Mondragon Unibertsitatea
Goi Eskola Politeknikoa
Escuela Politécnica Superior

JanAI

3RD YEAR OF COMPUTER ENGINEERING

Authors:

LUKEN IRIONDO, ANGE MAIZTEGUI, JULEN GALLASTEGUI,

MARINA EIGUREN, EKI ZUZAETA, LUCAS PALMEIRA

23rd of January 2025

Final version

ABSTRACT

Nutrition is a very important part of life and a healthy and balanced diet is the key to good health. However, a large part of society finds it difficult to control their diet, either due to lack of knowledge or laziness, and many people give up on their diet. To help solve this problem, JanAI has been created, a nutritional assistant that can identify different types of food using image recognition, provide their nutritional values, and track the user's dietary intake. In addition, JanAI can provide text-based dietary advice based on the user's information.

RESUMEN

La nutrición es una parte muy importante de la vida y una dieta sana y equilibrada es la clave de una buena salud. Sin embargo, a gran parte de la sociedad le resulta difícil controlar su dieta, ya sea por falta de conocimientos o por pereza, y muchas personas abandonan su alimentación. Para ayudar a resolver este problema, se ha creado JanAI, un asistente dietético que puede identificar diferentes tipos de alimentos mediante el reconocimiento de imágenes, proporcionar sus valores nutricionales y realizar un seguimiento de la ingesta dietética del usuario. Además, JanAI puede proporcionar consejos dietéticos basados en texto a partir de la información del usuario.

LABURPENA

Elikadura egunerokotasunean oso garrantzitsua da eta dieta osasuntsu eta orekatu bat funtsezkoa da osasunerako. Hala ere, gizartearen zati handi bati elikadura zaintzea kostatzen zaio, ezjakintasuna edo alferkeria dela eta, jende askok dietari uko egiten diote. Arazo honi aurre egiteko sortu da JanAI, irudi errekonozimendu bidez janari mota ezberdinak identifikatu eta balio nutrizionala emateko, eta erabiltzaileen janari kontsumizioa jarraitzen duen asistente nutrizionala. Horretaz aparte, JanAI-k aholku nutrizionalak emateko gai da testu bidez.

INDEX

IMAGE INDEX

1. Introduction

Nutrition has always played an important role in everyone's life, but with today's fast-paced lifestyles, people don't take the time to eat right, take care of themselves or even make sure that what they eat is good for them. Moreover, only a small percentage of the population has at least a minimal knowledge of nutrition, so it is quite normal to see an unbalanced diet.

This can lead to several major problems, such as obesity, diabetes, anemia, malnutrition... Overall, these diseases cause a very significant reduction in quality of life, and in some extreme cases lead to death. With this in mind, the main aim of the project is to help people make dietary choices that lead to a healthier lifestyle.

In order to provide a solution to this problem, it is important to understand that the main purpose of this project is to provide web-based services to track the user's diet, recommend their daily calorie intake and give them information about the meal they are eating, while being accessible to a large public. All of this has been developed as a web application called JanAI, so that anyone can use it free of charge, and new functionalities are locked for certain users. In other words, the application has been developed with accessibility and usability in mind, since the main result must be to improve the user's quality of life, and the resources needed to access the application are basic by today's standards.

A number of technologies were used to develop this project. Firstly, in order to have an interactive interface, the prototype was created using Node Red's (O'Leary & Conway-Jones, 2013) dashboard, which was also used to consume services from the Spring web server, which contains all the controllers. As the amount of data to be stored is large and varied, a database with the necessary tables was also created using MySQL (MySQL AB, 1995). The highlight of this product is the inclusion of an AI model that is able to classify images and an LLM model that has the ability to chat with a user and offer help with any food-related doubts the user may have. To connect the main architecture, a Spring server and a Flask API (Ronacher, 2010) were developed.

In terms of programming languages, the backend was developed exclusively in Java (Sun microsystems, 1995) to work with objects (user, food...), and the Node Red dashboard eliminates the need for an HTML front-end. Node Red (O'Leary & Conway-Jones, 2013) was also used to consume the server's web services and to connect to the AI and LLM parts with a series of nodes. As a database was also required, SQL (Donald D. Chamberlin & Raymond F. Boyce, 1974) was used to create and manage the data tables. There was also a synchronisation problem, so a simulation was developed in Python (Guido Van Rossum, 1991) , although it does not work directly with the main product.

After this introduction, the direction of the report is explained. The next section will analyse the state of the art, taking into account the existing alternatives on the market. After that, the project's objectives will be commented, before looking at the impact of JanAI in today's society according to the SDGs (United Nations, 2015). The planification of the project will also be mentioned, before the main explanation of the product and the explanation of each part. Finally, the report will end with conclusions and future improvements.

2. State of the art

This section analyses the products available on the market today that could serve as a replacement for JanAI, focusing on the improvements that JanAI offers.

There are many applications that can do anything the user wants, from the simplest tasks to the most complex. Web applications and nutrition are not topics that have not gone together before, although it is often more common to see this in a mobile application format. It is clear that nutrition has its market for applications, but many tools come at a cost that not everyone can afford.

On the one hand, there is Calorie Mama (Azumio Inc., 2017) , a food recognition app that uses deep learning algorithms. Its main aim is to track meals and provide information about a particular dish through an image. It also offers personalized meal plans and recipes, including vegan diets and dietician-approved recipes. It also integrates with other health and fitness apps, allowing users to track their exercise and daily steps, as well as their food intake. It is primarily free for mobile devices and has very strong features, some of which are locked unless a premium plan is purchased, including personalized meal plans and nutritionist-approved recipes, video workouts and ad removal.

It’s users include health-conscious individuals who want to actively track their meals and maintain a balanced diet, fitness enthusiasts who use it to match their diet with their fitness goals, busy professionals who have a hectic schedule and don't have much time to decide what to eat, or even users who need to follow a special diet, such as vegans or celiacs.

On the other hand, there is LogMeal (AIGecko Technologies S.L., n.d.) , which is also a type of food recognition mobile application that provides calorie counts and macronutrient breakdowns. It has the largest food data set in the world, so it can identify a very wide range of foods. It works in a similar way to Calorie Mama (Azumio Inc., 2017) , providing tailored dietary recommendations for many types of users, including dietitians, athletes, self-checkout solutions for restaurants and even hospitals. It also has collaborators such as Nestlé, Porsche, Unilever and professional sports teams.

The main difference between these two options is that Calorie Mama (Azumio Inc., 2017) is aimed at an individual user who is tracking their diet for weight loss, maintenance or fitness, while LogMeal (AIGecko Technologies S.L., n.d.) is more of a corporate solution aimed at nutritionists, dietitians, healthcare providers, restaurants and some hospitals.

What sets JanAI apart from the competition is its user-friendly plan, which makes all the features available from the outset and does not require a subscription, although there is a premium plan that prioritizes premium requests. It is also worth noting that JanAI takes allergies into account. Both of these competitors are great applications, but LogMeal (AIGecko Technologies S.L., n.d.) is more professional and Calorie Mama (Azumio Inc., 2017) does not offer all of its features for free.

3. Objectives

After researching some options in today's market, the objectives of the project are interpreted. As you can see from the list below, two main objectives have been defined, each with its own segments.

- Improve user's quality of life:
  - **Help with weight management**: A poor and unbalanced diet leads to diseases such as obesity, which in turn leads to a poorer quality of life. The usage rate of nutrition apps is expected to reach 4.4% of the global population last year (Statista, 2023), so it is a great way to implement nutritional help.
  - **Improve user's diet**: By inserting an LLM to chat with, the user can ask if it is right to eat a certain food or recommend something to eat based on some information such as allergies, intention to gain or lose weight or what they have eaten in the last few days. By recommending healthy options, the main goal is to reduce the obesity rate from 1 in 8 to 1 in 7 (World Health Oganization, 2024).
  - **Reminder to drink water**: It is common to forget to drink water, especially when you are on a tight schedule. To remedy this, the recommended daily calorie intake is accompanied by a recommendation on how many glasses of water to drink.

- Make an accessible and easy to use application:
  - **Create an app with no premium limitations**: Many of these apps are "freemium", meaning that they can be used for free, but the most interesting features have to be paid for. Another goal was to increase usage by making all functionalities free, taking advantage of the increase in the use of nutrition apps (Statista, 2023).
  - **Make the interface easy to use**: Nutrition is a foreign concept for many people due to the amount of things that need to be taken into account (calories, proteins, fats...), so the application had to be easy to use even for a beginner. This has been achieved by using graphs to display the macros consumed, the user's weight graphs and graph-based meal information.

To summarize, the app helps users to lead a healthier lifestyle, with the aim of improving people's quality of life by improving nutrition, helping with weight management and reminding them to stay hydrated, while having an application that has no premium restrictions and is easy to use.

4. Description of the product

Continuing with the report, this section will provide a detailed description of the product.

As explained earlier, the aim of this product is to create an accessible and easy-to-use application to improve the quality life of its users. To this end, a web application for JanAI has been developed.

The application offers two types of users: premium and non-premium. The user can register by entering his/her own information and choose whether he/she wants to be premium or not. After creating an account, the user can access the application by logging in. Inside, he/she can access the different services.

One of these is information about the user's daily macros, which are displayed in graphs. The user can also check how many glasses of water are needed to drink and how much he has drunk. If the user drinks a glass of water, it can be added with just a click and the counter will update.

On the same page as the macros and water intake, the user can access the nutrition chat. By clicking on it, the user will be redirected to the Telegram chat, where the user can ask whatever he/she wants about nutrition, for example, what the user could eat during the day, if the user has two options of food, which one is better, etc. These answers are personalized, that is, the chat model collects the information about the user and returns an answer depending on the information about the user. These answers are personalized, i.e. the chat model collects the information about the user and returns an answer depending on the information about the food eaten in the last seven days, the personal information, etc.

On the weight page, the user can add a weight goal by entering some information and also check how the goal is progressing with the user's weight, as well as weight goal graphs.

If the user has any dietary restrictions, they can enter them in the Restrictions section. With this information, Nutrition Chat will not recommend dishes that contain these restricted ingredients.

The user can also upload photos of meals. The image recognition model will indicate what the food is and provide information about the macros. After the model returns the information, the user can indicate when the meal was eaten, i.e. whether it was for breakfast, lunch or dinner, and how many calories the meal contained.

The previous paragraphs have explained what the user can do in the application. But they are not the only ones who can access it. Administrators can also access it, but they cannot do the same things as a user. In the case of the administrator, they can add new campaigns. These campaigns have ingredients that are added after the campaign is created. You can add as many ingredients as you like. Each time the administrator adds a new ingredient, the data is saved in the database.

5. Planification

This section explains how the project was managed. Adequate planning is important when carrying out a project, and even more so when it is a team project. This is why the Kanban method was implemented.

Kanban is a lean method to manage and improve work among the project's participants. The approach aims to manage the work by balancing demands with available capacity and improving bottlenecks that might appear in the process. In software development, Kanban aids in the decision making about what and when to create new functionalities.

Firstly, the roles for each team member were allocated and the necessary tasks were arranged. To do this, a backlog was created with a description of the task, as seen in **¡Error! No se encuentra el origen de la referencia.**, a rough estimate in hours, an assignee, a priority (high, medium, low) and the progress of the task. Once all the tasks had been defined, they were placed on the Gantt chart to show when and how much time each would take. To fill this Gantt chart, the priorities of the tasks were taken into account, giving priority to the most important tasks and leaving the less functional tasks for later.

Title	Description	Status	# Estimate	Priority	Assignee
Chat and calories LLM	Creating a chat and calculating calories with the LLM	Done	167	High	Luken, Lucas
Food recognition	Create a food recognition ai	Done	150	High	Julen, Lucas
Backend	Create the backend of the web application	Done	200	High	Anje, Marina, Eki, Luke...
Prototype with node-red	Make the prototype of the web application with...	Done	120	Medium	Luken, Anje, Marina
Automatic code analysis	Create automatic code analysis	Done	50	Medium	Lucas, Eki, Julen
Flask API (AI server)	Create Flask API	Done	145	High	Marina
Data-Base	Create the database to save the information	Done	48	High	All of us
Server Access simulation	Make a simulation about how will work the server...	Done	11	Medium	Eki
Image recognition simulation	Make a simulation to know the the information will...	Done	40	Medium	Anje, Marina
Report	Write the report	Done	30	Medium	All of us
Video	Think the video, record and edit	Done	20	Low	All of us
Poster	Make the poster of the product	Done	5	Low	All of us

1. Figure: Product backlog

Once the Gantt was done a kanban board was created to visualize the process by which each task would be completed. The kanban board was divided in the following phases, each with their work in progress limits:

- **Analysis:** Task that are being analyzed, their dependencies, code examples, work in progress limit of five.
- **Development:** Tasks currently under development, work in progress limit of five.
- **Testing:** Tasks or functionalities under code testing, checking if they function as expected. Work in progress limit of seven.
- **Deployment:** Tasks waiting to be implemented in the main project, work in progress limit of ten.

Tasks are represented as cards in the kanban board, making them easier to manage and move through phases. Kanban cards contain the following elements: an ID, a description of the task, an assignee, hours estimate, beginning and end dates, and the lead time of that task. In order to see an example of the kanban board used in the project, go to the following annex **Kanban board**.

Miro (Khusid & Shardin, n.d.) was used to create the backlog as well as the Gantt chart and the kanban board. The following picture shows what the Gantt diagram looked like at the start of the project:

2. Figure: Initial Gantt

After the initial planification, the work was distributed, taking into account the tasks and the times expected for when they would be needed. Finally, the team started working on it.

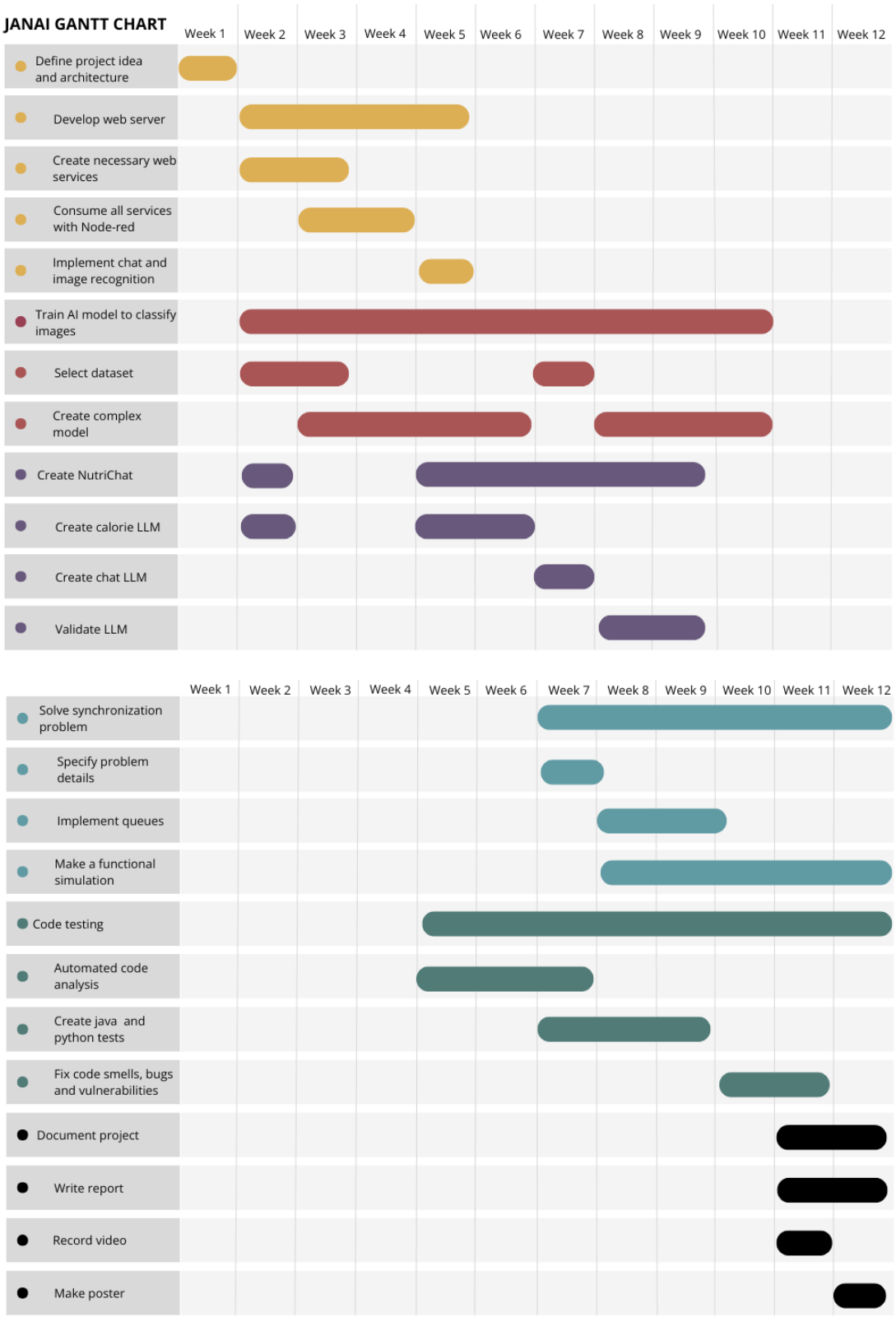
In order to know how we were dealing with each one and how we were planning and achieving the tasks, we made different meetings during the project: daily meetings, replenishment meetings and service delivery meetings.

But to ensure that the team followed a proper workflow and to establish some metrics, the team established Kanban policies at the beginning of the project. These policies defined how to communicate in the team, when to consider a task as done, the WIP(Work in progress limit) of each project phase, when to pass a task to another phase, etc. For more in detail information about these policies, go to the annex **Kanban policies**Kanban policies.

One policy and another very important part of the kanban methodology are it's various meetings. Each of these meetings have a different purpose and frequency, and the team decided to do the following ones:

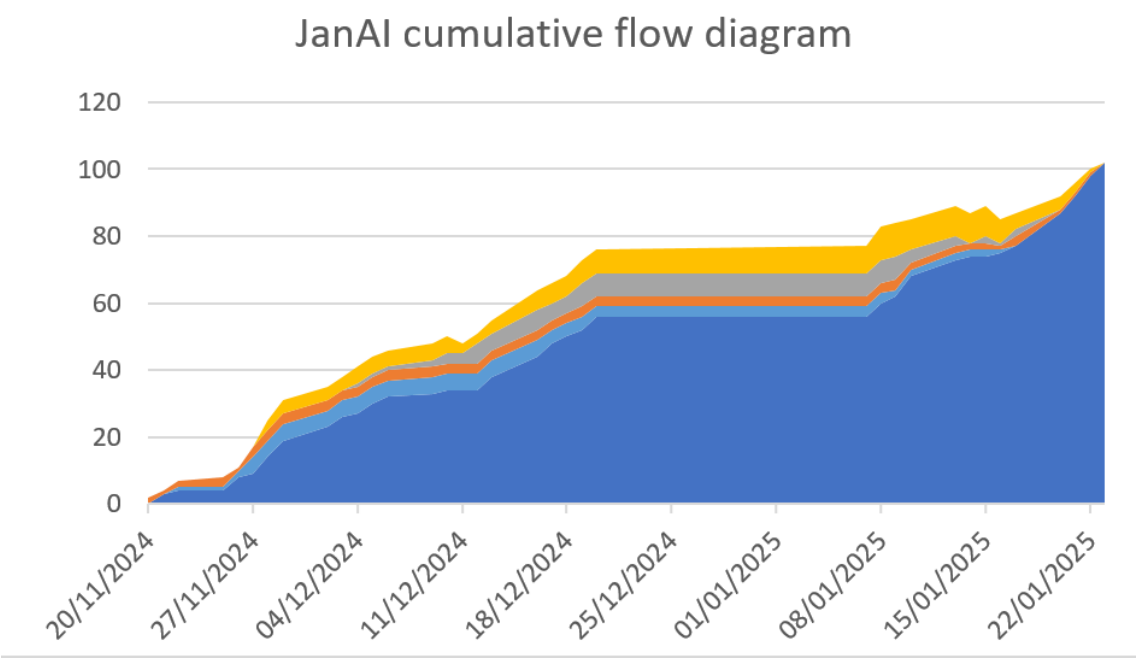
- **Daily meeting:** These meetings, as the name suggests, are executed everyday, and in them, each team member updates the rest of the team with the work done in the previous day, the work to do on that day, and if there's any problems or roadblocks communicating them to the team.
- **Replenishment meeting:** Replenishment meetings were done weekly, and in them the team decides which tasks of the kanban board to focus on that week.
- **Service delivery meeting:** Service delivery meetings were done at the end of each week. In them, the team checks the progress of the tasks proposed in that weeks replenishment meeting and if the flow of the tasks isn't appropriate, the team must figure out on how to improve.
- **One on One meetings:** These meetings were also done weekly, but instead of having only the team taking part in them, the tutor also took part in the meetings. The objective of these meetings is to update the tutor with the progress of the project and to gather feedback from him to improve the project.
- **Delivery planning meeting:** This meeting was done at the beginning of the project, in order to make an estimate on when the different features and tasks of the project would be delivered.

