SYSTEM SPECIFICATION DOCUMENT

1. Introduction

1.1 Purpose of the document

The following document is meant to convey system specification document for "Recipe Suggester Tool", realized as a final project for the course 440MI Machine Learning Operations at University of Trieste.
The document is organized as follows:

1.2 Scope of the product

The product "Recipe Suggester Tool" provides real-time, step-by-step, and easy-to-cook recipes to the user, based on the food recognized in pictures sent by the user himself. This will allow the user to make less friction in deciding what to cook, reducing food waste and increasing user satisfaction and health through balanced diets.
The product will be available as a web-app, accessible from any device with internet connection.
The product will be based on Machine Learning and Large Language Models to provide state-of-the-art solutions to the user, and it will be GDPR-compliant to ensure user data privacy.

1.3 Definitions, acronyms, and abbreviations

- Platform, Webapp, Product, System are all used here
- RST: Recipe Suggester Tool, the name of the product
- ML: Machine Learning
- LLM: Large Language Model
- GDPR: General Data Protection Regulation

2. General description

2.1 Product workflow

Here is presented a general overview of the expected workflow followed by the product. \

1. A user accesses the webapp via a browser and logs in (or registers) into his/her account.
2. The user sends to the webapp a picture of food, e.g., sends a photo of a fridge.
3. The picture is sent to a Machine Learning trained model running in the backend.
4. The different types of food in the picture are recognized by the ML model and stored into an aggregated proper JSON file.
5. The file is sent to the webapp for a user approval. In case something is missing, or something is wrong, the user can modify the file.
6. The corrected file is sent to a Large Language Model (LLM), that will provide a small set of recipes to the user. The LLM is properly initialized and has knowledge of the user's history.
7. Finally, the recipes are sent to the frontend and shown to the user, which can select one of them as his/her own favorite.

2.3 User characteristics

A user, here and after referred to as "the user", is a person in need of advice for deciding what to cook and how. The user should have an internet connection and should be registered to the webapp for using the product.

2.4 General constraints

Some constraints are defined to ensure a sufficient quality of the service provided.

- The whole workflow will require at most 10 seconds + 5 seconds of LLM latency.
- The product should be robust to manage multiple (at least 5) concurrent accesses through the platform.
- The product should be robust to failures, e.g., the connection to the backend is not working. In such cases, a proper error message should be shown to the user.
- The percentage of accesses causing failures should be below 1%.
- Some data is possibly prone to corruption. The percentage should be below 1%.
- The webapp should be unavailable for any maintenance operations less than 2 hours per month.
- The size of the input image should be less than 5MB.

2.5 Assumptions and dependencies

For a proper functioning of the product, some assumptions and dependencies are defined as follows:

- The user has a device with internet connection and a browser.
- The user is registered to the webapp.
- The user is willing to share some data with the webapp, e.g., images of food.
- The user is willing to accept cookies for the webapp.
- The user is willing to accept GDPR terms for data privacy.
- The backend is able to run the ML model and the LLM.
- The ML model is properly trained and able to recognize food in images.
- The LLM is properly initialized and able to provide recipes based on the food recognized.
- The webapp is able to communicate with the backend and viceversa.

3. Requirements

The following section will distinguish between Functional Requirements (code `F0x` ,) and Non-functional requirements (code `N0x`).

3.1 Functional Requirements

User account & Authentication

- `F01.01` : The system should allow users to register a new account using (a) an email and a password and/or (b) using a social authentication via OAuth2 (e.g. : Google, Meta).

- `F01.02` : The system should allow users to log in using (a) an email and a password and/or (b) using social authentication.
- `F01.03` : During the registration process, the system should ask the user to accept Cookies and GDPR privacy terms.
- `F01.04` : During the registration process, the system should ask the user the following information: `Name, Surname, Age, Gender, Country, Allergies` , in order t provide better recipe suggestions.

Image Upload & Pre-processing

- `F02.01` : The system shall allow users to upload an image file via the web interface. The allowed formats are: `.JPEG` , `.JPG` , `.PNG` , `.BMP` , `.WEBP` .
- `F02.02` : The system shall validate the input size. A maximum of 5MB per image is allowed.
- `F02.03` : The system shall display a loading icon or popup to make the user understand what is going on.

Food Recognition & Segmentation

- `F03.01` : The system shall pass the uploaded image to the Food Detection Model, a fine-tuned version of YOLOv8 or following.
- `F03.02` : The system shall be able to recognize food types and generate a list of detected labels. A minimum of 50 different food types should be recognizable.
- `F03.03` : The system shall aggregate the detected labels and their respective confidence scores into a structured format (e.g., `.JSON`).
- `F03.04` : The system should send the generated `.JSON` file to the front-end for user review and confirmation. The corrections allowed are: add/remove food types.

