ingredient>2 cups Cheap Red Wine</ns2:ingredient>
    <ns2:ingredient>1 whole Large Ripe Peach, Thinly Sliced</ns2:ingredient>
    <ns2:NER>Grapes</ns2:NER>
    <ns2:NER>Boiling Water</ns2:NER>
    <ns2:NER>¼</ns2:NER>
    <ns2:NER>Sugar</ns2:NER>
    <ns2:NER>Cheap Red Wine</ns2:NER>

    </ns2:Recipe>

    <ns2:Recipe>
    <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
    <ns2:id>66240c45475b5a21c48ba45f</ns2:id>
    <ns2:title>Grainery Bread</ns2:title>
    <ns2:ingredient>4 cups boiling water</ns2:ingredient>
    <ns2:ingredient>12 cup cracked red wheat berries</ns2:ingredient>
    <ns2:ingredient>12 cup millet</ns2:ingredient>
    <ns2:ingredient>13 cup honey</ns2:ingredient>
    <ns2:ingredient>3 tablespoons yeast</ns2:ingredient>
    <ns2:ingredient>12 cup oats</ns2:ingredient>
    <ns2:ingredient>12 cup coarse cornmeal</ns2:ingredient>
    <ns2:ingredient>12 cup dry nonfat dry milk powder</ns2:ingredient>
    <ns2:ingredient>14 cup wheat germ</ns2:ingredient>
    <ns2:ingredient>2 teaspoons coarse sea salt</ns2:ingredient>
    <ns2:ingredient>1 cup shelled sunflower seeds</ns2:ingredient>
    <ns2:ingredient>7 cups whole wheat flour</ns2:ingredient>
    <ns2:ingredient>egg wash</ns2:ingredient>
    <ns2:ingredient>14 cup egg substitute</ns2:ingredient>
    <ns2:ingredient>1 tablespoon water</ns2:ingredient>
```

```
        <ns2:ingredient>coarse sea salt</ns2:ingredient>
        <ns2:NER>boiling water</ns2:NER>
        <ns2:NER>berries</ns2:NER>
        <ns2:NER>millet</ns2:NER>
        <ns2:NER>honey</ns2:NER>
        <ns2:NER>yeast</ns2:NER>
        <ns2:NER>oats</ns2:NER>
        <ns2:NER>coarse cornmeal</ns2:NER>
        <ns2:NER>milk</ns2:NER>
        <ns2:NER>germ</ns2:NER>
        <ns2:NER>salt</ns2:NER>
        <ns2:NER>sunflower seeds</ns2:NER>
        <ns2:NER>whole wheat flour</ns2:NER>
        <ns2:NER>egg wash</ns2:NER>
        <ns2:NER>egg substitute</ns2:NER>
        <ns2:NER>water</ns2:NER>
        <ns2:NER>salt</ns2:NER>

        </ns2:Recipe>

        <ns2:Recipe>
        <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
        <ns2:id>66240c45475b5a21c48ba445</ns2:id>
        <ns2:title>Sweet And Sour Sauce</ns2:title>
        <ns2:ingredient>1/2 cup pineapple juice</ns2:ingredient>
        <ns2:ingredient>1 tablespoon cornstarch</ns2:ingredient>
        <ns2:ingredient>1/4 cup brown sugar</ns2:ingredient>
        <ns2:ingredient>1/4 cup white vinegar</ns2:ingredient>
        <ns2:ingredient>1 teaspoon soy sauce</ns2:ingredient>
        <ns2:ingredient>1 tablespoon ketchup</ns2:ingredient>
        <ns2:ingredient>1/4 cup pineapple, in small chunks</ns2:ingredient>
        <ns2:NER>pineapple juice</ns2:NER>
        <ns2:NER>cornstarch</ns2:NER>
        <ns2:NER>brown sugar</ns2:NER>
        <ns2:NER>white vinegar</ns2:NER>
        <ns2:NER>soy sauce</ns2:NER>
        <ns2:NER>ketchup</ns2:NER>
        <ns2:NER>pineapple</ns2:NER>

        </ns2:Recipe>

        <ns2:Recipe>
        <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
        <ns2:id>66240c45475b5a21c48ba3f9</ns2:id>
        <ns2:title>Chocolate French Toast</ns2:title>
        <ns2:ingredient>2 Eggs</ns2:ingredient>
        <ns2:ingredient>1/3 cups Milk</ns2:ingredient>
        <ns2:ingredient>2 Tablespoons Vanilla Extract</ns2:ingredient>
        <ns2:ingredient>4 slices White Or Whole Wheat Bread</ns2:ingredient>
        <ns2:ingredient>2 teaspoons Nutella Or Chocolate Spread</ns2:ingredient>
        <ns2:NER>Eggs</ns2:NER>
        <ns2:NER>Milk</ns2:NER>
        <ns2:NER>Vanilla</ns2:NER>
        <ns2:NER>White</ns2:NER>
```