In the progress of this project, while the tasks were being finished, another Gantt was filled in order to check the actual progress of the project. This way, the team compared both gantt diagrams and discussed how the initial planning matched the real work.



3. Figure: Final Gantt

Last but not least, taking the kanban boards cards progress into account a cumulative flow diagram was created. The cumulative flow diagram represents the amount of tasks in each phase every day, being the lowest layer, the last phase known as Done. In the picture below, the project's cumulative flow diagram can be seen.



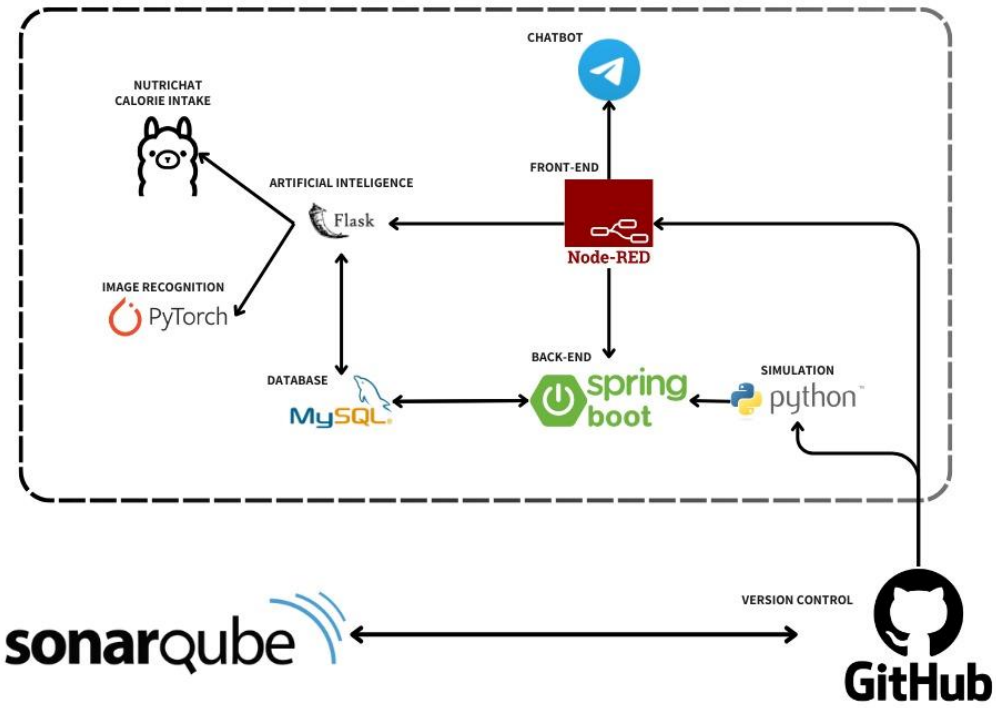
4. Figure: Cumulative Flow diagram

Thanks to the cumulative flow diagram, the progress of the tasks can be observed and conclusions can be made more easily. Some of the conclusions are that at the beginning of the project, the band of tasks starts widening, and it doesn't stop widening until mid December. The widening of the band means that more tasks are being created than the ones being completed, it is also known as a bottleneck. A few days later the bottleneck disappears for a few days and the band narrows, meaning that the throughput is higher than the entry rate. Later on, the band stays pretty unchanged until the end of Christmas holidays, where another bottleneck occurs. The team had to increase the amount of work done, sometimes exceeding the established wip limits until the final delivery of the project.

After the analysis, it's clear that the team exceeded themselves in a few occasions, creating to many tasks and in consequence, needing to work extra hours and burning themselves in the process. The overall management and task creation must be improved in future projects, defining the phases better and increasing the WIP of each task to more realistic measures. The holidays were also a period of full paralysis of the project, as seen in the diagram, working in those days needs to be taken in consideration.

6. System development

In this section, the technical parts of the project will be described as well as their respective images.



5. Figure: System architecture

6.1. Frontend and backend description

In this part, the frontend and the backend of the web application are going to be explained.

6.1.1. Frontend

Before starting with the creation of the frontend, we made page sketches. When we decided the content of each page, we started creating them using the dashboard in node-red. For that, we used different nodes, such as, form, dropdown, button and ui_control.

We created different flows depending on the page that will appear in the web application. These flows are going to be described in the following lines:

6.1.1.1. Flows (dashboards)

The flows are created with different nodes and among them, the request node to communicate with the backend part. Those flows are the following ones:

6.1.1.1.1. Create user

To create a user, we have different flows. Those flows are the following ones:

- **Initialize user:** here we have a group of little flows, the user selects its own information using the given dropdowns and this way, the information is shared into the flow variables. After selecting the best options, using the create user button, the dashboard changes to the next one.
- **Personal information:** the first flow is a form where the user has to introduce her or his own personal information. After submitting it, it sets the headers and it sends a request to receive a verification code. After that, the code is shared in the flow variable and it goes to the next dashboard, the one that verifies the code.
- **Verify code:** It has a form to enter the code that the user has received. Using the schema validator, it validates the user information and if it is correct, it makes a request to verify the code and to create the user.

If the user is created correctly, the dashboard changes to the login one.

6.1.1.1.2. Log in

The users and the administrator could use the functionalities of the web application, they will have to log in it. For that, we have the login flow that is the one that creates the respective dashboard. To start with the flow, we have a form node that is used to introduce the username and the password.

After that, it sends a request from the request node to do a post to verify if the user or the administrator is created. It will return the token that will be shared into the flow variable to be used in the rest of the request nodes. Depending on the answer of the request, whether it has a role like "USER" or "ADMIN", the flow will go its respective way. In both ways, there is another request to take the user or administrator´s information and after that, it is shared into the flow variable and it is set to the next respective flow. In the user´s case, a timeout will also be set to log out when the time is over.

6.1.1.1.3. Restrictions

To add restrictions, there are different flows that are necessary. These flows are going to be explained in the same order that the user will see them on the dashboards.

- Restriction information:
  - **Get classes:** after the user has logged in, this flow starts running. First of all, it gets the information of the user using the request node. After that, it makes another request to get the food classes that the user has available. If there are not more food classes to add in the list of the restrictions, it will show a notification to send the user back to the main page. Otherwise, the information is saved until the user has access to the restriction tag.
  - **Class selection:** in the class dropdown, the options will be according to the list received from the previous request. After selecting the class, it has two paths. One is to save the classID in the flow variable. For that, it uses the request node to get information about it. And the other path is to get the available types that the user has in the respective food classes by requesting to the backend.
  - **Type selection:** in this dropdown, there will appear the options got in the previous request and there will also be a special one called "All types".
    If an option is selected, it will get the information of the type by requesting with the type name to save the typeID in the flow variable. It also gets the available groups of the respective type by requesting. However, if the "All types" is selected, it will have to select all the group options to the next flow´s dropdown.
  - **Group selection:** This flow will do the same action the type selection flow does but in this case, instead of "All types" option, it will be "All group" option.
  - **Ingredient selection:** In this flow, there are two paths. One path will go if an ingredient is selected, then it will be requested and the return object will be saved. And the other path will be the "all ingredients" selection where it will be saved into an ingredient object.

- **Add restriction:** This flow starts with a button “Add restriction”, when the user clicks on it, using a function node, it will get all the information the user previously selected in restrictions flows. After that, it will do a post request to create the restriction. If the restriction is created correctly or not, a notification will appear.
- **Finish:** This flow is made by a button and a “ui_control”. By clicking the finish button, it redirects to the main page.

6.1.1.1.4. Main page

Before the user gains access to this tab, the system initializes water intake data by running a specific flow. In this flow, essential information such as the userID and username is retrieved from the flow variables. After setting up the necessary headers, the flow branches into four distinct paths:

1. Daily Calorie Intake Retrieval

A GET request is made to fetch the user's final daily calorie intake based on their username. Once the information is retrieved, various function nodes process the data to calculate specific metrics. These calculated values are then displayed on individual gauge nodes for clear visualization.

2. Macro List Retrieval

Another GET request retrieves the user's list of macronutrients. The retrieved list is processed and sent to the same set of gauge nodes, ensuring consistency in the user interface.

3. Water Intake Recommendation

The system sends a request to retrieve the recommended number of water glasses the user should drink. This information is displayed in a text node, providing users with actionable hydration guidance.

4. Current Water Intake

A request is made to fetch the current amount of water the user has consumed. The retrieved value is displayed on a dedicated gauge node, enabling the user to track their progress throughout the day.

Additionally, there is another flow triggered by a button node labeled "One More Glass of Water." When the user clicks this button, the system adds one glass of water to their total for the day. The flow performs the following steps:

- Retrieves the current water intake from the backend.
- Increments the amount by one.
- Updates the new value in the database.

This interactive feature allows users to actively manage their hydration in real-time.

Furthermore, the main page includes a button that directly redirects users to the Telegram chatbot. This integration seamlessly bridges the Telegram bot and the web page, offering users a unified experience. Through the Telegram bot, users can interact with the system in a conversational format, complementing the web-based features.

6.1.1.1.5. Add campaign

This flow belongs to a dashboard where the administrator will introduce information about a campaign. After filling in the form, it will request to add the campaign. If it is created correctly, it will redirect to the campaignIngredient dashboard. If not done correctly, it will send the administrator an error message.

6.1.1.1.6. Ingredient in Campaign

Two flows complete this dashboard:

- The first flow is related to the dashboard of the same name. In this flow, there is a form to introduce information about the ingredients that are in the campaign. After submitting the information, it requests to get information about the ingredient using the name of it. After that, it requests to add the object to the database.
- The second flow starts with a button node. If the user has not any other ingredient to add, he or she will push in this button to be redirected to another dashboard.

6.1.1.1.7. Weight goal

This dashboard allows users to either set a new weight goal or view the graphical representation of their weight goals. The functionality is divided into two flows, as explained below:

- Create weight goal:

  Users can create a new weight goal by entering the necessary information, which is then linked to their profile. This data is added to the appropriate database table by sending a request to the backend with the weightGoal object.

- Show weight goals:

  This flow is triggered when a user logs into the web application. The user's information is retrieved from the flow variable, and a request is sent to the backend to fetch the list of weight goals. The weight information is displayed in one graph, while the goal weight is shown in another graph.

  If the user adds a new weight goal, the flow runs again, but it skips fetching the user's information as it is already stored from the previous session.

- Additional Functionality

  Whenever a user creates or adds a new weight goal, a request is sent to the Flask API. The API processes the user's data along with various parameters (detailed in the **Trigger**: and calculates the user's daily calorie intake. This value is then sent to the main page, where it is used to update the maximum limit on the daily calorie intake graph.

6.1.1.1.8. Integration with Telegram

The Telegram nodes in Node-RED (O'Leary & Conway-Jones, 2013) include a Receiver Node configured with the token of the Telegram bot. This node captures all messages sent by users to the bot. The username from Telegram is saved to match and retrieve corresponding information from the MySQL (MySQL AB, 1995) database, but it's essential that the Telegram username matches the username registered on our platform to ensure proper data association. Additionally, the chat ID is stored to identify where to send the response.

A request is then sent to our Flask API (Ronacher, 2010), which invokes the LLM (Large Language Model) to process the query. The response from the LLM is returned to the user via Telegram.

6.1.1.1.9. Integration with Image Recognition Model

The integration consists of two distinct pages:

1. Image Upload Page:

On this page, users can select and upload an image. The image is converted to a Base64 string, resized, and displayed on the Image Result Page so users can verify the uploaded image. A request is then sent to the image recognition model, which analyzes the image and returns the food item with the highest confidence level, along with an array of other identified foods and their confidence levels. After processing, the user is redirected to the Image Result Page.

2. Image Result Page:

This page displays the uploaded image and the food item identified by the model with the highest confidence. Next, a request is sent to the back-end to retrieve the nutritional macros (e.g., calories, proteins, carbs, fats) for the identified food, which are visualized through various graphs on the page.

Additionally, a dropdown menu is provided for users to select the meal (e.g., breakfast, lunch, dinner) in which they consumed the food. Users also need to input the amount (in grams) of the food they ate. After clicking the "Submit" button, the system redirects to the Main Page, where the day's graphs are updated with the nutritional information of the consumed food.

6.1.2. Backend

While we were working with the frontend, we were completing the backend too. This part is used to control the requests that the user made from the frontend and to get and manipulate the information of the database.

To create the backend, we first created the database, where we built the necessary tables to store the information we wanted. These tables are related to each other, because many of them need information from other tables. These tables are used in the backend to create the models. These models are composed of the same variables as in the tables, with their respective names, variable types and joints between models. In addition, they are made up of the respective setters and getters of each variable. The models we have created are the following ones:

- Administrator
- Campaign
- Food
- FoodClass
- FoodGroup
- FoodList
- FoodType
- hasIngredients
- Ingredients
- IngredientsInCampaign
- Restrictions
- UserData
- WeightGoals

Each model has a repository. This is used to query the database and obtain the required information.

Once we had created the models and the joints between them, we built the necessary controllers so that all the recessive requests could be made from the frontend. Some of these are more important than others. These are the following:

- **CampaignController:** This controller has three different get requests. The first one is to get all the campaigns saved in the database. The second one is to collect the information from a single campaign, specifically the one with the *Id* we want. The third one is for the same purpose as the second, but in this case the campName is used instead of the campaign *Id* field. In addition, this controller has a post request function, it checks if there is any campaign with the given name and whether the administrator exists. If it matches, the campaign is stored.

- **EmailVerificationController:** This controller has a single post request. To understand this function, it is necessary to note that when a user creates an account in the application, the backend sends an email to the user with a verification code that he/she must enter in the application. For this we have these two classes:

- EmailService: This class consists of two functions, one is to send a simple message and the other is to send an html message to the email address specified by the user when entering their details.
- EmailVerifiacionService: This class consists of a variable that is set to a random integer value in a generateVerificationCode function. There is another function that returns the html of the email that the user will receive. Lastly, another function sends the email using the function located in the emailService class.

After this explanation, the post function in this controller compares the code sent to the user and the one the user introduces in the frontend. If the codes are equals, the user will be created.

- **IngredientsInCampaignController:** In this controller we created a get function and a post function. On the one hand, the get function returns the list of all ingredients that are in campaigns. On the other hand, the post function adds ingredients to a particular campaign.

  To do this, the post function first checks the campaign and the ingredient in the database and then, it checks if the ingredient is not already added in the campaign. After checking everything, we store the information in the database.

- **RestrictionsController:** In this controller we have several get functions and one post function.

  As in the other controllers, there is a get function to get a list of all the constraints.

  On the other hand, we have a function that returns the list of classes the user has available in the database, the list of classes that are not added to the restrictions table (available in **Backend annexes**). To do this, it gets a list of all the classes included in the food class table and another list of the classes included in the restrictions table. Then, it calls the filterInvalidFoodClasses function, where it goes over the list of all the classes the user has added to the restrictions table. It checks whether or not the other list, the list where the user has added to the constraints, contains that class. If it matches, it collects two more lists: one is the list of all the types the class has and the other is the list of all the types the user has entered for that class.

  When it has collected the two lists, it calls the following function: shouldRemoveFoodClass. This function checks if all the types are included in the list of constraints, and if they are not, it checks if one of the types included in the constraints is the one that refers to all the types. If it fulfills either of these conditions, the function returns true, which removes that class from the list of all classes in the class table.

  When it has finished looking at all the classes, it returns the list of classes available to the user.

There are other functions in the controller for the same purpose, but with the appropriate information to be checked. In the case of the get Ingredients request function, only one function is used because it does not have any other table related to it and it is related to the restrictions table.

The last function is the post request where the new constraint is added. To do this, it first checks whether the user, restriction, class, type, group and ingredient exist in the database, and if they do, the constraint is added to the database.

- **UserDataController:** In this controller we have get, post, put and delete functions. The get functions: get an administrator's information, get a list of all users, get a specific user by their userID and get a specific user by their username. The post function is used to add the user to the database, the put function is used to update the user's information and the delete function is used to remove the user from the database.

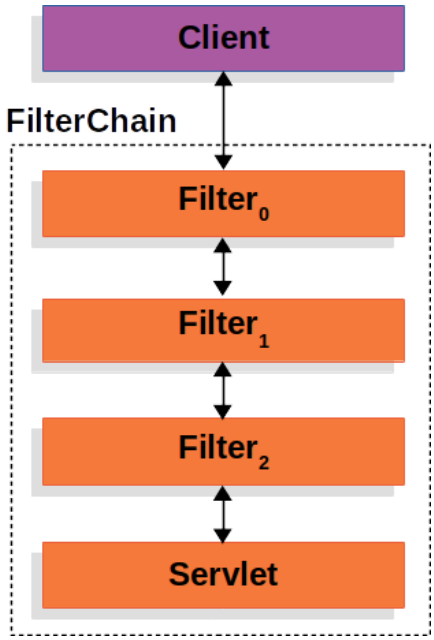
- **WeightController:** Here we have a get function to get a list of all the weight goals a particular user has and a post function to add a new weight to the user.

In all the models, we have implemented the validation. We put in the models what each variable should be. Then in the controllers, we added the validation in the post request function. We did this to verify if the information that would be included in the database was correct or not. To do that, we indicated which variables we wanted to be validated and we also added an error variable. If the variables have the same error in the structure or in the variable type, the error variable will indicate which error it is. Inside the functions, they first check if there is an error and if there is, it will return which error it is and it will not continue with the rest of the function.

6.1.2.1. Security

As an additional measure of security for the backend project, JWT (Auth0, 2015) authentication was added to the workflow using the Spring security framework. Each endpoint in the servlet is protected by a security filter chain, a data structure provided by the framework that supports the layering of filters to provide an exploit-resistant basis for communication between the server and client agents.

Below is an illustration of the structure of a standard security filter chain, as described in the Spring Boot documentation:



6. Figure: Spring Boot security filter chain

FilterChain structure according to Spring Security documentation

The filter chain created for the project, given its scale, is designed to establish the basic session management system with JWT (Auth0, 2015), so the following configurations were applied:

- **CSRF Protection Disabled:** CSRF protection is disabled, given that JWTs are used in the Authorization header, mitigating CSRF risks by design.
- **Stateless Session Management:** The session management policy is set to `SessionCreationPolicy.STATELESS`, meaning that no session state is stored on the server. Instead, each request is authenticated independently using the JWT provided by the client.
- **Public Endpoints:** Specific endpoints, such as `/login`, `/register`, and `/user/add`, are excluded from authentication filtering. All other requests require a valid JWT token.

This behaviour of filtering out any unauthorized requests relies on two key components, a custom JWT Authentication filter to validate whether the JWT token provided is valid and authenticated, and a utility JWT class for generating, validating and parsing information sent in the headers.

With this established, the custom filter developed is a custom implementation that extends `OncePerRequestFilter`, and performs validation in the following way:

- **Authorization Header Parsing**: Extracts the `Authorization` header and checks if it contains a Bearer token.
- **Token Decoding:**
  - Extracts the username and roles from the token using `JwtUtil`.
  - Loads user details from the database through `MyUserDetailsService`.

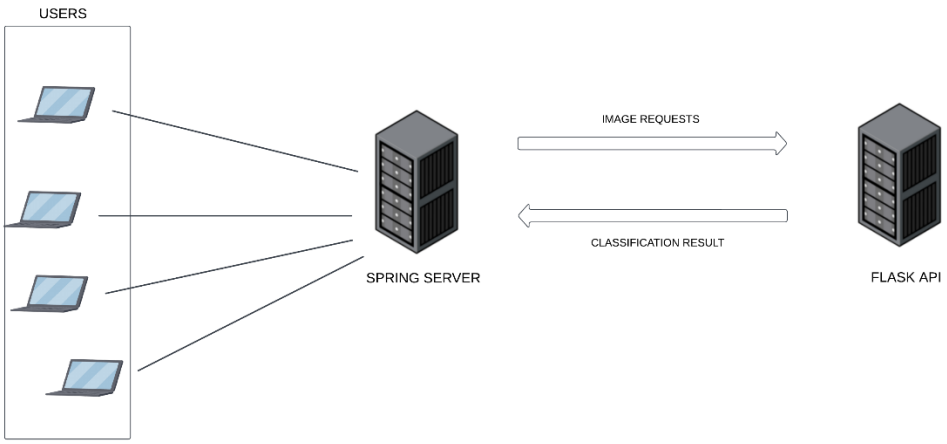
- **Authentication**:
    - If the token is valid, a `UsernamePasswordAuthenticationToken` is created and set in the `SecurityContextHolder`. This token contains the user's details and roles, granting access to authorized resources.
    - If the token is invalid, the request is filtered out, and an error is logged.

To support the functionality of decoding and encoding these JWT systems, the JWT Utils class encapsulates all parsing logic for use in the application, it is important to mention that it retrieves its configurations (e.g., secret key and expiration time) from the application properties file, for easy maintainability.

- **Token Generation**: Generates JWT tokens with the following information:
    - **Claims**: Contains user-specific data, such as roles, which are added to the token during creation.
    - **Subject**: Stores the username.
    - **Expiration Time**: Tokens expire after a configurable time, retrieved from the application properties file.
    - **Signing Key**: Tokens are signed using a secret key to ensure authenticity.
- **Claims Extraction**: Methods such as `extractUsername`, `extractRoles`, and `extractExpiration` allow the application to parse tokens and retrieve necessary information.
- **Token Validation**: Verifies the validity of tokens by comparing the provided username with the one encoded in the token and checking for expiration.
- **Token Expiration Check**: Ensures that expired tokens are rejected by extracting and validating their expiration times.

6.2. Thread simulation

Before diving into the developed simulation, the system architecture for this part must be taken into consideration, being visible on 7. Figure:



7. Figure: System architecture

The system architecture involves two main servers, as seen in 7. Figure: the Spring server on the left and the Flask API server on the right. The Spring server is the core of the application, hosting the Java code that manages controllers, data classes, security features, and other essential components. Meanwhile, the Flask API server houses the Python-based image recognition model.

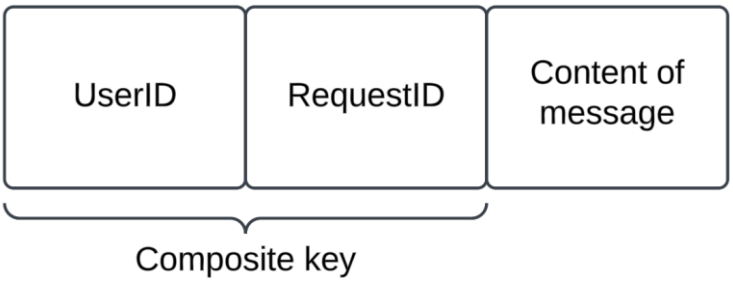
Since JanAI grants the option to load images, the process begins when users upload a picture of a dish through the website. The image is first sent to the Spring server, which then forwards the request to the Flask API for classification. This classification enables JanAI to provide information based on the uploaded image.

To prevent overloading the AI model, which has limited resources, the system imposes a restriction on the number of requests processed simultaneously. This ensures the model operates effectively without failure, while still fulfilling all user requests. Once the model completes the classification, the results are sent back to the respective users. This is made possible by incorporating user and request id-s in the message, ensuring responses are correctly matched to each request.

The message structure includes three key fields: userID, requestID, and content. In a practical scenario, the content would represent the uploaded picture, as seen in 8. Figure.

Users are created through a for loop, with the userID assigned based on the iteration of the loop. Each user can generate a random number of requests, ranging from 1 to 5, which are also created using a for loop. The requestID for each request is similarly assigned according to the loop's iteration.

This approach ensures that every message has a distinct combination of userID and requestID, making each message unique. This uniqueness guarantees that responses are accurately matched to the correct user and request. For example, if User 1 uploads two pictures—one of a banana and another of fried rice—each message will have a unique composite identifier. This ensures that the response for the banana is correctly matched to it, and the same applies to the fried rice.

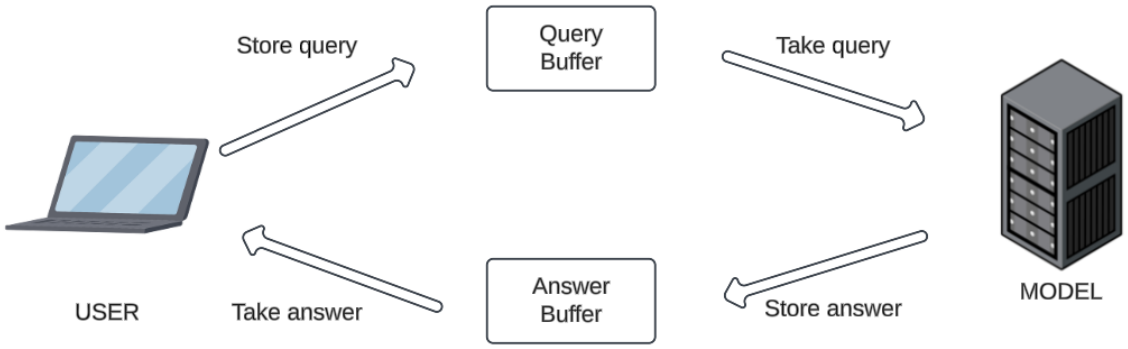


8. Figure: Message structure

To summarize, the solution of the problem belongs in the Flask server, the main conflict being the excessive amount of incoming requests and the unavailability of giving an answer to all of them simultaneously.

6.2.1. Solution approach

To control the incoming number of requests in the Flask server, two buffers have been implemented, one for the queries (or requests) and another to store the answers, as seen in 9. Figure.



9. Figure: Thread problem structure

For this problem two main actors have been created. The users and the model. The users create requests that are taken as threads, and the model creates one answer thread to service the requests.

The program flows as the following: first the user puts a request in the query buffer, then the model takes it, processes it and stores the answer in the answer buffer for the user to take it.

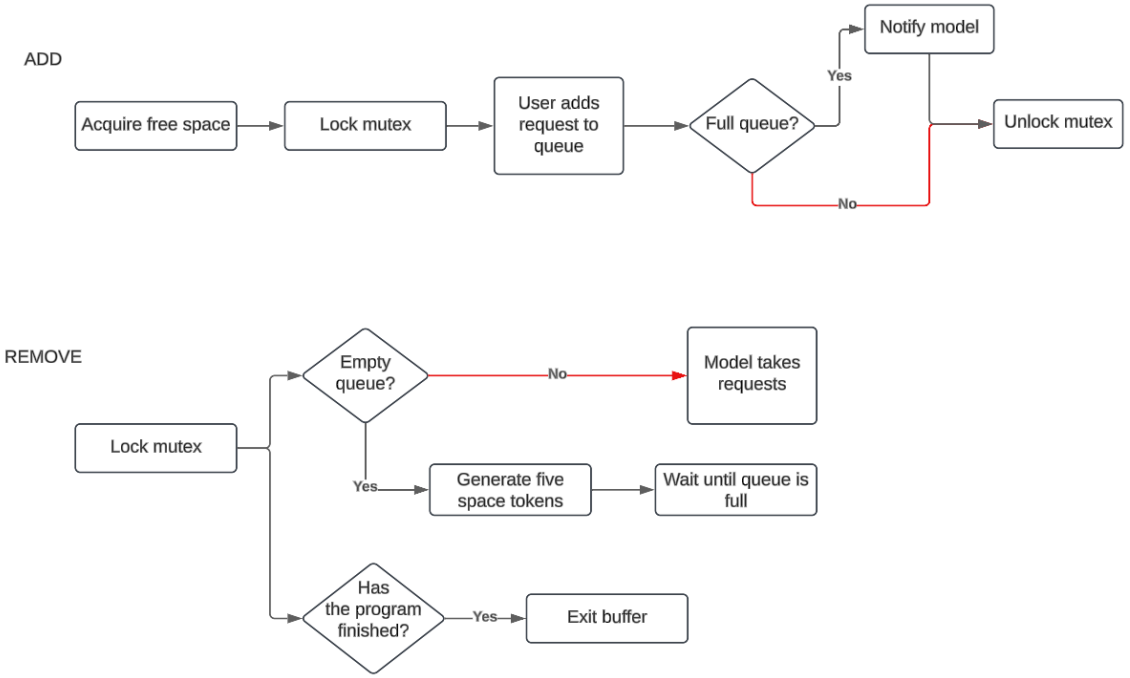
6.2.2. Query buffer

Since the model can work with only one request at a time, a blocking queue has been implemented in the query buffer, with a maximum size of 5 requests (for the simulation). This is a blocking queue, which works as a FIFO queue and is thread-safe. Besides that, mutual exclusion between threads must be ensured so no information is changed at the same time by multiple threads, so a lock has been implemented from Python's threading package, working as a mutex semaphore but with added simplicity, like not having to manually release the lock if a thread is waiting in a condition.

In terms of how the query buffer works, it can be seen in 10. Figure. The free spaces of the queue are managed with a semaphore with five available tokens (size of the queue), and once a request enters the buffer, acquires a token of the "spaces" semaphore from the "add" function of the buffer. If there are no tokens available, the thread waits until tokens are generated in the "remove" function before locking the mutex and adding the request to the queue. Once the queue is full the model will be notified and the mutex will be unlocked regardless.

In the "remove" function, also the mutex is locked before checking if the program has ended or the queue is empty. If the queue is empty five tokens will be generated for the "spaces" semaphore and the model will wait in the condition until the queue is full. If the queue is not empty, the model takes the requests until there are none left.

It should be noted that the users generate a random number of requests each, and the model waits until the queue is full before retrieving the requests, so the total number of requests generated are not divisible by five, a deadlock would occur because there would never be enough requests to fill the queue in the end. To fix this, the threading condition has a timeout, so if ten seconds the model is not notified it will take the remaining elements in the queue and process them.



10. Figure: Query buffer flowchart

6.2.3. Answer buffer

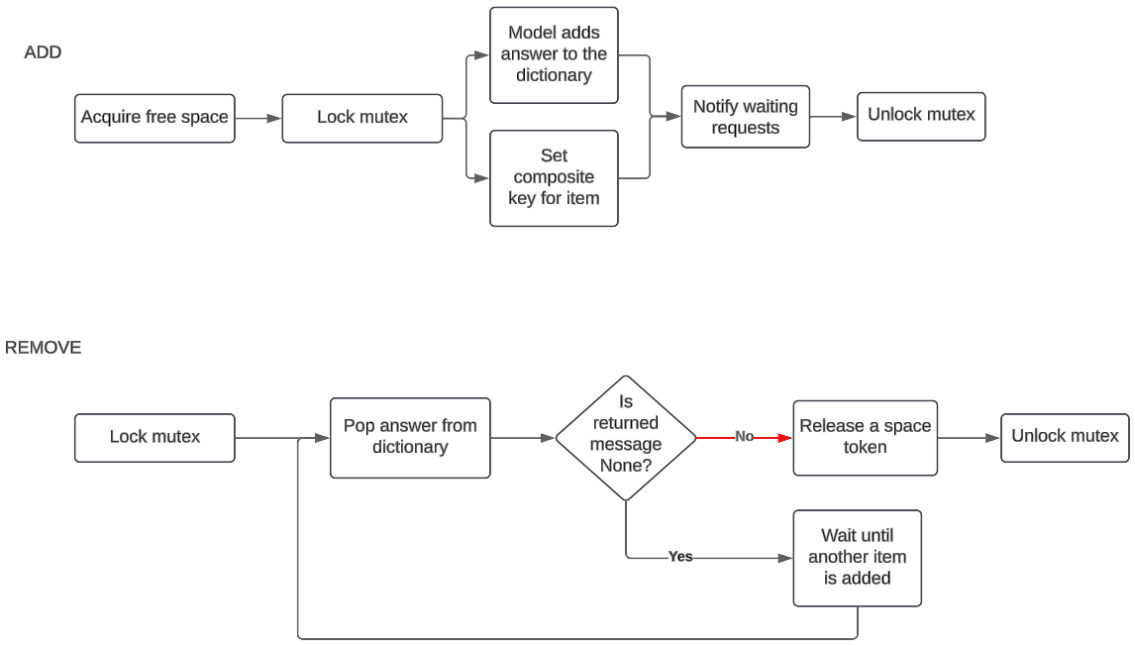
The functionality of the answer buffer is quite similar to the query buffer, with the exclusion of the FIFO blocking queue, because there is no need for a user to wait for a queue to be filled up in order to take the answer to their queries, nor a FIFO strategy is needed.

Instead, a dictionary has been used in combination with a tuple as a composite key formed by the userID and requestID, significantly improving the searching process for the message to be retrieved. In the case of a list, each request would have to go through the list until it finds the indicated answer message, but by using a dictionary and composite keys, the searching process is way simpler, since the element that matches the key is retrieved and removed from the dictionary, and if there are no matches, a “None” is returned.

As seen in 11. Figure, once a request thread enters the “remove” function it enters in a “while” loop until it finds its answer. First it tries popping an answer from the dictionary. If no item is found with the same key, the request will wait in a condition, and the threads that are waiting for an answer will be notified when the model adds a new item before checking the dictionary again. If an item is returned, a token will be released for the “spaces” semaphore before unlocking the mutex, and else the process will be repeated.

Regarding the model, it acquires a token from the “spaces” semaphore when it enters the “add” function and locks the mutex. Then, it adds an item to the dictionary, also setting the composite key for the item to add and notifying all waiting requests before unlocking the mutex.

The code for both buffers can be seen in Thread simulation code parts for more information in regards to their functionality.



11. Figure: Answer buffer flowchart

6.2.4. Program ending

In order to ensure the program finishes adequately, another implementation has been done in the query buffer, a simple boolean which indicates the model there are no more requests to be fulfilled and it has to exit from its execution. In other words, if the model gets a "None" from the query buffer, it will exit from the "while" loop and it will end properly, ensuring all threads are terminated.

In python, if a thread has to be stopped, a flag has to be set in a threading event ("_stop_event") (*Python Threading*, n.d.), this way once the flag is set the threads are terminated gracefully. If when interrupting the thread it has not finished its execution, by joining them we can ensure that they first finish their job and then are terminated, to prevent any data loss or exceptions.

6.3. AI Models and API Service Description

This system integrates multiple AI models and APIs to deliver a comprehensive, user-centered experience in nutrition and dietary recommendations. The core architecture blends two language models (LLMs) and an image recognition model to provide personalized guidance. The LLMs are tailored to handle different aspects of interaction, while the image recognition model enhances functionality by processing visual data. Together, they enable efficient calorie estimation, dietary assessment, and responsive chatbot interactions, creating a cohesive and robust AI-driven solution.

6.3.1. Calorie LLM

Meta Llama3 (Meta, 2024) LLM is used to calculate the daily calorie intake a user should follow. After creating an account, the user adds their weight and weight goal. Once this information is submitted, it is stored in a MySQL database, and a database trigger is executed (see **Trigger:**). This trigger initiates various calculations, including Basal Metabolic Rate (BMR) and Total Daily Energy Expenditure (TDEE), using scientifically validated formulas.

From these calculations, three potential daily calorie intake values are generated. All user data stored in the MySQL (MySQL AB, 1995) database, along with the three calculated calorie values, is then sent to the LLM in a prompt. The LLM analyzes the data and selects the most appropriate calorie value based on the user's goal and information. For example, if one of the formulas considers the user's weight and the goal is to lose weight, the LLM would prioritize that value.

Since all three calculated values are scientifically accurate, any choice made by the LLM will also be correct. As a result, no additional validation is required, ensuring 100% accuracy in the LLM's decision-making process.

6.3.2. Recommendation LLM

In addition to using Meta Llama3 (Meta, 2024) for calorie recommendations, it is also utilized to create a chatbot using a Retrieval-Augmented Generation (RAG) approach. This chatbot is designed to be an expert in nutrition, capable of providing detailed and personalized food recommendations.

Core Components:

- Database Integration:

The chatbot queries a MySQL (MySQL AB, 1995) database for user-specific details such as height, weight, age, activity level, health goals, dietary restrictions, and recent meals. These details ensure personalized responses aligned with the user’s unique dietary needs.

- Document Processing:

PDF documents relevant to nutrition are converted into embeddings. This involves:

  - Reading PDF files and chunking them into smaller text segments.
  - Generating embeddings using Ollama's nomic-embed-text model.

    - Storing the embeddings in a Chroma vector database for efficient similarity-based searches.
- Vector Store and Retrieval:

Chroma's vector store retrieves relevant context by comparing the user query to the embedded document chunks. This context is passed into the chatbot for generating precise and evidence-based responses.

- Chatbot Interaction:

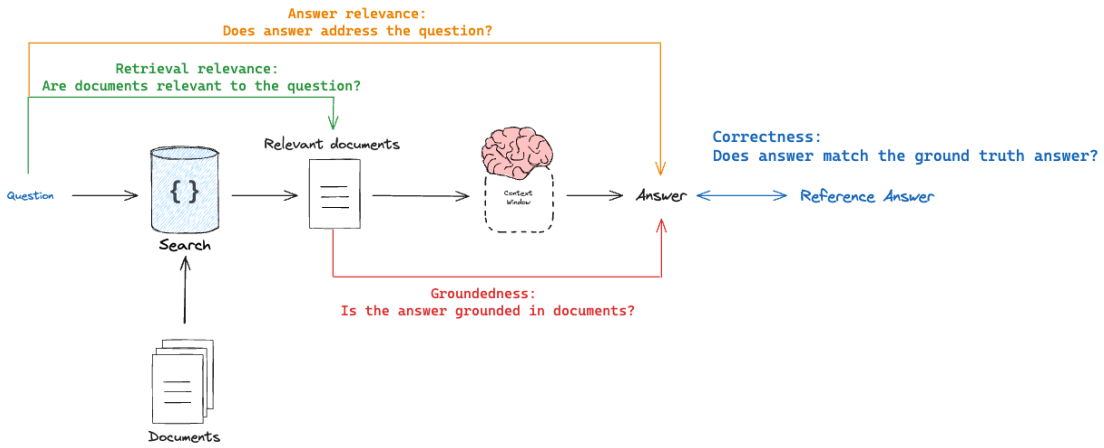
The chatbot uses the LangChain framework with the Ollama LLM (llama3:latest) to combine the user data and retrieved context into structured, actionable recommendations. A system prompt ensures the chatbot adheres to guidelines, such as avoiding restricted foods and aligning advice with user goals.

- Process Flow:
    - User data and recent meals are fetched from the database.
    - A retrieval chain is created, combining document embeddings and a chat model.
    - User queries are processed by retrieving relevant documents, forming a context.
    - The context, user data, and query are input into the chatbot, which generates a concise, helpful response.
    - This integration ensures Janai delivers scientifically grounded, personalized nutrition advice while considering user preferences and dietary restrictions.

12 Figure: Flow of the RAG used

6.3.2.1. LLM validation

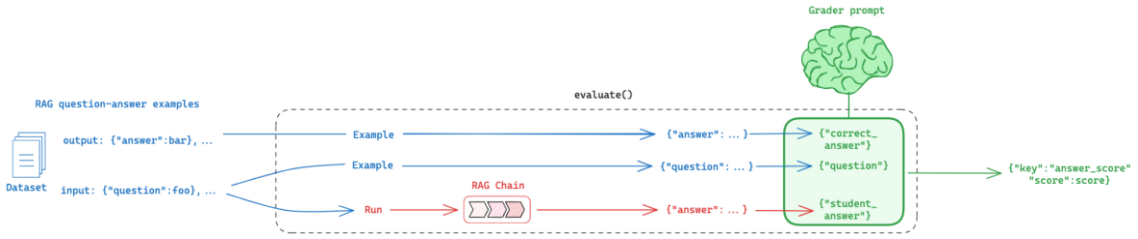
To validate this LLM-based Retrieval-Augmented Generation (RAG) system, three distinct metrics across ten different tests were made, accounting for variations in LLM responses. A self-made dataset was used to ensure consistency, and an LLM was leveraged as a judge for evaluation purposes. Here are the evaluation types and their outcomes:



13 Figure: Retrieval Augmented Generation Evaluation

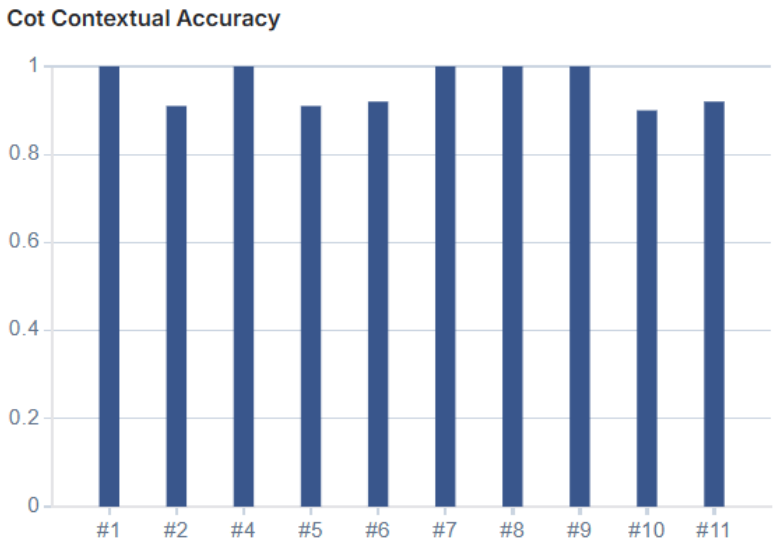
- **Reference Answer Evaluation**

Using an LLM-as-judge with a customizable grader prompt, the dataset inputs and reference outputs with the generated outputs were compared to assess Chain-of-Thought (CoT) contextual accuracy.