```
    <ns2:NER>Nutella</ns2:NER>

    </ns2:Recipe>

    <ns2:Recipe>
    <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
    <ns2:id>66240c45475b5a21c48ba395</ns2:id>
    <ns2:title>Graham Cracker Fruit Cake</ns2:title>
    <ns2:ingredient>1 box crushed graham crackers</ns2:ingredient>
    <ns2:ingredient>1 box seedless raisins</ns2:ingredient>
    <ns2:ingredient>1 jar cherries</ns2:ingredient>
    <ns2:ingredient>1 jar or box candied mixed fruit</ns2:ingredient>
    <ns2:ingredient>1 stick margarine</ns2:ingredient>
    <ns2:ingredient>2 c. pecans</ns2:ingredient>
    <ns2:ingredient>1 large pkg. marshmallows</ns2:ingredient>
    <ns2:NER>graham crackers</ns2:NER>
    <ns2:NER>raisins</ns2:NER>
    <ns2:NER>cherries</ns2:NER>
    <ns2:NER>margarine</ns2:NER>
    <ns2:NER>pecans</ns2:NER>
    <ns2:NER>marshmallows</ns2:NER>

    </ns2:Recipe>

    <ns2:Recipe>
    <ns2:userId>66192bb8c7bd0a6d4be881e8</ns2:userId>
    <ns2:id>66240c45475b5a21c48ba2d1</ns2:id>
    <ns2:title>Frozen Orange Balls</ns2:title>
    <ns2:ingredient>1 (12 oz.) box vanilla wafers</ns2:ingredient>
    <ns2:ingredient>3 c. confectioners sugar</ns2:ingredient>
    <ns2:ingredient>1/2 c. butter, melted</ns2:ingredient>
    <ns2:ingredient>1 (6 oz.) can orange juice (frozen)</ns2:ingredient>
    <ns2:ingredient>1 c. nuts, chopped</ns2:ingredient>
    <ns2:ingredient>1/2 c. coconut</ns2:ingredient>
    <ns2:NER>vanilla wafers</ns2:NER>
    <ns2:NER>confectioners sugar</ns2:NER>
    <ns2:NER>butter</ns2:NER>
    <ns2:NER>orange juice</ns2:NER>
    <ns2:NER>nuts</ns2:NER>
    <ns2:NER>coconut</ns2:NER>

    </ns2:Recipe>

</ns2:Recipes>
```

Fetching suggestions

The meal planning UI tries to refresh the meals by fetching from the client side to the backend for frontend:

Then the backend does a request to the meal planning service:

```
GET request to mealplanningservice:8080/retrieveMeals?userId=
661d3f6f1d0684e3cc565e1a&fromDate=22-04-2024&toDate=28-04-2024
```

The client side gets the following back:

```
0:["$@1",["pSFr-Wf9KWCar5uxFep3O",null]]