14 Figure: Reference answer evaluation metric

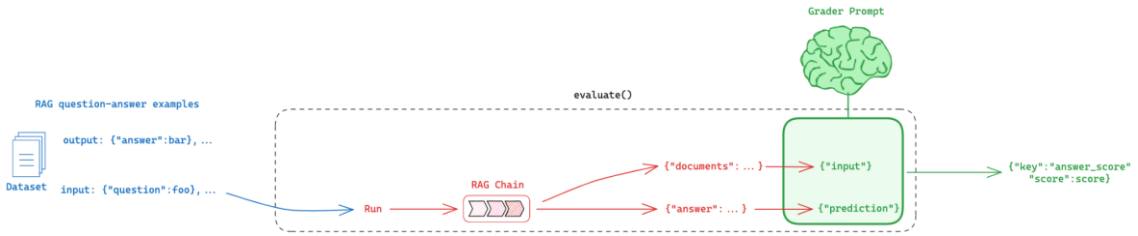
The CoT contextual accuracy, as shown in the results, is nearly 1/1. This indicates that the generated answers align closely with the expected context and reasoning.



15.Figure: Cot contextual accuracy results

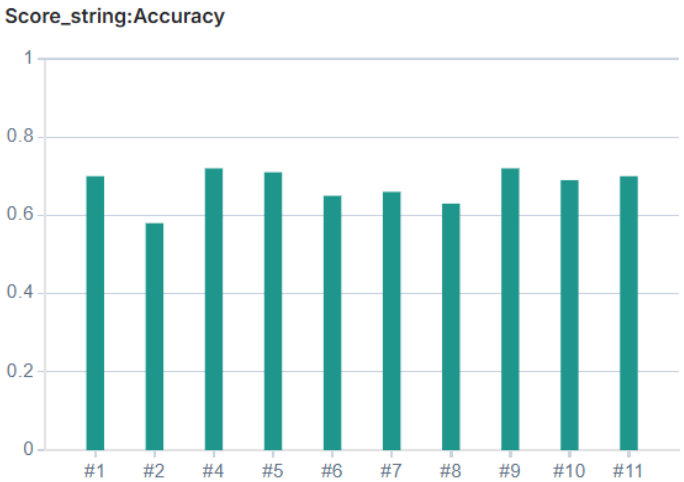
- **Answer Hallucination**

This evaluation also relied on an LLM-as-judge, customized to analyze the dataset inputs, retrieved documents, and generated answers. The score_string accuracy was calculated to determine how well the system avoids hallucinations.



16 Figure: Answer hallucination evaluation

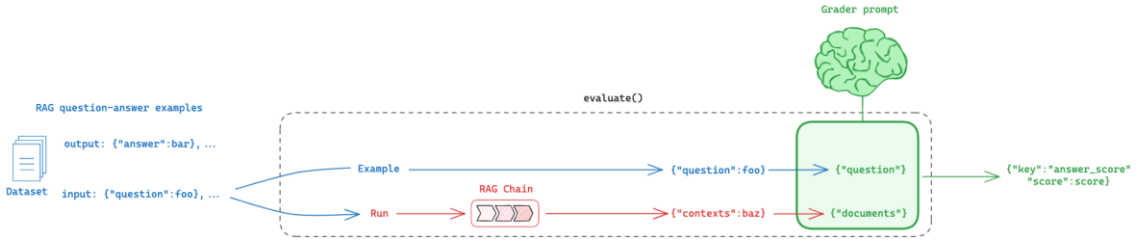
The results showed a score of approximately 0.7/1, indicating some hallucinations still occur. This suggests room for improvement in aligning the retrieved documents with the generated responses.



17 Figure: Score string accuracy results

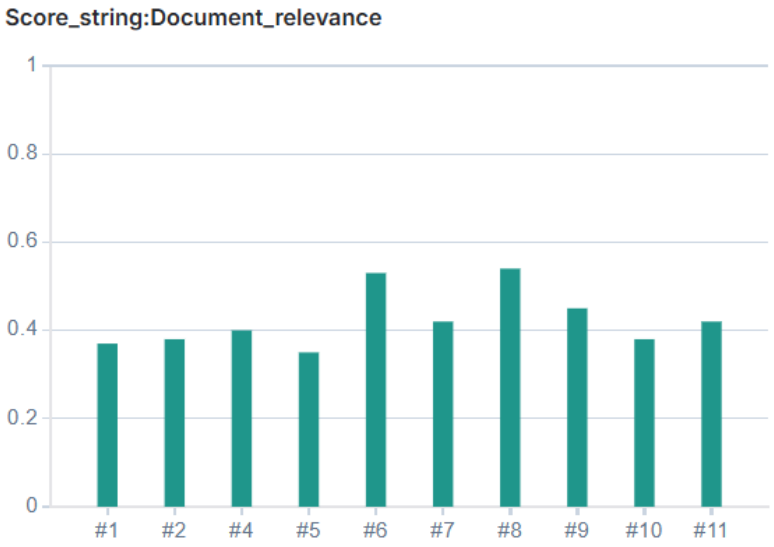
- **Document Relevance to Question**

For this metric, the LLM-as-judge assessed the relevance of retrieved documents to the dataset inputs and the user's question.



18 Figure: Document relevance to question evaluation

The score_string document relevance was found to be around 0.4/1, revealing that the retrieved documents are not fully aligned with the questions asked. This highlights the need to expand the document base in the vector database, as many user queries are not adequately addressed by the available PDFs.



19 Figure: Score string document relevance results

Conclusion:

While the CoT contextual accuracy demonstrates strong reasoning capabilities, improvements are needed in reducing hallucinations and increasing document relevance. Expanding the vector database with more targeted content will help address gaps and enhance the system's overall performance.

6.3.3. Image Recognition Model

To make the image recognition model, we used PyTorch (a Python library) (FAIR, 2016) to develop two Deep Neural Networks (DNN) that make the final model, and Pillow to preprocess the images used for training and validating the two models. These two DNNs are an autoencoder and a classifier that uses the autoencoder's encoder.

We have two versions of each, one "simpler" than the other. This happened as we made several iterations of the autoencoder with small changes until achieving a point where there was no more improvement. This first version got ~25% accuracy with the Food-101 dataset, a dataset with 101 classes and 1000 images of each class. As this wasn't enough, we did some research and learned about residual and skip connections, so we made a V2 version using them. This gave us big trouble, as our laptops became incapable of training it, so we had to find other ways for training the model. One of us knew a person with a somewhat powerful GPU that we could use under some conditions, such as having to provide a "plug and play" file and only using it overnight (since the owner doesn't have knowledge about programming or AI training), so we went ahead and used it for about a week, but as we had few time left and had trouble setting up the environment and training, we had to step down and go back to our laptops with a reduced dataset with few classes so it wouldn't take several days to train. With this reduced approach, we achieved ~60% accuracy on the second version of the models.

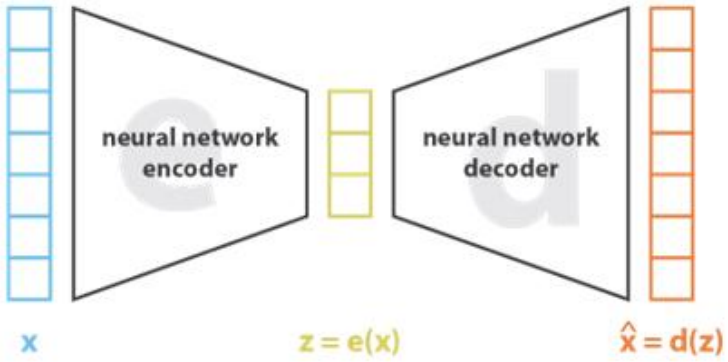
6.3.3.1. Preprocessing

Preprocessing all the images of the dataset (the images used for training and validating) is mandatory, as the original images may be too big or in a format that the models don't understand. This is resolved with the preprocessing functions, that take the original images, they get "fixed" and saved in a folder where all the images are valid for the models. This is done by firstly taking the class the image is from (so what dish the image is), then resizing the image and changing it to a valid format, and lastly saving it on the corresponding path. Both V1 and V2 use the same preprocessing, except for the image size, where V1 uses 224x224 and V2 uses 256x256 images.

Although, before this preprocessing, there is another *prepreprocessing* that fixes the structure of the original dataset, as the structure it comes with is not valid either for the preprocessing function or the models. In this function, we simply take the images and move them to where they should be, without any alterations to the actual images

6.3.3.2. Autoencoder

An autoencoder's purpose is to compress and decompress data, an image in our case. It works using an encoder and a decoder:



20. Figure: Autoencoder

Using the image as a reference, the input image would be 'X', the encoded image 'Z' and the reconstructed image 'X̂'.

The working of an autoencoder is quite simple: X is inputted to the encoder, which converts the image to a tensor (something like a big vector) and makes this vector smaller by encoding or compressing it. In this encoding process, data is lost, but the encoder "learns" what the most useful information of the image is and keeps it.

The region of the autoencoder with the smallest representation of the image is called Latent Space, a place where the tensor of the image is completely abstract and has the most important characteristics of the original image. This is the Z of the image (encoded X).

After X is encoded to Z, the tensor is inputted to the decoder, whose role is to reconstruct the latent representation of the image to another one as close to the original as possible, creating the image X̂ (decoded Z).

All of the explained above is valid both for V1 and V2 autoencoders, but the inner workings of them are quite different.

6.3.3.2.1. Autoencoder V1

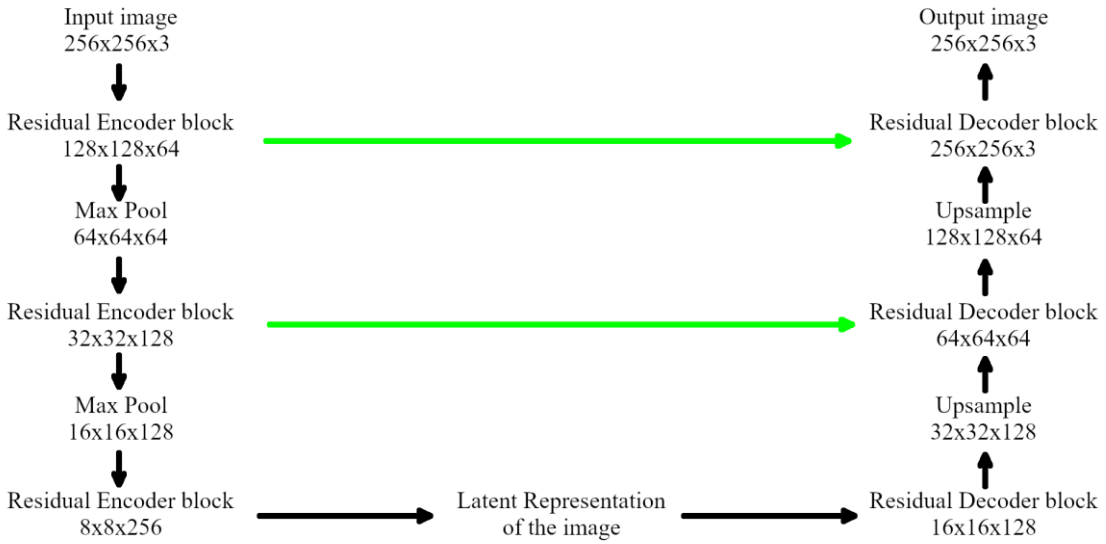
On the one hand, the inner workings of the encoder goes as follows: Convolution layer, Batch Normalization layer, Activation layer and Dropout layer. And on the other hand, the decoding process is: Convolution Transpose layer, Batch Normalization layer, Activation layer and Dropout layer.

Each of these layers have different purposes:

- Convolution layer: this is the layer that learns the most and does the actual compressing of the image. The size of the output tensor is smaller than the input's. The dimensionality changes depend on the parameters of the layer, as the formula shows (*PyTorch Formula*, 2023). In our case, we chose to halve the dimensions of the image on each convolution layer and double the layers of the image (except on the first conv layer that goes from 3 to 64 layers).
- Batch Normalization layer: as the name implies, it normalizes all the values from the images of the batch to be with a mean of 0 and a variance of 1. It also introduces learnable scaling and shifting parameters to adjust the normalized values, and normalized values work better with other layers (like activation layers).
- Activation layer: this layer can be of several types: ReLU, SoftMax, Sigmoid, etc. Both the encoder and decoder use ReLU. A ReLU function is a *max(0, x)* function, so only positive values will go through it, and negative values will be zeroed. This helps reduce the vanishing gradient problem, which is explained later.
- Dropout layer: this layer helps the model generalize more as it deactivates a percentage of random neurons of the tensors. This prevents the model from relying too much on a set of neurons, as they could get deactivated.
- Convolution Transpose layer: similar to the Convolution layer but reversed. Instead of compressing, it decompresses. The size of the output tensor also depends on the parameters of the function, as shown here. In our case, we chose to halve the layers of the image and double the dimensions (so the opposite to the normal Convolution layers).

6.3.3.2.2. Autoencoder V2

The second version of the autoencoder has the next structure:

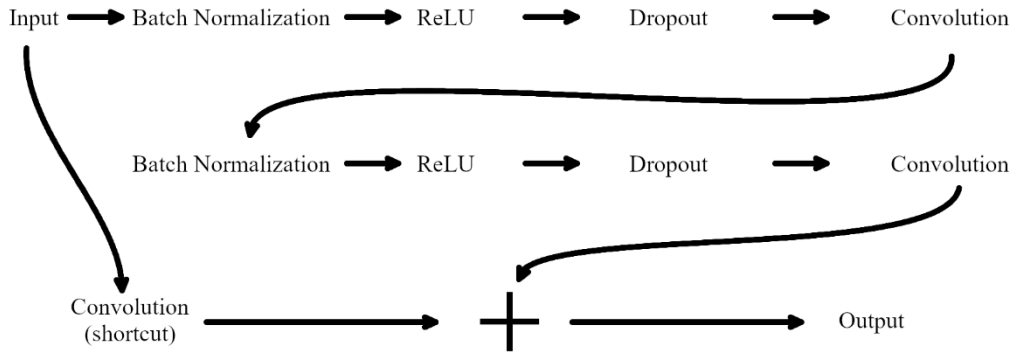


21. Figure: Autoencoer V2 structure

The input image goes through several Residual Blocks and Pooling or Upsampling layers. A black arrow means that the output of a layer is the input of the next, and a green arrow represents a skip connection.

A skip connection allows one encoding block to connect directly to the corresponding decoding block, providing very useful information to the decoder as it will know how the output of the encoding block was. This way there's a better information flow throughout the autoencoder and all the residual blocks (both encoding and decoding) will learn much better which characteristics of the images are the most important.

The inner workings of the Residual Blocks are the next:



22. Figure: Residual block

The only difference between the encoding and decoding blocks is that the first Convolution layer and the shortcut Convolutions are transposed. The logic of this block was extracted from the ResNet V2 (Microsoft Research, 2016) model.

The residual connections help mitigate something known as the “Vanishing Gradient”. The gradient can be understood as how much parameters inside the model need to change to improve performance, and the vanishing gradient means that the gradients become so small that the parameters change very little, resulting in no improvement in performance.

The structure of the residual blocks can be seen in the image above. There are two sequences of: Batch Normalization, ReLU activation, Dropout and Convolution. Only this first convolution layer makes a dimensionality change in the image, as the parameters of the second conv layer makes it learn without changing the output size.

What makes this a residual block is the shortcut, as it works as a path for the original image to be “passed” to the following blocks. This shortcut is a Convolution layer that only does the dimensionality change with as few changes as possible in aims of letting next blocks or layers “know” how the original image was. This can prevent the vanishing problem, as it will never get too small because the original image always gets added to the output.

6.3.3.3. Classification model

The classification model takes the encoder from the autoencoder and uses it to classify the latent representation of the image that it yields. This makes the end result of the classifier better as it uses an encoder that already knows how to extract the most important features or characteristics from the image (as previously this encoded image had to be decoded).

6.3.3.3.1. Classifier V1

This first version of the classifier is pretty simple. As explained, it takes the encoder and passes its output through a flattening layer and through a final dense layer. When training the classifier, the last layers of the encoder also get fine tuned, so the outputs of the encoder fit better the two layers of the classifier.

We tried adding more intermediate layers between the flatten and the final dense, but found out that no matter what we tried, the results of the classifier were being capped by the encoder, so we started developing the autoencoder and classifier V2.

6.3.3.3.2. Classifier V2

This second version of the classifier is more complex, as it uses an encoder with residual connections (it doesn’t have the skip connections as they only work when used with the decoder), and several layers, like: an initial flattening, three dense layers, two batch normalizations, two ReLU activations and two dropouts. What most of those layers do is prevent overfitting, as they do in the autoencoder.

There is another big difference between the V1 and V2 classifiers, and that is the size of the tensor that the encoder outputs. The original encoder outputs a tensor of size 14x14x512 (a total of 100352 values), and the V2 encoder outputs a tensor of size 8x8x256 (a total of 16384 values). This means that the second encoder encodes more the image, and those values are very representative of the original image as previously it was able to decode back it into the original image. What this means is that the classifier has less values to take into account, but the values it gets are the most important features of the images. Overall, this makes the classification process simpler and more effective.

6.3.3.4. Training and validation of the models

Both the V1 and V2 use the same methods for training, except for the optimizer. Also, both the training of the autoencoder and classifier are almost identical, the only differences are the metrics.

The V1 model uses the Adam optimizer (PyTorch contributors, 2023a), and as a result of the research for making a second improved model, the V2 uses the AdamW optimizer (PyTorch contributors, 2023b). The difference between these two models is the way the weight decays are calculated:

- Adam uses a simple method where the weight decay value is added directly to the gradient. This can impact the training negatively as the addition could affect the gradient in a negative way.
- AdamW uses more complex techniques where the weight decay is applied directly to the weights of the model instead of the gradient, thus fixing the issue of the gradient being affected by it.

6.3.3.4.1. Autoencoder training

The autoencoders use the Mean Squared Error (PyTorch contributors, 2023d) as the loss metric (how different are the outputs to the inputs), as this loss compares the values of each pixel of the images. The lower this metric is, the better, as that means that both inputs and outputs are similar, meaning that the autoencoder knows how to encode and decode the image properly.

6.3.3.4.2. Classifier training

The classifiers use two metrics: Cross Entropy Loss (PyTorch contributors, 2023c) and accuracy.

The easiest of these is the accuracy, as the calculation is just taking the predicted class from the classifier, and checking if it's correct. This comparison is done for the entire batch, and a percentage of correct guesses is created. The higher the accuracy is, the better, as more images got the class predicted correctly.

The Cross Entropy Loss works by taking the predicted class and the confidence level of the prediction. This confidence level is the probability of that class to be the correct one. When the predicted class is wrong, the loss increments depending on the confidence value. If this value was high (so the classifier was sure that that was the class, but it was wrong), then the loss increments more compared to if the confidence was low (so the classifier

thought that that was the correct class, but it wasn't very sure of it). As this loss increases, the lower it is the better.

6.3.3.5. Image predictor

Apart from the autoencoder and classifier, there is one more class that serves as a predictor. This class gets the output files of the encoder and the classifier where the models are saved, and it creates an instance of the classifier. Then, this instance can be used for predicting what the classes of the dishes are.

During this prediction (or inferencing) the model doesn't learn or change. This means that the model is deterministic, as the output of the models will always be the same if the image is not changed.

The predictor has two functions for predicting: a function for predicting the class of a single image, and a function for predicting the classes of several images in a batch. The two functions work the same way: they take the image, transform it so it has the same properties as the images the models got trained with, make a call to the model to get the class of the image, and return in a json format the next things: the most confident class and its confidence value, and a list of all the other possible classes with their confidence values.

6.3.4. Python API

Finally, the manner in which all of the models were integrated into the other components was to make a Python API using the Flask Framework. It provides endpoints necessary to communicate with each module of the system, including the calorie prediction, the meal recommendation, and the food recognition models.

6.3.4.1. Views & Error Handling

Several views have been created to support communication between the backend Java application and the frontend NodeRed application with the AI modules, as described below:

- **Calorie intake prediction (‘/intake_prediction’, POST) :** Processes a user's biometric information to predict calorie intake, accepts a JSON input containing the user's ID, queries the database and returns the intake response from the llama3 model.
  If the given user ID does not match anything in the database, or if the content type of the request is different from a JSON payload, the view throws the necessary exception.
- **Image classification (‘/image_prediction’, POST):** Processes an uploaded image sent from the Node-Red frontend to classify it into a given food class trained in the AI model. It returns a dictionary containing the name of the class with the highest confidence and a list of the confidence levels of all other classes.
  If the request does not contain an image file, a Bad Request exception will be thrown.
- **NutriChat (‘/chat’, POST):** Processes a given request from the user's Telegram chat and exchanges it with the llama3 chatbot. Returns the model's answer to the conversation.

Additionally, several error handlers have been created using the Flask framework's 'errorhandler' decorator to handle the above exceptions and any wildcards, to provide a common error message structure with meaningful HTTP error codes, without exposing any sensitive debugging information to either the clients or the terminal.

6.3.4.2. Application Environment Variables

An .env file has been created to define essential application settings, model paths, environment variables, database keys, etc. in a run-time safe manner according to Flask framework standards:

- **Flask settings:**
  - **FLASK_ENV**: Set to `development`, enabling debugging features for easier development.
  - **FLASK_DEBUG**: Set to (`True`) to enable debug mode, which provides detailed error messages during development.
  - **SECRET_KEY**: A placeholder for a secret key, which is essential for securing session tokens within the Flask application.
- **Upload folder configuration:**

    - **UPLOAD_FOLDER**: Defines the directory path where uploaded files, such as images, are stored on the server. The absolute path specifies the location within the user's document folder.
- **Model paths:**
    - **MODEL_PATH**: Specifies the path to the trained model used for food recognition or classification.
    - **MODEL_ENCODER_PATH**: Specifies the path to the encoder model used to process input data for the AI system.
- **Other settings:**
    - **ALLOWED_EXTENSIONS**: Specifies the file types that are allowed to be uploaded (PNG, JPG, JPEG).
    - **MAX_CONTENT_LENGTH**: Limits the size of incoming files to 16MB to avoid overloading the server.
- **Database settings:**
    - **HOST**: The database host, which is set to `localhost` to point to a local MySQL instance.
    - **USER**: Specifies the MySQL username.
    - **PASS**: The MySQL password for authentication.
    - **DB**: Specifies the name of the database (`janai`), where data related to the application will be stored.

6.4. Code testing

In order to maintain a high code quality, prevent and identify bugs and errors before they happen, and ensure that the finishing product meets the initial expectations, numerous tests were created and run during the whole development process.

Being the code separated in two main parts, the web backend written in java and Flask API and the AI models in python, each of these parts needed their own sets of testing. In these chapter, the tests for these two parts will be explained more thoroughly and in more detail.

6.4.1. Backend testing

Backend testing is essential for evaluating a software's server-side components, such as databases, repositories, APIs, controllers, etc. This process ensures that data processing, storage, and retrieval functions work as intended.

Backend testing helps ensure that the system operates reliably and meets user expectations by identifying potential issues that could impact data integrity, performance, and security.

The main components of the backend that were subject to code tests were the models, data structures that transfer the desired data between layers, controllers, mediators responsible for accepting http requests from the user, modify models and convert them for the view, and the security classes meant to limit the access to certain resources for some users, encrypting passwords and so on.

6.4.1.1. Model testing

Testing backend models is relatively easy and simple, not having almost any logic at all, these java classes were the first tested.

The models are representations of the databases tables, and they only have the following functionalities: three constructors, a default one without any parameters, another one with all the parameters, and a third one with every parameter except the id, and the getters and setters for each parameter.

To test these functionalities, the logic has been the following:

- **Default constructor:** For the default constructor tests, first, an object is created with the default constructor, and their parameters are checked if they're null. For the id, compare it with the number zero.
- **Constructor without id:** For these tests, the id is also compared with zero and the rest of the parameters are first created as local variables, and then added to the constructor. Once the object is initialized, the local variables and the objects parameters are compared.
- **Constructor with all parameters:** Same as before but including the id as a local variable.
- **Getters and setters:** The getter and setter tests are similar to the previous three but with a small twist. First, the object is initialized with its default constructor. After that, every parameter is initialized as local variables, and, using the setter method for each parameter, they are set as the objects parameters. Lastly, in the assertion part of the test, the local variables are asserted to be equal to the getter method for that exact parameter, if the assertion is correct, the test completed as expected.

6.4.1.2. Controller testing

Comparing the controller tests and the previously explained model tests, the controller tests are more complex than them, due to their nature and functionalities. Controllers usually have dependencies in complex objects such as repositories, in order to answer the user's GET, POST, PUT, DELETE... requests, and these repositories find what's been requested in databases, meaning that the requests ask for something that exists independently on a database.

To address that issue in the tests, mocking is used. Mocking is basically creating objects that simulate the behavior of real objects, called mocks or mock objects. Mock objects are used during mock testing to isolate and test specific functionalities without relying on the actual implementation of dependent components.

In order to help with the mocking tests, the project uses the Mockito framework. Mockito is a mocking framework that lets you write tests with a clean and simple API. Mockito tests are readable and easy to understand, and the verification errors are easy to understand. For these reasons Mockito was used in the project.

The tests have been created following the Act, Arrange, Assert pattern for unit testing. This patterns idea is to create the test following the next three steps:

- **Arrange:** setup the testing objects, such as repositories and controllers, and prepare the prerequisites for the test
- **Act:** perform the actual work for the test, calling mock objects methods for example.
- **Assert:** verify the result.

The process for each controller test class is the following, taking UserDataControllerTest.java as an example.

First, the controller is declared with the InjectMocks annotation, this is because InjectMocks injects mocked dependencies such as repositories in the annotated class mocked object. Then the dependencies are declared with the Mock annotation and the setup() method is created. The setup() method is called before each test and it calls MocitoAnnotations's openMocks() method, initializing every object that has a mockito mock annotation.

After that, the tests are created. Following the previously explained Arrange, Act, Assert pattern, the controller method tests start with initializing an instance of a model and calling the when() method of the Mockito framework. This method tells a mock to return a certain value when a certain method is called. See the image below for an example. Then, the method subject to the test is called, usually returning a ResponseEntity. Finally, the response status code is asserted with the expected http status code. If the response is supposed to return another value apart from the status code, the response body will be asserted.

6.4.1.3. Security and email service testing

Regarding the security and email service tests, since most of them are dependent on more complex objects, these tests are similar to the controller tests, with a main difference being that mock objects aren't declared with annotations, since most of them only use one mock.

Thanks to all of the previous tests, the project achieved a test coverage of 93.4%, with high quality tests including the clear use of mocks.

6.4.2. Python Testing

As with the backend test suite, it is essential to test and validate the code quality of all the additional web services used for an application with a distributed network of APIs, servers and components, including in particular the Flask API used to provide and interface with the LLM and image recognition models used by the JanAI application as a whole. A test suite was written using the pytest framework (Pytest-dev team, 2015) for the modules that contain key functionality, including:

- Calorie and recommendation LLM model interfacing
- Image recognition model interfacing and upload pre-processing
- Application endpoints and error handlers

6.4.2.1. View & Error Handler testing

Firstly, testing the views of the API requires additional setup, as there is no fully fledged test server with a dedicated embedded database, so most of the behavior related to querying the database and accessing the local file system must be properly mocked to isolate the tests and validate only the behavior of our relevant code. As such, the entire suite has been organized according to the same main phases of **Arrange**, **Act**, **Assert**.

Starting with an important preparatory step, a pytest fixture method was created to provide a test client set up with specific environment variables and a mocked connection to the database using the patch decorator. All tests in the suite are based on this client.

Using the test case for the view that provides calorie intake predictions as an example, we start by mocking the response from the calorie LLM's calculate_calories method so that its return value is 2000 kcal and include the base fixture prepared test client as an argument to be used within the test. An important observation is that this test suite is written specifically for the views of the API, so the responses of the AI model are only mocked and not directly called.

We then collect the response of a POST request to the /intake_prediction endpoint with a JSON payload passing a generic user ID of 0 and assert that the status code is 200 and that the response contains the mocked calorie prediction of 2000. Both the success cases and the exception handling blocks were tested, and in those cases the default mocks for each model were modified to have the expected side effects, such as not returning any user data or wildcards.

6.4.2.2. Calorie and Recommendation Model testing

The suite for validating the calories LLM follows a much simpler structure, as the calories module mainly involves parsing a prompt template with user data from the database, then sending and returning the response from the Ollama3 model. The LLMChain call and database connectors were mocked, with only an additional test case for a missing user.

In the case of the LLM recommendations, the same structures were mocked, with the addition of mocking the embeddings and the Chroma database used by the Ollama (Meta, 2024) chatbot.

6.4.2.3. Recognition Model and Preprocessing testing

The recognition model test suite validates its core functionalities that can be tested with the limited VM testing server, the coverage includes preprocessing, the autoencoder, the classifier, and the prediction pipeline.

- **Preprocessing:** The `preprocess` function was tested using mocks for file and image operations to verify correct resizing (to 224x224), alpha channel preservation, and saving of processed images. Cases include nonexistent directories (ensuring creation) and invalid paths.
- **Residual Autoencoder:** A forward pass test ensures the encoded tensor reduces dimensions correctly while the decoded output matches the input shape.
- **Classifier:** Tests verify the classifier produces outputs with the correct shape (batch size, number of classes) and integrates properly with the encoder.
- **Image Predictor:**
    - **Single Image:** Tests mock image loading, transformation, and inference to confirm predicted class, confidence values, and softmax probabilities.
    - **Batch Prediction:** Validates handling of multiple images, ensuring accurate predictions for each input.
- **Class Probabilities:** Ensures probabilities are correctly computed, with the top class matching the highest probability.

6.4.3. Static & Dynamic Analysis Automation

In order to automatically generate static and dynamic analysis reports, a combination of the Github Actions (Microsoft, 2019) platform and a standalone virtual machine hosted on Google Cloud was used to run the SonarQube (SonarSource, 2007) scanner for both the JanAI Java backend and AI API projects. The following conditions were decided to be the standard to be followed by the quality gate of the JanAI projects, starting with **New Code:**

- At least 10 new issues are introduced
- All security hotspots are reviewed by the development team
- Coverage is greater than or equal to 80%
- Less than 3% of the code base is duplicated code

As for the **Overall Code** conditions:

- Total coverage is greater than or equal to 75%
- Cyclomatic complexity is less than or equal to 10
- Total duplication of code blocks is less than or equal to 2%.
- A maximum of twenty maintainability or reliability issues.
- A maximum of one security issue.

This Quality Gate has been configured and set for both projects to maintain a coding standard according to the **Clean as you Code** methodology, a software development

practice that operates on the basis that when new code is pushed into a given repository, it must meet a quality standard before it is released.

This ensures that new code, even if it contains changes to existing code, contributes positively to the overall quality of the project, with the goal being that the entire code base is in a state of clean code.

Additionally, in terms of the Github action files, as soon as something is merged into the main branches, the site creates a job that communicates with the compute machine to check out all the code, build the project and install all the necessary dependencies needed to run the test pipeline. The coverage report is then generated and processed by the dashboard.

The details of the steps used in each Github action, for the Java repository and the AI repository respectively, are detailed in the following section.

6.4.3.1. Backend Workflow

- **Checkout Code:** The repository's latest code is checked out using the actions/checkout@v3 and made available for analysis and testing.
- **Set up JDK 17:** Java 17, the required runtime for the backend project, is set up using the actions/setup-java@v3 action with the Temurin distribution.
- **Build and test the project:** The project is built and tested using Maven's mvn clean test command.
- **Run SonarQube analysis:**
    - The SonarQube scanner is downloaded and extracted to the virtual machine runner.
    - The scanner is used to analyze the project, including uploading test results and coverage data.
    - The scanner waits for the SonarQube Quality Gate result to complete, ensuring that only compliant code is released by the workflow.
    - Debugging information is enabled using the -X flag for detailed logs in case of errors.

6.4.3.2. Python Workflow

- **Checkout code:** The latest code from the repository is checked out using actions/checkout@v3.
- **Set up Python:** The required Python runtime (version 3.11) is set up using actions/setup-python@v4.
- **Install dependencies:** All necessary dependencies are installed using pip and the requirements.txt file.
- **Install PyTorch and Chroma:** The workflow also installs PyTorch (with CUDA support) and Chroma explicitly using pip, as these are also important dependencies, but require specific arguments for the install command.
- **Set permissions for the uploads folder:** The workflow creates a temporary upload folder, sets permissions, and generates placeholder files (1.jpg, 19.jpg) to simulate file upload scenarios. These placeholders ensure that any tests related to file handling can be executed seamlessly.
- **Run tests and generate coverage report:**

  - Tests are executed using the pytest framework with coverage reporting enabled.
  - Code coverage data is captured and stored in an XML report (coverage.xml) for analysis.
  - Environment variables such as APPLICATION_ROOT, UPLOAD_FOLDER, and MAX_CONTENT_LENGTH are configured to match the application settings.
- **Run SonarQube analysis:**
  - The SonarQube scanner is downloaded and prepared in the runner.
  - The project is analyzed, including the uploaded coverage report for accurate test coverage statistics.
  - The scanner enforces Quality Gate compliance by waiting for its evaluation before allowing the code to be approved.

7. Conclusion

Once the evolution of the project has been explained, the obtained conclusion will be portrayed. On the one hand the solution given to the raised problem will be evaluated, and compared to the initial objectives displayed at the beginning of the report and check if it has been fulfilled or not. Finally the SDGs will be further elaborated, justifying their compliance.

7.1. Achieved goals

As mentioned earlier in this report, the goal of JanAI is to increase the user's quality of life while being accessible and easy to use. Nowadays 1 in 8 people in the world suffer from obesity, taking the lives of 2.8 million people each year (World Health Organization, n.d.). The number of people with diabetes also rose from 200 million in 1990 to 830 million in 2022, one of the main causes being an improper diet (World Health Organization, 2024).

In short terms, a possible solution has been made for the raised problem, since a milestone was to provide an application that's free, easy to use and helpful for the user by using modern technologies. Another point to note is that the user's satisfaction is the first concept in mind, due to the fact that this web app is completely personalized for each user, taking into account aspects like allergies, diets or personal goals.

Regarding the objectives of the project, in the third point of this report various objectives to be achieved were specified, these being divided into two separate sections: Improving the user's quality of life and to make an application that's accessible and easy to use.

According to the user's quality of life, one of the sub objectives was to help with weight management, and in accordance with the developed product, this point was fulfilled thanks to the implementation of weight goal charts and NutriChat being able to recommend meals depending on the goal to achieve. This is something that LogMeal (AIGecko Technologies S.L., n.d.) or Calorie Mama (Azumio Inc., 2017) do not take into account. Secondly, to improve the user's diet was also wanted, and to help with this, the main page of JanAI where the most important macro consumption can be seen is very helpful, and again with the help of NutriChat the application is able to give healthy alternatives or options for diets fully personalized for each user. Besides, image recognition model also helps a lot in this duty, giving JanAI the opportunity to track meals and display macronutrient information of a scanned food. Thirdly, the last sub objective was to remind the user to stay hydrated, recommending the amount of glasses of water that should be consumed along with the calories, so every time the user enters the application, it will be reminded to drink water thanks to the implemented graph displaying how much water has been consumed.

The other main objective was directed towards the application itself and not the user, ensuring the app is easy to use for everyone. The first sub objective was to create an app with no premium limitations. As seen in the state of the art, the competitors tend to lock the most interesting features behind a subscription which not everyone wants to pay for, leading to demotivation. In the development process the customer was the main protagonist, so all functionalities are available from the start to keep the user motivated to improve their health. The second sub objective was to make the interface easy to use, and with the help of Node Red's dashboard the developed interface is intuitive and easy to use, with the correct labels on the pages, buttons, graphs, etc. This decreases the user's frustration caused by an app that's not so easy to use.

7.2. SDG impact

In addition to the technical goals mentioned above, JanAI actively contributes to various Sustainable Development Goals (SDG) (United Nations, 2015), thanks to JanAI's intelligent advice and the ability to register the user's meals, along with the nutritional value of each one, promoting good health and a balanced diet. As a strategy, in line with these goals, the official SDG impact assessment tool was used to maximize the positive impact. The strategy includes giving each user a detailed daily macronutrient target to achieve their goals, personalizing the experience with the app to a personal level, and the nutritional advice provided by the nutrition chatbot to help with dietary decisions. Each user's data can also be used in future studies on people's eating habits and the use of nutrition apps. All this with the sole aim of improving people's health and eating habits.

In terms of impact on the Sustainable Development Goals, the following have a direct positive impact on them.

7.3. Good health and well being

JanAI promotes a healthy and balanced diet and gives personalized advice to each user to prevent future illnesses linked to poor eating habits. Diseases such as obesity, malnutrition.... Thanks to the nutrition chatbot, the project is making a positive contribution to this goal.

7.3.1. Target 3.4: Reduce mortality from non-communicable diseases and promote mental health.

Providing dietary advice and the nutritional value of different foods significantly reduces the risk of developing diet-related diseases. To back this up, a study conducted by the National University of Singapore showed that users who actively engaged with ≥5 app features achieved an overall weight loss of 10.6% from baseline after 6 months and reduced glycated hemoglobin (the main cause of diabetes) by 1.2% (National University of Singapore, 2022).

23. Figure: SDG 3.4

7.3.2. Target 3.8: Achieve universal healthcare

By making expert nutrition advice widely available, such chatbots help improve health equity and provide essential services, especially in underprivileged or remote areas.

24. Figure: SDG 3.8

The following SDGs are affected indirectly by the project.

7.4. Zero hunger

JanAI informs the user about the nutritional values of different foods, especially their macronutrients, and gives advice on how to eat a balanced diet, thus indirectly combating malnutrition.

7.4.1. Target 2.2: End all forms of malnutrition, including addressing the nutritional needs of vulnerable groups.

By providing personalized food recommendations, JanAI helps combat malnutrition and promotes nutritious meals for diverse populations.

25. Figure: SDG 2.2

7.4.2. Target 2.1: Ensure access to safe, nutritious, and sufficient food for all people, especially vulnerable populations

Thanks to the app's food recognition feature, users can be encouraged to make healthier dietary choices, reducing various nutrient deficiencies and making users more independent in their food choices through education.

26. Figure: SDG 2.1

7.5. Responsible consumption and production

By providing users with the calories and macronutrients they need for the day, they will be able to plan their meals more efficiently and thoroughly, buy the food they need each time, save some money and encourage responsible consumption, making the production of these foods more realistic and reducing waste.

7.5.1. Target 12.3: Halve per capita global food waste at the retail and consumer levels.

Users can ration their food to the amount needed, minimizing over-consumption and reducing food waste by providing accurate nutritional information per 100 grams of the specified dish.

27. Figure: SDG 12.3

7.5.2. Target12.8: Ensure that people everywhere have the relevant information for sustainable consumption

JanAI teaches users about the nutritional value of different dishes, increasing food knowledge and informing users every time they eat.

28. Figure: SDG 12.8

7.6. Industry, innovation and infrastructure

JanAI innovates the nutrition/fitness app market by adding the power of AI to its functionalities, making it capable of recognizing different foods with state-of-the-art image recognition models, and giving the assistant 'life' by giving advice in real time thanks to a large language model.

7.6.1. Target 9.5: Enhance research and technological capabilities.

JanAI uses innovative AI image recognition models and large language models to provide users with personalized advice and nutritional information about the foods they consume, driving innovation in the health tech space.