1:[
  {
    "id": "66241a80c3bfe265b7742ca0",
    "name": "Fresh Fruit Salad",
    "ingredients": [
      "2 oranges, peeled and sectioned",
      "2 apples, cubed",
      "2 bananas, sliced",
      "Salad Dressing"
    ],
    "date": "2024-04-21",
    "userId": "66192bb8c7bd0a6d4be881e8",
    "ner": [
      "oranges",
      "apples",
      "bananas",
      "Salad Dressing"
    ]
  },
  {
    "id": "66261a656516aa406e69bb1b",
    "name": "Chocolate Chip Cookies",
    "ingredients": [
      "1 c. granulated sugar",
      "1 c. brown sugar",
      "2 c. butter flavored Crisco",
      "2 eggs",
      "2 tsp. vanilla",
      "3 1/2 c. flour",
      "1 tsp. baking soda",
      "1 c. pecans",
      "1 (12 oz.) pkg. chocolate chips"
    ],
    "date": "2024-04-22",
    "userId": "66192bb8c7bd0a6d4be881e8",
    "ner": [
        "sugar",
        "brown sugar",
        "butter",
        "eggs",
        "vanilla",
        "flour",
        "baking soda",
        "pecans",
        "chocolate chips"
      ]
    },
    {
      "id": "66261a656516aa406e69bb1c",
      "name": "Eat For Eight Bucks: Bean Gratin Recipe",
      "ingredients": [
        "1/2 cup fresh breadcrumbs",
        "Olive oil",
        "Salt",
        "1 1/4 cups dry cranberry or borlotti beans, soaked if you prefer to soak
beans before cooking",
        "1/2 smallish onion, finely diced",
        "1 small carrot, peeled and finely diced",
        "1 small stalk celery, finely diced",
        "4 garlic cloves, thinly sliced",
        "6 fresh sage leaves, chopped",
        "1/2 cup chopped tomatoes, fresh or canned"
      ],
      "date": "2024-04-23",
      "userId": "66192bb8c7bd0a6d4be881e8",
      "ner": [
        "fresh breadcrumbs",
        "Olive oil",
        "Salt",
        "borlotti beans",
        "smallish onion",
        "carrot",
        "stalk celery",
        "garlic",
        "sage",
        "tomatoes"
      ]
    },
    {
      "id": "66261a656516aa406e69bb1d",
      "name": "Quickens",
      "ingredients": [
        "2 cups sugar",
        "1 stick butter",
        "4 tbsp. cocoa powder",
        "1/2 cup milk",
        "3 cups oatmeal",
        "1/2 cup chunky peanut butter",
        "1 tsp. vanilla",
        "1 dash salt"
      ],
      "date": "2024-04-24",
      "userId": "66192bb8c7bd0a6d4be881e8",
      "ner": [
        "sugar",
        "butter",
        "cocoa powder",
        "milk",
        "oatmeal",
        "chunky peanut butter",
        "vanilla",
        "salt"
      ]
    },
    {
      "id": "66261a656516aa406e69bb1e",
      "name": "Hot Chicken Salad",
      "ingredients": [
        "2 c. diced chicken",
        "1 c. chopped celery",
        "1/2 c. slivered almonds or water chestnuts",
        "1/2 c. diced green pepper",
        "1/2 c. pimentos",
        "1/2 tsp. salt",
        "1 Tbsp. lemon juice",
        "1 can cream of chicken soup",
        "1/2 c. Hellmann's mayonnaise",
        "2 Tbsp. minced onion",
        "2 c. crushed potato chips"
      ],
      "date": "2024-04-25",
      "userId": "66192bb8c7bd0a6d4be881e8",
      "ner": [
        "chicken",
        "celery",
        "water",
        "green pepper",
        "pimentos",
        "salt",
        "lemon juice",
        "cream of chicken soup",
        "mayonnaise",
        "onion",
        "potato chips"
      ]
    },
    {
      "id": "66261a656516aa406e69bb1f",
      "name": "Stuffed Hamburger-Cabbage Buns (Runzas Or Bierocks)",
      "ingredients": [
        "1 lb frozen white bread dough, thawed (equals 2 loaves)",
        "1 lb ground beef",
        "2 cups shredded cabbage",
        "1 cup chopped onion",
        "1/4 cup chopped parsley (fresh)",
        "1/2 teaspoon salt",
        "1/2 teaspoon pepper",
        "1 teaspoon minced garlic",
        "1 cup cheddar cheese",
        "1/4 cup ketchup",
        "1 egg, slightly beaten",
        "1 tablespoon milk",
        "kosher salt (optional)",
        "cracked pepper (optional)"
      ],
      "date": "2024-04-26",
      "userId": "66192bb8c7bd0a6d4be881e8",
      "ner": [
        "frozen white bread dough",
        "ground beef",
        "cabbage",
        "onion",
        "parsley",
        "salt",
        "pepper",
        "garlic",
        "cheddar cheese",
        "ketchup",
        "egg",
        "milk",
        "kosher salt",
        "cracked pepper"
      ]
    },
    {
      "id": "66261a656516aa406e69bb20",
      "name": "Kansas City Spareribs",
      "ingredients": [
        "3/4 cup (packed) golden brown sugar",
        "1/2 cup paprika",
        "2 1/2 tablespoons coarse salt",
        "2 1/2 tablespoons ground black pepper",
        "1 tablespoon onion powder",
        "1/2 teaspoon cayenne pepper",
        "3 large racks spareribs (about 9 pounds)",
        "8 pounds (about) 100% natural lump charcoal or charcoal briquettes",
        "4 cups (about) hickory wood smoke chips, soaked in cold water at least
30 minutes",
        "1 1/2 cups purchased tomato-based barbecue sauce (such as KC
Masterpiece)"
      ],
      "date": "2024-04-27",
      "userId": "66192bb8c7bd0a6d4be881e8",
      "ner": [
        "golden brown sugar",
        "paprika",
        "coarse salt",
        "ground black pepper",
        "onion powder",
        "cayenne pepper",
        "spareribs",
        "lump",
        "minutes",
        "barbecue sauce"
      ]
    }
  ]
```