29. Figure: SDG 9.5

The remaining Sustainable Development Goals are not affected by the existence of this project. For more detailed information on the SDGs relevant to JanAI, please see the report produced via the SDG Impact Assessment Tool available in **SDG Impact Assessment Tool analysis**.

8. Future objectives

It must be taken into account that this product is a prototype and some functionalities are not developed fully, though it would be possible to add the remaining ones in the final product. In this section the improvements and other additions will be mentioned.

8.1. Improvements

To start off one of the key improvements would be to implement history in the hat, so it does not have to be reminded constantly about the previous queries or other essential information the model should already know about. This would improve the usability of the app and overall experience. The RAG of the LLM could also be improved, to get more relevant and precise answers and less hallucinations.

On the other hand, right now the recommended macronutrients are calculated by using the same percentages, in other words, the percentage of calories, proteins or fats that have to be ingested in the diet are calculated using the same ratio. Ideally, the ratio should change depending on the diet the user wants to follow. For example, if the desire is to gain weight, calories should take up a larger percentage than fiber. The main idea would be to implement different ratios for each diet.

Finally, it would be a very nice idea to show this app to nutritionists and get feedback from them, since the developers are not nutritional experts and are relying on data from the internet.

8.2. Additions

When it comes to the possible additions, three would substantially improve the usability of JanAI.

Firstly, to give the option to talk with a nutritionist would be very nice, this could be added in a similar way like the NutriChat, where a button redirects to telegram with an open conversation. This feature would be useful, since we are not nutritional experts.

Some additions could be made for the image recognition part too, right now and image has to be uploaded to the app, this requires the user to first take a picture, save it and then use it. Instead, it would be a useful fix to implement real time recognition by using the camera, eliminating the need to save any pictures. And finally, there might be times that the user is not able to use the camera, so a list could be implemented with registered foods so the user can select the one that’s eating and receive information quicker than with the camera.

9. Bibliography

AIGecko Technologies S.L. (n.d.). *LogMeal*. 2021.

Auth0. (2015). *JWT*. https://jwt.io/

Azumio Inc. (2017). *Calorie Mama*.

Donald D. Chamberlin, & Raymond F. Boyce. (1974). *SQL*.

FAIR. (2016). *PyTorch*. Https://Pytorch.Org/.

Guido Van Rossum. (1991, February 20). *Python*. Https://Www.Python.Org/.

Khusid, A., & Shardin, O. (n.d.). *Miro*. 2011.

Meta. (2024). *Llama3*. Https://Www.Llama.Com/.

Microsoft. (2019). *GitHub actions*. Https://Github.Com/Features/Actions.

Microsoft Research. (2016). *ResNet v2*. Https://Pytorch.Org/Docs/Stable/Generated/Torch.Nn.Conv2d.Html.

MySQL AB. (1995). *MySQL*.

National University of Singapore. (2022). Association Between Mobile Health App Engagement and Weight Loss and Glycemic Control in Adults With Type 2 Diabetes and Prediabetes (D'LITE Study): Prospective Cohort Study. *Https://Pubmed.Ncbi.Nlm.Nih.Gov/36178718/*.

O'Leary, N., & Conway-Jones, D. (2013). *Node Red*.

Pytest-dev team. (2015). *PyTest*. Https://Docs.Pytest.Org/En/Stable/.

Python threading. (n.d.). Https://Docs.Python.Org/Es/3.8/Library/Threading.Html.

PyTorch contributors. (2023a). *Adam optimizer*. Https://Pytorch.Org/Docs/Stable/Generated/Torch.Optim.Adam.Html.

PyTorch contributors. (2023b). *AdamW*. Https://Pytorch.Org/Docs/Stable/Generated/Torch.Optim.AdamW.Html.

PyTorch contributors. (2023c). *CrossEntropyLoss*. Https://Pytorch.Org/Docs/Stable/Generated/Torch.Nn.CrossEntropyLoss.Html.

PyTorch contributors. (2023d). *MSELoss*. Https://Pytorch.Org/Docs/Stable/Generated/Torch.Nn.MSELoss.Html.

PyTorch formula. (2023). PyTorch Contributors.

Ronacher, A. (2010). *Flask*.

SonarSource. (2007). *SonarQube*. Https://Www.Sonarsource.Com/Products/Sonarqube/.

Statista. (2023). *Nutrition Apps - Worldwide*. Https://Www.Statista.Com/Outlook/Hmo/Digital-Health/Digital-Fitness-Well-Being/Health-Wellness-Coaching/Nutrition-Apps/Worldwide.

Sun microsystems. (1995). *Java*.

United Nations. (2015). *SDG*.

World Health Oganization. (2024). *Obesity and overweight*. Https://Www.Who.Int/News-Room/Fact-Sheets/Detail/Obesity-and-Overweight.

World Health Organization. (n.d.). *Obesity*. Https://Www.Who.Int/Data/Gho/Indicator-Metadata-Registry/Imr-Details/3420.

World Health Organization. (2024). *Diabetes*. Https://Www.Who.Int/News-Room/Fact-Sheets/Detail/Diabetes.

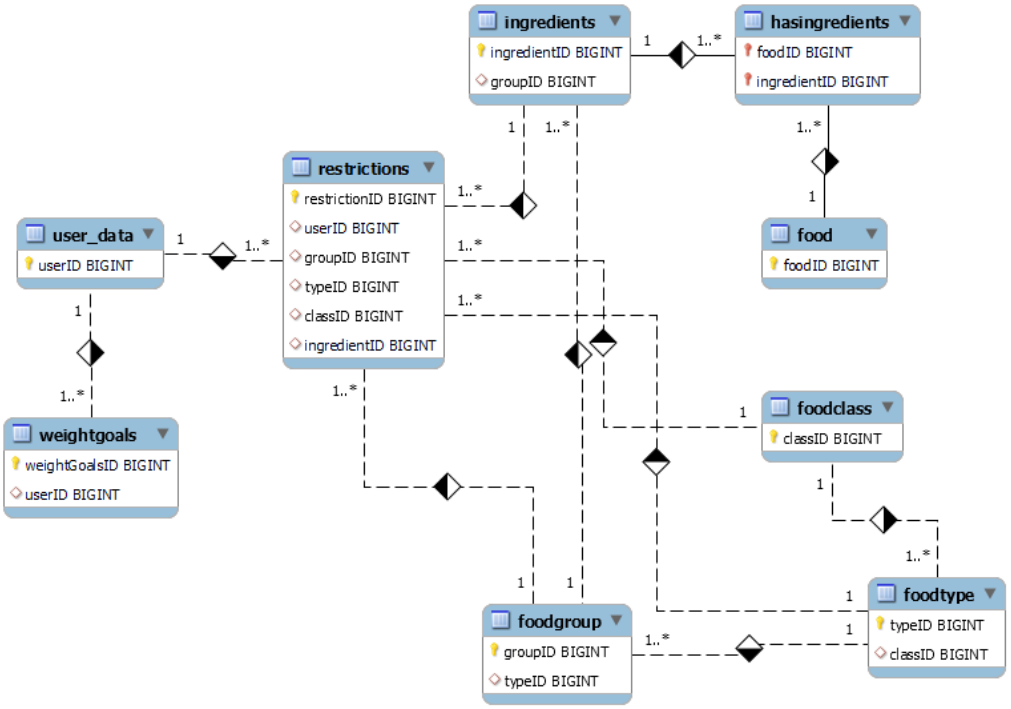
10. Annexes

10.1. MySQL DataBase

All the information used in the web page is saved in a database of MySQL.

10.1.1. Diagram:

In this photo we can see all the tables of the database and each primary key and foreign key.



10.1.2. Trigger:

There is an LLM to calculate the daily calorie intake. But all the data sent to the LLM is calculated with different formulas by this trigger. This trigger is executed automatically after an insert operation on the weightGoals table, ensuring that user's calorie-related metrics are kept up-to-date in the database.

This trigger automates the computation of several health-related metrics, based on a user's inputted data. The results are then updated in the userData table, providing a centralized source for user health data.

The body fat percentage is calculated with this formula:

Men: $BFP = 100 * ((4.95 / (1.0324 - 0.19077 * log10(waist - neck) + 0.15456 * log10(height))) - 4.5)$

Women: $BFP = 100 * ((4.95 / (1.29579 - 0.35004 *$

$log10(waist + hip - neck) + 0.22100 * log10(height))) - 4.5)$

The basal metabolic rate (BMR) is calculated in 3 different ways.

1-Mifflin-St Jeor Equation:

Men: $BMR = 10W + 6.25H - 5A + 5$

Women: $BMR = 10W + 6.25H - 5A - 161$

2-Revised Harris-Benedict Equation:

Men: $BMR = 13.397W + 4.799H - 5.677A + 88.362$

Women: $BMR = 9.247W + 3.098H - 4.330A + 447.593$

3-Katch-McArdle Formula:

$$BMR = 370 + 21.6\,(1 - F)\,W$$

The total daily energy expenditure (TDEE) is calculated in 3 different ways for each activity level, as it uses the 3 different BMR calculated before.

$$TDEE = BMR * Activity\ Multiplier$$

Activities = {

Sedentary: 1.2;

Light: 1.375;

Moderate: 1.55;

Active: 1.725;

Very Active: 1.9;

}

The total weight loss is calculated with this formula:

$$TWL = weight - goalweight$$

The weekly calorie deficit is calculated with this formula: (1kg =~ 7700 kcal)

$$WCD = (TWL * 7700) / durationToAchieveGoalWeight$$

The daily calorie intake is calculated in 3 different ways, as it uses the 3 different TDEE calculated before.

$$DCI = TDEE - (WCD / 7)$$

The water intake is calculated with this formula:

$$DWI = weight * 0.033$$

10.1.3. Event:

There is an Event to put the daily water counter into 0 when the 00:00 hour reaches of each day. This is made by creating an event in the database and scheduling to execute

everyday at 00:00, and updating the userData table the variable of all users waterCounter to 0

Db	Name	Definer	Time zone	Type	Execute at	Interval value	Interval field	Starts	Ends	Status	Originator
janai	ResetWaterCounter	root@localhost	SYSTEM	RECURRING	NULL	1	DAY	2025-01-23 00:00:00	NULL	ENABLED	1

10.1.4. Image recognition dataset:

When the image recognition model predicts an image and return a food, we see the database to find that food macros and print them in the graphs of the image result page. To make this possible, we had to take all tha classes that are 101 and create 101 identical named foods with each macro as can be seen in the photo.

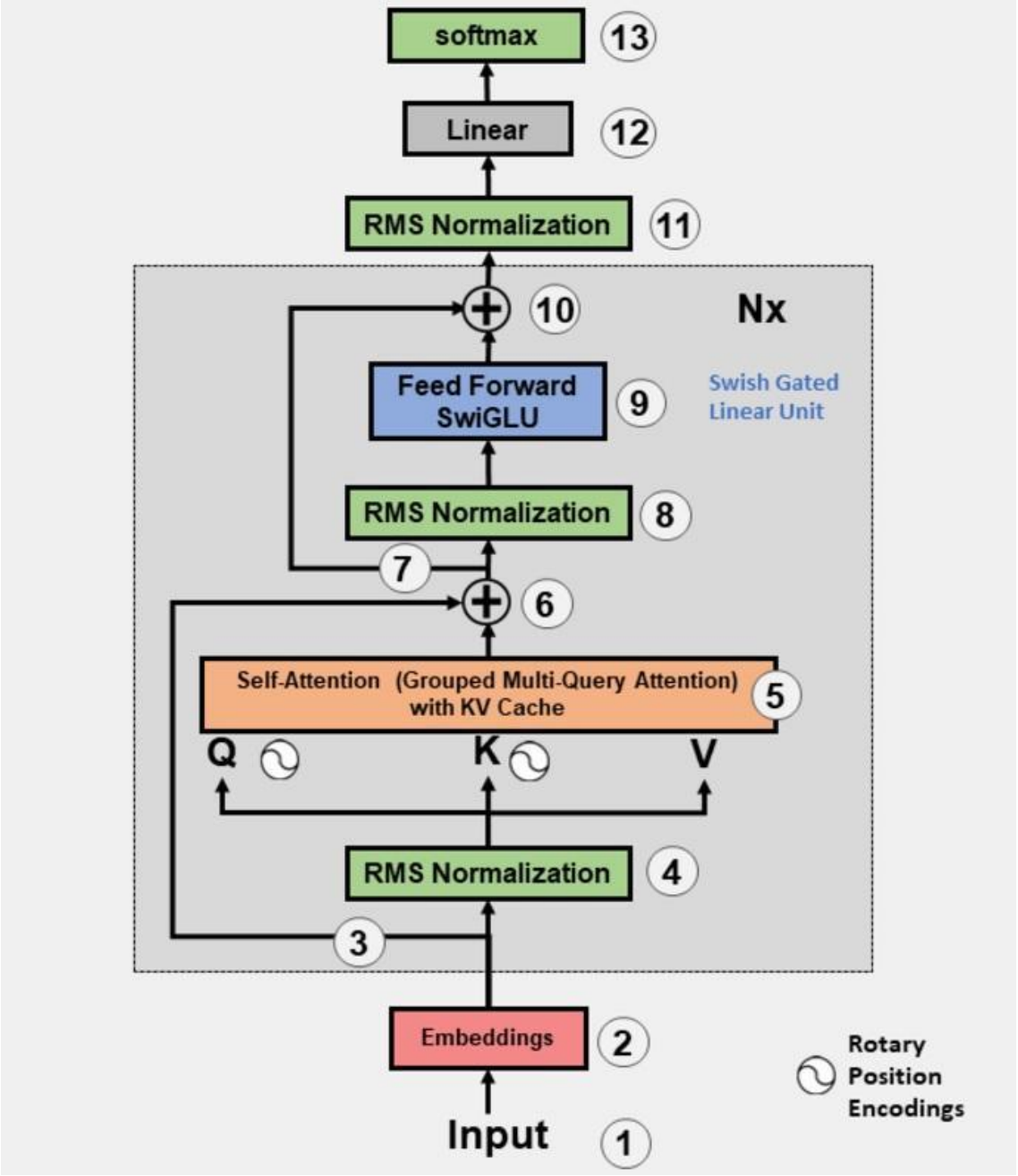
```
INSERT INTO food(foodID, foodName, proteins, carbs, fats, fiber, calories) VALUES
  (1, 'deviled_eggs', 11.57, 201, 16.23, 0, 16.23),
  (2, 'gyoza', 7.5, 22.2, 7, 0, 185),
  (3, 'escargots', 16.1, 2, 1.4, 0, 90),
  (4, 'fried_calamari', 15.13, 9.9, 2.17, 0.5, 125),
  (5, 'mussels', 11.9, 3.69, 2.24, 0, 86),
  (6, 'oysters', 7.05, 3.91, 2.46, 0, 68),
  (7, 'sashimi', 21.62, 0, 5.93, 0, 146),
  (8, 'scallops', 18.14, 10.49, 10.96, 0.5, 217),
  (9, 'tuna_tartare', 4.58, 17.09, 3, 0.76, 266),
  (10, 'eggs_benedict', 11.96, 8.32, 21.6, 0.6, 276),
  (11, 'huevos_rancheros', 6.89, 8.53, 6.66, 1.5, 119),
```

10.2. Macros daily intake:

In node-red, in the main page, there is a function where the user's daily macros are calculated. As you can see in the photo, the carbohydrates are the final calorie intake multiplied by 0.45 and divided by 4. Fats are multiplied by 0.25 and divided by 9. Proteins are multiplied by 0.3 and divided by 4. And finally fiber is set 30 to men and 25 to women.

```
msg.max = msg.payload;
msg.carbs = Math.round((msg.max * 0.45) / 4);
msg.fats = Math.round((msg.max * 0.25) / 9);
msg.proteins = Math.round((msg.max * 0.30) / 4);
msg.finalDailyCalorieIntake = Math.round(msg.max);
if (flow.get("user.gender")=="M"){
    msg.fibers = 30;
}
else {
    msg.fibers = 25;
}
return msg;
```

10.3. How LLAMA3:8B works:



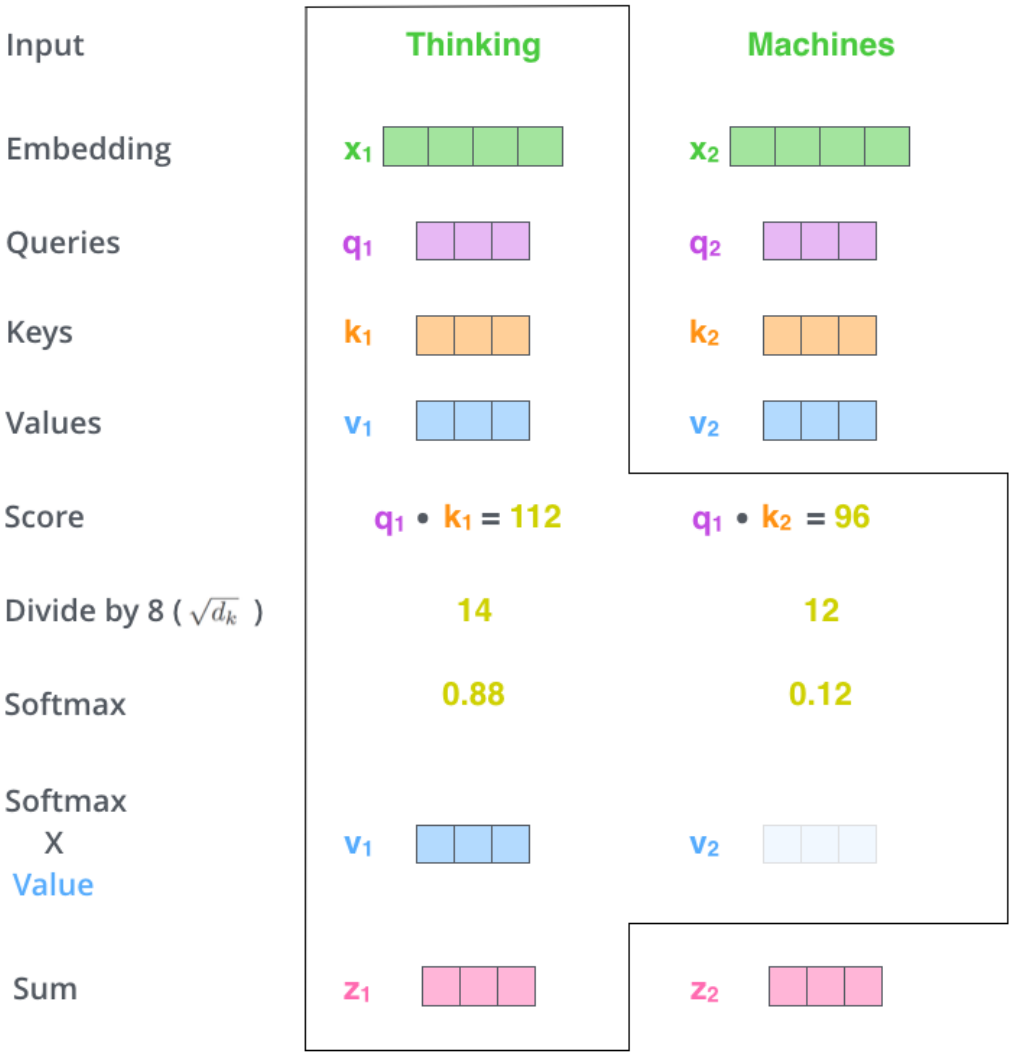
- Input: It receives the input stream, which can be text, code or any tokenizable data. Each token is an integer representing a word, character or symbol in the model's vocabulary. [github]
- Embeddings: It converts tokens into high-dimensional continuous vectors representing semantic information.
- Cache Unprocessed Active Tokens for Step 6: Stores or caches the embeddings of active tokens that have not yet been processed, for later use in Step 6.
- RMS Normalization: It normalises each vector by dividing its values by the square root of the root mean square of its components. This stabilises the inputs. [paper] [github]

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

- Rotary position embeddings: Adds positional information to embeddings through mathematical rotational transformations. [paper] [github]
  Given a token with embedding vector x, and the position m of the token inside the sentence, this is how we compute the position embeddings for the token.

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

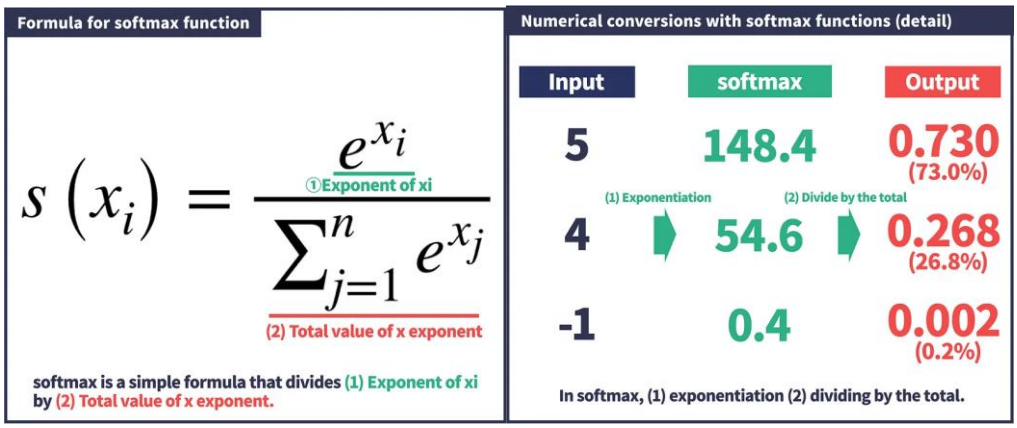
- Self-Attention (Grouped Multi-Query Attention) with KV Cache (Q, K, V): It performs self-attention on the embeddings, calculating attention scores across tokens using a multi-query clustered attention mechanism with query (Q), key (K) and value (V) vectors. [paper]



- Add unprocessed embeddings from step 3: Incorporates unprocessed embeddings from Step 3 (cached active tokens) into the processing pipeline.
- Cache Unprocessed Token for Step 10: Caches or stores the embeddings of tokens that have not yet been processed, for later use in Step 10.
- RMS Normalization: Normalises the values obtained after self-attention to avoid instability.
- SwiGLU Feed Forward: Procesa las representaciones normalizadas a través de una red neuronal feedforward que usa la activación SwiGLU (Swish Gated Linear Unit). [paper]

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

- Add Cached Token from Step 6: Incorporates cached embeddings from Step 6 (unprocessed tokens) into the processing pipeline.
- Final RMS Normalization: Standardise representations for final output.
- Linear projection: Transforms the vectors to a lower-dimensional space corresponding to the vocabulary.
- Softmax: Convierte las proyecciones en probabilidades para cada token del vocabulario.



- Tokenizer decode: Convierte la secuencia de tokens generada en texto legible. Esto se logra mediante un diccionario invertido que mapea cada número a su palabra o símbolo correspondiente.

Other data:

Params: 8B

Training tokens: 15 trillion

Attention: Grouped-Query Attention

Size context: 8.192

Tokenizer: Byte Per Encoding tiktoken

Token vocabulary: 128.256

Data knowledge cutoff: March 2023

Model Release Date: April 2024

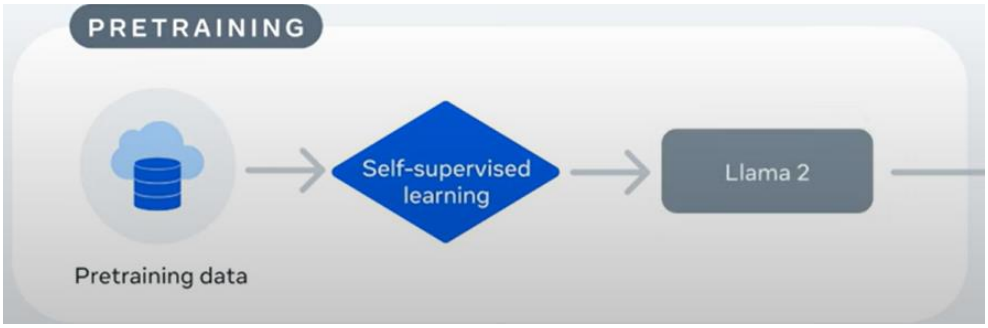
Trained for next-token prediction

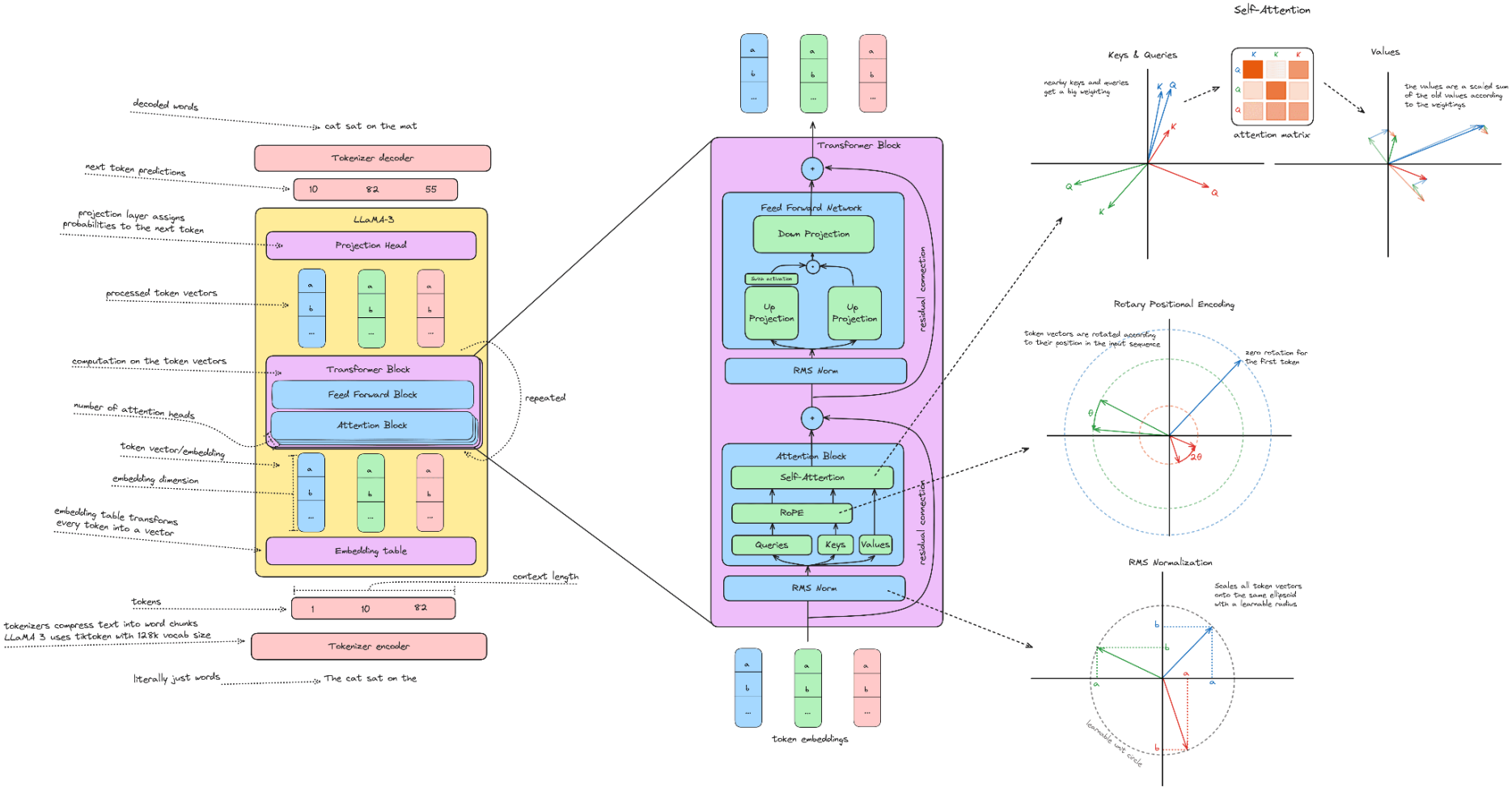
Llama only have an encoder

However, there's some debate if Llama is truly open. The pre-training datasets and fine-tuned datasets are not published.

The pretraining was made with the help of Llama 2

Meta has shared that 5% of the pretraining data used on Llama 3 is multilingual and covers 30 languages. This shows that while the model can perform in different languages, it may not be as effective as in English.




decoded words
cat sat on the mat
Tokenizer decoder
next token predictions
10 82 55
LLaMA-3
projection layer assigns probabilities to the next token
Projection Head
processed token vectors
computation on the token vectors
Transformer Block
Feed Forward Block
Attention Block
repeated
number of attention heads
token vector/embedding
embedding dimension
embedding table transforms every token into a vector
Embedding table
context length
tokens
1 10 82
tokenizers compress text into word chunks
LLaMA 3 uses tiktoken with 128k vocab size
Tokenizer encoder
literally just words
The cat sat on the
token embeddings
Transformer Block
Feed Forward Network
Down Projection
SwiGLU activation
Up Projection
Up Projection
RMS Norm
residual connection
Attention Block
Self-Attention
RoPE
Queries
Keys
Values
RMS Norm
residual connection
Self-Attention
Keys & Queries
nearby keys and queries get a big weighting
attention matrix
Values
the values are a scaled sum of the old values according to the weightings
Rotary Positional Encoding
token vectors are rotated according to their position in the input sequence
zero rotation for the first token
RMS Normalization
Scales all token vectors onto the same ellipsoid with a learnable radius
learnable unit circle

10.3.1. LLAMA3 REFERENCES:

Github:

https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

https://github.com/FareedKhan-dev/Building-llama3-from-scratch

Medium:

https://medium.com/@pranjalkhadka/llama-explained-a70e71e706e9

https://medium.com/@ccibeekeoc42/unlocking-low-resource-language-understanding-enhancing-translation-with-llama-3-fine-tuning-df8f1d04d206

Kili:

https://kili-technology.com/large-language-models-llms/llama-3-guide-everything-you-need-to-know-about-meta-s-new-model-and-its-data

Arxiv:

https://arxiv.org/pdf/2303.18223v13

https://arxiv.org/pdf/2407.21783v3

Youtube:

https://www.youtube.com/watch?v=r-heqmMYNL0&ab_channel=KyeGomez

https://www.youtube.com/watch?v=6PMZqOfzjmY&ab_channel=BotsKnowBest

https://www.youtube.com/watch?v=Mn_9W1nCFLo&ab_channel=UmarJamil

https://youtu.be/4Ns6aFYLWEQ

Hasgeek:

https://hasgeek.com/simrathanspal/the-llama3-guide/sub/decoding-llama3-part-5-grouped-query-attention-HcTT9kcU91JaaLi5YdYSpV

https://hasgeek.com/simrathanspal/the-llama3-guide/sub/decoding-llama3-part-4-rotary-positional-embedding-3K8ZHpdLi6E56N8ejnaWzm

Hugging face:

https://huggingface.co/meta-llama/Llama-3.1-8B

Meta:

https://ai.meta.com/blog/meta-llama-3/

Others:

https://devopedia.org/llama-llm

https://stuartfeeser.com/blogs/ai-engineers/llama-model/index.html

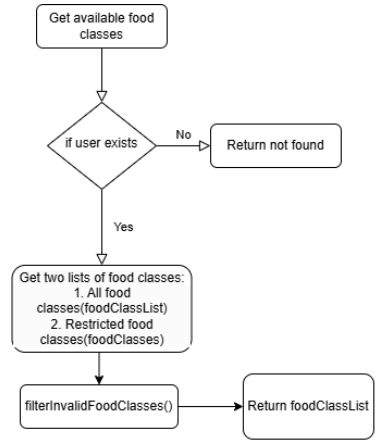
https://jalammar.github.io/illustrated-transformer/

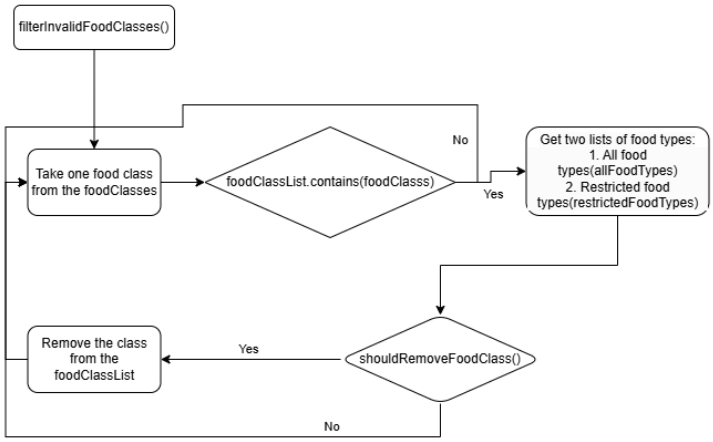
https://tinkerd.net/blog/machine-learning/multi-query-attention/

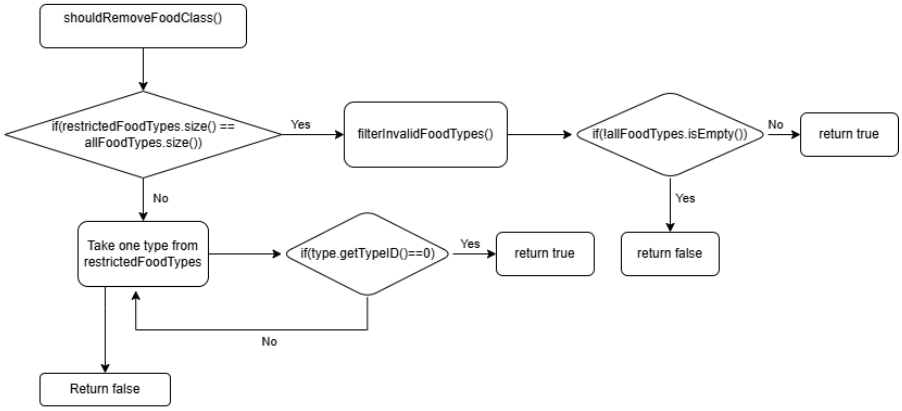
https://www.forbes.com/sites/janakirammsv/2024/04/19/meta-unveils-llama-310-key-facts-about-the-advanced-llm/

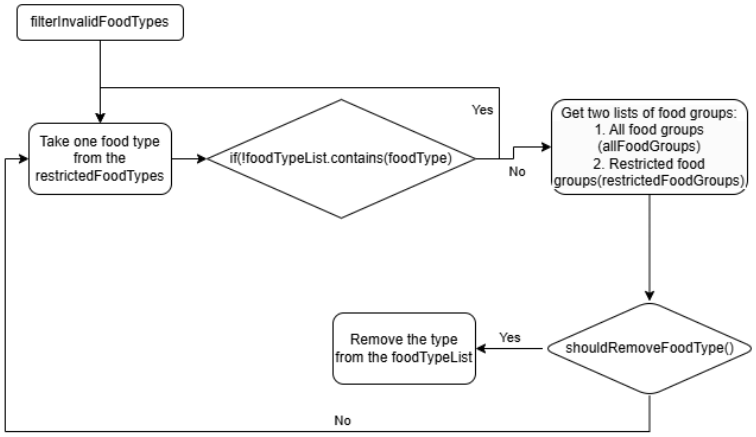
https://ttumiel.com/blog/LLaMA3/

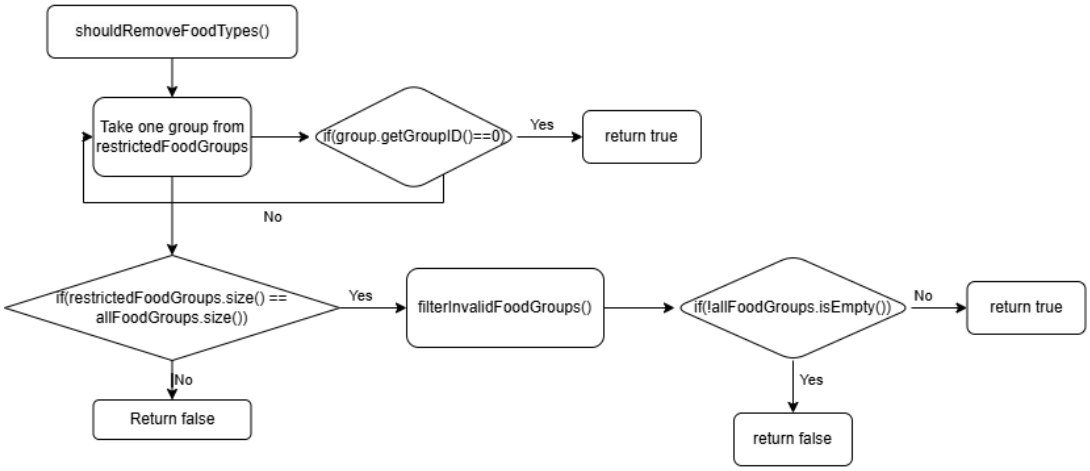
10.4. Backend annexes

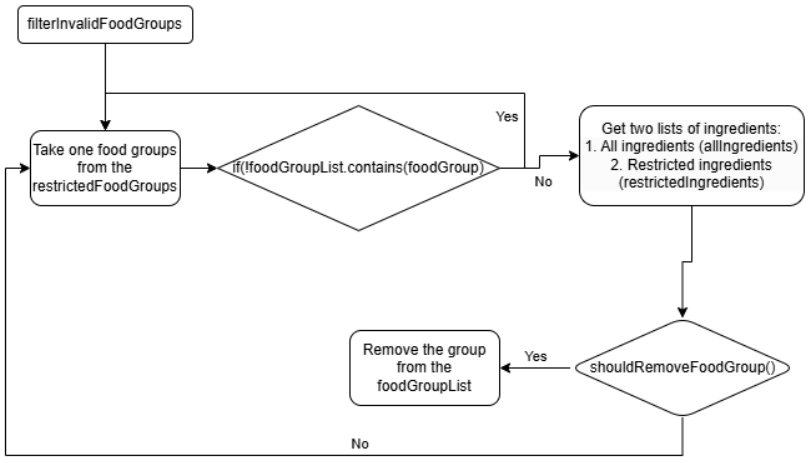


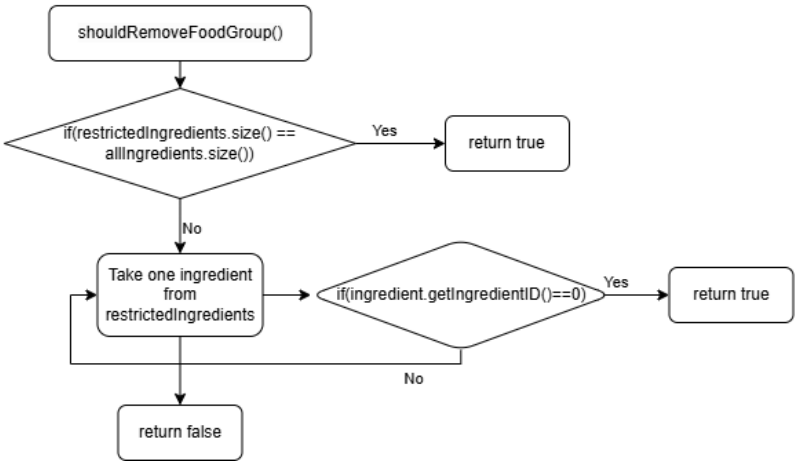





filterInvalidFoodTypes
Take one food type from the restrictedFoodTypes
if(!foodTypeList.contains(foodType)
Yes
No
Get two lists of food groups:
1. All food groups (allFoodGroups)
2. Restricted food groups(restrictedFoodGroups)
shouldRemoveFoodType()
Yes
Remove the type from the foodTypeList
No


shouldRemoveFoodTypes()
Take one group from restrictedFoodGroups
if(group.getGroupID()==0)
Yes
return true
No
if(restrictedFoodGroups.size() == allFoodGroups.size())
Yes
filterInvalidFoodGroups()
if(!allFoodGroups.isEmpty())
No
return true
No
Return false
Yes
return false


filterInvalidFoodGroups
Take one food groups from the restrictedFoodGroups
if(!foodGroupList.contains(foodGroup)
Yes
No
Get two lists of ingredients:
1. All ingredients (allIngredients)
2. Restricted ingredients (restrictedIngredients)
shouldRemoveFoodGroup()
Yes
Remove the group from the foodGroupList
No


shouldRemoveFoodGroup()
if(restrictedIngredients.size() == allIngredients.size())
Yes
return true
No
Take one ingredient from restrictedIngredients
if(ingredient.getIngredientID()==0)
Yes
return true
No
return false

10.5. Thread simulation code parts

```
def add(self, reqData):
    self.spaces.acquire()
    with self.mutex:
        self.bq.put(reqData)
        print(str(reqData[0]) + " ADD QUERY >  " + str(reqData[2]) + "\n")
        if self.bq.full():
            self.modelReady.notify()

def remove(self):
    item = 0

    with self.mutex:
        if self.bq.empty():
            self.spaces.release(5)
            self.modelReady.wait(timeout=10)
        if self.end == True:
            return None

        item = self.bq.get()
        print(str(item[0]) + " TAKE QUERY < " + str(item[2]) + "\n")

        return item
```

```
def add(self, item):
    self.spaces.acquire()
    with self.mutex:
        self.answerDict[(item[0], item[1])] = [item[0], item[1], item[2]]

        print(str(item[0]) + " ADD ANSWER >  " + str(item[2]) + "\n")
        self.checkBuffer.notify_all()

def remove(self, reqData):
    with self.mutex:
        while True:
            popped_message = self.answerDict.pop((reqData[0], reqData[1]), None)

            if popped_message is not None:
                print(str(popped_message[0]) + " TAKE ANSWER < " + str(popped_message[2]))
                self.spaces.release()
                return popped_message

            self.checkBuffer.wait()
```

10.6. Planification annexes

10.6.1. Kanban policies

To ensure proper teamwork with the least amount of misunderstandings in the definition and completion of tasks, to limit the amount of work for each teammate to avoid overburdening the teammate, and for the correct duration and progress of the project.

Seven policies have been defined, each with different objectives. The policies are as follows:

- **Definition of Done**: Defines when a task can be recognised as done.
- **Work in Progress Limits**: This policy limits the amount of work a teammate can do at the same time, avoiding stressful situations related to the project.
- **Criteria for moving tasks between phases**: As the name suggests, criteria for moving a task from one phase to another.
- **Bug handling policy**: Defines how to handle and fix bugs whenever they occur in the project's code.
- **Collaboration and Communication policy**: Defines how team members communicate with each other, share their progress, ask for help with a task, etc.
- **Meeting policies**: Defines when the designed meetings will take place during the project.
- **Expert policy:** These policies define when the team will arrange a meeting with the subjects expert in order to help them with any doubts or task

10.6.1.1. Definition of done

The **Definition of Done** policy ensures that all team members know when a task is ready for completion. Whenever a task responsible thinks that the a kanban card can be recognised as done, he or she will act upon it differently depending on the type of task he or she is working with.

If the task is related to documentation, presentations, videos... overall not technical, that task must present the information which it was created for clearly and in detail.

If the task is technical, the functionality it was created for must work, the rest of the team must be notified of the completion and the changes will need to be pushed to their respective github repositories.

10.6.1.2. Work in Progress (WIP) limits

To avoid stress and saturation among team members and to ensure a proper workflow, the number of tasks in each phase was limited as follows:

- **To Do**: Undefined. The number of tasks added to the backlog will be as much as necessary.

- **Analysis:** Up to 5 tasks to correctly analyse each task and make a correct judgement on its implementation. This includes checking for dependencies, examples and existing implementations
- **Development:** Limit to 3 tasks to avoid overwhelm and ensure work is completed on time. Depending on the progress of the project, the limit can be exceeded in order to finish the project on time
- **Testing**: Limit of 7 tasks to avoid overwhelming testers and reviewers.
- **Deployment**: Limit of 10 tasks per team member to ensure that tasks are completed correctly and tasks do not pile up.
- **Done**: Same as To Do.

The WIP limits can be changed as the project progresses. For example, if the team is running out of time, or if a team member has hit a development roadblock and the team's attention needs to shift.

10.6.1.3. Criteria for moving tasks between phases

This policy has been created to define when a task needs to be moved from one phase to another:

To Do →Analysis:

- The a kanban card has been created for the task
- The hours estimate has been established
- The task has a responsible for completing it

Analysis → Development:

- The task has been properly analized, youtube tutorials have been checked, guides read...
- If the task needs some new dependencies, they have been properly installed
- If no information was found or the info was too simple, the responsible has met with the expert to help clear some doubts

Development → Testing:

- The code works as intended

Testing → Deployment:

- The code has passed the quality gate defined in the code scanning tool (SonarQube).

- The feature has passed the required tests and has reached a test coverage of 80% at least.
- All bugs and vulnerabilities have been removed.
- The team has been notified

Deployment → Done:

- The changes have been added to the main branch of the repository.
- The integration doesn't ruin the main projects functionality, it's integrated naturally.
- The vast majority of the code smells have been removed.

For the non technical tasks, they will only pass through the development phase, as it doesn't make sense to test and deploy a document for example.

10.6.1.4. Bug handling policy

The purpose of this policy is to properly define the attitude towards bugs in the development of the project. Bugs are divided into two categories: critical and minor, each with its own response from the team.

- **Critical Bugs**: If a critical bug is found that prevents the new feature or the whole programme from working, the bug must be fixed as soon as possible. If the team member who found the bug can't fix it, they must report it to the team.
- **Minor Bugs**: Minor bugs should be noted and prioritised for a later phase. The bugs should be fixed in a reasonable amount of time so that they do not accumulate.

10.6.1.5. Communication policy

The purpose of this policy is to define how the team will communicate and work together throughout the project.

- **Oral communication:** When the team is gathered, and whenever something needs to be shared, the team members will obviously talk face to face
- **Online communication:** When the team isn't gathered and communication is needed, the team will communicate via Whatsapp and Google Meet, Whatsapp for more informal communication and Google Meet for meetings and important questions

10.6.1.6. Meeting policy

Last but not least, the meeting policy. This policy has been created to properly define the meetings that'll take place through the duration of the project. Each meeting will have a frequency, participants and topics to work on:

- **Daily meetings**: Hold daily meetings with all of the present team members to indicate what each member did the last day, what will they work on today, and if there's any roadblock or problems that might need the attention of more team members or maybe the expert.
- **Replenishment meeting**: This meetings will take place at the beginning of each week between the team members. In this meetings the tasks that will be worked on in that week will be specified.
- **Service delivery meeting:** This meetings will take place at the end of the week, and the progress of the tasks that were specified in the replenishment meeting will be updated. Depending on the progress of those tasks, the team will need to work extra or increase the intensity of the work in the following week. The team members take part in the meeting.
- **One on One meeting:** This meeting will take place each wednesday between the team and the tutor. The tutor will be notified about the progress of the project and if any meaningful advances are made they will be shown to the tutor.
- **Delivery planning meeting:** This meeting will be done at the beginning of the project and no more. The team will give a rough estimate of when each feature will be finished and delivered.

The following meeting will be held only if necessary, if the progress of the project is too slow or if something happens.

- **Risk review meeting:** This meeting will take place if the progress of the project is dangerously slow, and the decisions taken in it will be related on how the team will work upon then to recover the proper workflow of the project. This meeting can be done extra officially.

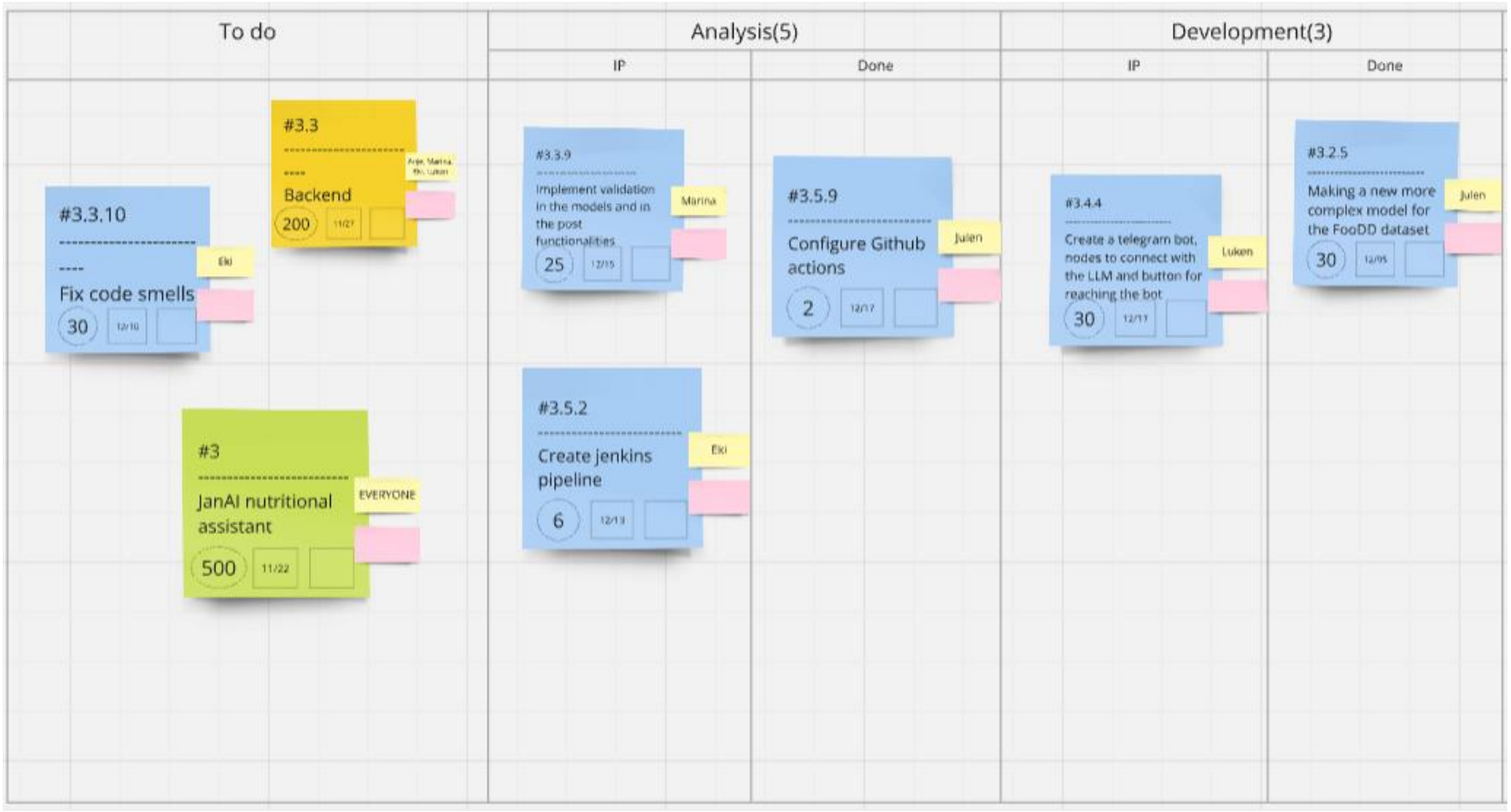
Every meeting must have it's minutes written and uploaded to it's respective google drive folder.

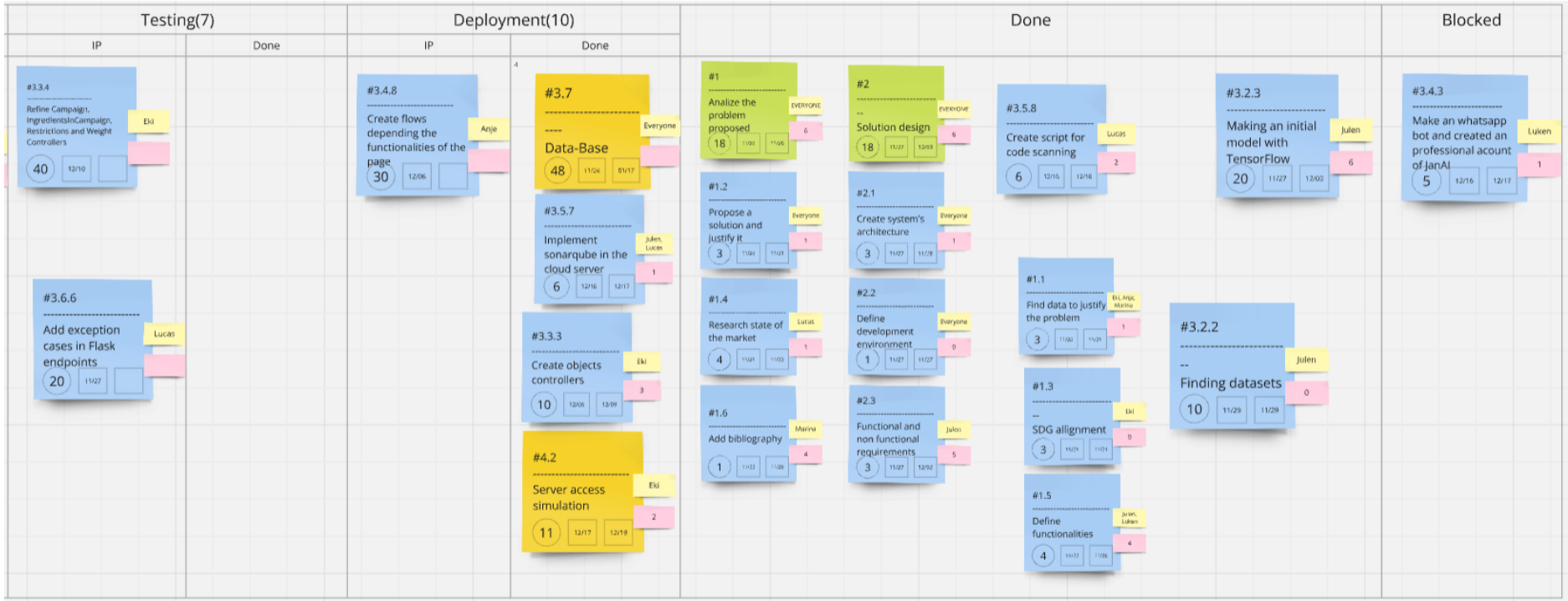
10.6.1.7. Expert policy

Last but not the least, the expert policy. The purpose of this policy is to define when the team will contact the designed expert and arrange a meeting with them. The meetings will be arranged mainly when two things happen:

1. **Blocked tasks:** If a team member is stuck with a task longer than a day, and if no valid solution was found, the expert will be contacted in order to help with it.
2. **Proposal for the project:** When the team defines a new functionality for the solution and it's not specified in the technical rubric or if the team isn't sure that the functionality is valid, the team will arrange a meeting with the expert to clear the doubts.

10.6.2. Kanban board




Testing(7)
IP
Done
Deployment(10)
IP
Done
Done
Blocked
#3.3.4
Refine Campaign, IngredientsInCampaign, Restrictions and Weight Controllers
Eki
40
12/10
#3.6.6
Add exception cases in Flask endpoints
Lucas
20
11/27
#3.4.8
Create flows depending the functionalities of the page
Anje
30
12/06
#3.7
Data-Base
Everyone
48
11/26
01/17
#3.5.7
Implement sonarqube in the cloud server
Julen, Lucas
6
12/16
12/17
1
#3.3.3
Create objects controllers
Eki
10
12/06
12/09
3
#4.2
Server access simulation
Eki
11
12/17
12/19
2
#1
Analize the problem proposed
EVERYONE
18
11/06
6
#1.2
Propose a solution and justify it
Everyone
3
11/21
1
#1.4
Research state of the market
Lucas
4
11/03
1
#1.6
Add bibliography
Marina
1
11/22
11/28
4
#2
Solution design
EVERYONE
18
11/27
12/03
6
#2.1
Create system's architecture
Everyone
3
11/27
11/18
1
#2.2
Define development environment
Everyone
1
11/27
11/27
0
#2.3
Functional and non functional requirements
Julen
3
11/27
12/02
5
#3.5.8
Create script for code scanning
Lucas
6
12/15
12/18
2
#1.1
Find data to justify the problem
Eki, Anje, Marina
3
11/20
11/21
1
#1.3
SDG allignment
Eki
3
11/21
11/21
0
#1.5
Define functionalities
Julen, Lukas
4
11/22
11/26
4
#3.2.3
Making an initial model with TensorFlow
Julen
20
11/27
12/03
6
#3.2.2
Finding datasets
Julen
10
11/29
11/29
0
#3.4.3
Make an whatsapp bot and created an professional acount of JanAI
Luken
5
12/16
12/17
1

10.7. SDG Impact Assessment Tool analysis

SDG IMPACT ASSESSMENT TOOL

Jan Ai

Description

Our project is a meal tracker/portable nutritionist. On the app, users will first introduce their personal info (relevant information to define a diet such as: age, sex, weight and height) and their goal for the future, (losing or gaining weight depending on their state, mantaining the weight...) and based on those preferences our app will recommend different kinds of foof to consume. Apart from that, if a user has any doubts about something the'll eat, they can take a photo and the app will decide whether the food is good or not.

Strategic choices

These are the prioritised areas that we will take action on.

- [x] Positive impacts we can strengthen even further
- [ ] Negative impacts we can eliminate or minimise
- [ ] Knowledge gaps we need to fill

Strategy

Based on the result above, we can see that the project impacts positively, as expected, in four objectives and doesn't have any impact on the rest of them. In addition to that, the project has a positive impact in only one of the goals, and affects the rest indirectly. In a future, the team would like to add more functionalities to improve the existing impact and make the app impact

directly in the rest of the goals, via collaboration with health institutions to develop a more precise AI model to give a more accurate advice, more accurate nutritional values, etc.

NO POVERTY

End poverty in all its forms everywhere

Impact

NO IMPACT

Motivation

No impact

ZERO HUNGER

End hunger, achieve food security and improved nutrition and promote sustainable agriculture

Impact

INDIRECT POSITIVE

Motivation

JanAI hasan impact in the following targets: - Target 2.1: The food recognition can help users making better dietary choices, reducing nutrient defficiencies. - Target 2.2: By offering personalized dietary recommendations and knowing that even if there's access to food, JanAI helps combat malnutrition and promote nutrient-rich food choices for diverse populations.

GOOD HEALTH AND WELL-BEING

Ensure healthy lives and promote well-being for all at all ages

Impact

DIRECT POSITIVE

Motivation

JanAI impacts on the following targets of the goal: - Target 3.4: JanAI helps prevent diseases like diabetes, obesity, and cardiovascular issues by promoting healthy eating habits through it's nutritional chat. - Target 3.8: By providing easily accesible, AI drive nutritional advice, JanAI opens the access to nutritional advice for people that don't have access to a nutritionist - Target 3.D: JanAI empowers individual independence managing a diet, giving advice when needed and reducing the amount of work healthcare systems need to do regarding nutrition

QUALITY EDUCATION

Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all

Impact	Motivation
NO IMPACT	No impact

GENDER EQUALITY

Achieve gender equality and empower all women and girls

Impact	Motivation
NO IMPACT	No impact

CLEAN WATER AND SANITATION

Ensure availability and sustainable management of water and sanitation for all

Impact	Motivation
NO IMPACT	No impact

AFFORDABLE AND CLEAN ENERGY

Ensure access to affordable, reliable, sustainable and modern energy for all

Impact	Motivation
NO IMPACT	No impact

DECENT WORK AND ECONOMIC GROWTH

Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

Impact

NO IMPACT

Motivation

No impact

INDUSTRY, INNOVATION AND INFRASTRUCTURE

Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

Impact

INDIRECT POSITIVE

Motivation

Our project alligns with this objetive in the fact that it encourages technological innovation via image recognition and the use of algorithms to bring diet recommendations. On a more specific level, it alligns with the following target: - Target 9.5: JanAI uses AI and image recognition technologies, improving the innovation in health related technologies

REDUCED INEQUALITIES

Reduce inequality within and among countries

Impact

NO IMPACT

Motivation

No impact

SUSTAINABLE CITIES AND COMMUNITIES

Make cities and human settlements inclusive, safe, resilient and sustainable

Impact

NO IMPACT

Motivation

No impact

RESPONSIBLE PRODUCTION -- AND CONSUMPTION

Ensure sustainable consumption and production patterns

Impact

INDIRECT POSITIVE

Motivation

The app has an impact on the following targets of this goal: - Target 12.3: JanAI provides the needed amount of daily food, helping the users to ration their foods more efficiently, reducing overconsumption and food waste. - Target 12.8: JanAI educates the user on their food's nutritional value, making the users more mindful and more informed when eating

CLIMATE ACTION

Take urgent action to combat climate change and its impacts

Impact

NO IMPACT

Motivation

No impact

LIFE BELOW WATER

Conserve and sustainably use the oceans, seas and marine resources for sustainable development

Impact

NO IMPACT

Motivation

No impact

LIFE ON LAND

Protect, restore and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, and halt and reverse land degradation and halt biodiversity loss

Impact

NO IMPACT

Motivation

No impact

PEACE, JUSTICE AND STRONG -- INSTITUTIONS

Promote peaceful and inclusive societies for sustainable development, provide access to justice for all and build effective, accountable and inclusive institutions at all levels

Impact

NO IMPACT

Motivation

No impact

PARTNERSHIPS FOR THE GOALS

Strengthen the means of implementation and revitalize the global partnership for sustainable development

Impact	Motivation
NO IMPACT	No impact