Recipe Generation

- `F04.01` : THe system shall construct a text prompt that includes (1) the system instruction (e.g. "Create three recipes using these ingredients ..."), (2) the list of detected ingredients, (3) anonymized user information, including her/his country and allergies, (4) the past accepted recipes if available.
- `F04.02` : The system shall send the prompt to an external LLM, possibly fine-tuned.
- `F04.03` : The system shall parse the LLM's response to ensure it follows as a structured format, i.e., a `JSON` containing (1) Title, (2) Ingredients List, (3) Cooking Steps, (4) Cooking time for each of the three recipes provided.
- `F04.04` : If the LLM request fails, e.g., because of timeout resources reached, a readable message should be shown to the user.

Output & User Interaction

- `F05.01` : The system shall show the user all the 3 recipes, including all the specifications provided in the structured LLM response.
- `F05.02` : The system shall the user to regenerate the recipes in case of incorrectness in the ones provided.
- `F05.03` : The system shall allow the user to select the most-preferred recipe among the three proposed.
- `F05.04` : The system shall save the preferred recipe among the user's information.

Metadata & Analytics

- `F06.01` : The system shall log the *metadata* of the request (`timestamp` , `number of ingredients detected` , `processing time`) for analytics, without storing the user's image for GDPR compliance.

3.2 Non-Functional Requirements

Usability

- `N01.01` : The web application should be *mobile-first*, ensouring auto-layout from smartphones to desktop monitors.
- `N01.02` : Error messages displayed to the user shall be non-technical and actionable, e.g., a message "No food recognized. Please try again." and a button "Try again" instead of only a message "404: Bad Request".

Performance & Scalability

- `N02.01` : The system shall acknowledge the image upload and display a "Loading" or "Processing" state to the user within 2 seconds over a 5G network.
- `N02.02` : The system shall generate and display the final recipes within 15 seconds for 95% of requests.
- `N02.03` : The system shall support at least five concurrent users performing image analysis simultaneously without service degradation or data collision.
- `N02.04` : The system shall be capable of handling a minimum of 100 requests per day.
- `N02.05` : The system should interrupt the workflow in case of 30 seconds without LLM answer.

Reliability & Availability

- `N03.01` : The system shall be available 99% of the time once deployed.
- `N03.02` : The system shall record and log into a log file all the errors occurring during the whole workflow, from user login errors to final recipe suggestions, for debugging, manual checking, and improvement purposes.
- `N03.03` : The percentage of unhandled internal server errors (HTTP 500) shall be less than 1%.
- `N03.04` : The system shall be able to recover from transient failures, such as temporary network issues or service unavailability, by implementing retry mechanisms (e.g., Docker Swarm/Compose).
- `N03.05` : The system shall degrade by informing the user ("External service unavailable") in the event of an external service failure.

Security & Privacy

- `N04.01` : The system shall **NOT** permanently store user-uploaded images. Images shall be held in volatile memory or temporary storage only for the duration of the transaction and deleted immediately after processing. This both ensures GDPR compliance and reduces database management costs.
- `N04.02` : All the dtaa transmission between the Client (frontend) and the Server (backend) should follow an HTTPS encryption system.
- `N04.03` : The system shall implement a Cookie Consent banner via CookieBot preventing non-essential cookies from loading until user approval.
- `N04.04` : The system shall prevent SQL injections once dealing with SQL queries.
- `N04.05` : No credentials (API keys, Database passwords) shall be hardcoded, but they must be loaded exclusively via Environment Variables in `.env` files or via a Secret Manager.

Software Architecture

- `N05.01` : The backend shall be devloped using Python with FastAPI framework to leverage its native ML libraries compatibility.
- `N05.02` : The frontend shall be developed using TypeScript and a framework like React to ensure type safety and maintainability.

- `N05.03` : The interface between frontend and backend shall be strictly defined using the OpenAPI standard proposed by Swagger.
- `N05.04` : The system shall avoid using static filenames, e.g., `ingredients.json` , for transactional data. All intermediate data must be passed in memory and/or stored using unique identifiers to prevent race conditions between users.
- `N05.05` : In case of disk storage is strictly necessary, uploaded files must be written into a temporary directory `/tmp` and deleted via a `finally` block in the code execution flow.
- `N05.06` : The ML components (YOLOv8 or higher for detection, LLM interface) shall be implemented as decoupled modules/classes, allowing individual components to be swapped or updated without rewriting the code logic.

Data Requirements

- `N06.01` : The system shall maintain a persistent record for each registered user containing the following attributes: `user_id` , a unique and auto-generated primary key, `name` , `surname` , `age` , `gender` as personal infos, `mail` , `password` for the log-in
- `N06.02` : The system shall never store passwords in plain text, but they must be hashed before storage.
- `NO6.03` : The system shall allow a *many-to-many* relationship between users and allergies, e.g. peanuts, lactose, gluten, ..., allowing a user to select multiple restrictions.
- `N06.04` : The system shall allow a *many-to-many* relationship between users and their prefered recipes.

Maintainability & Model Lifecycle

- `N07.01` : The entire application (frontend, backend, database) must be defined in a `docker-compose.yml` file, allowing the system to be sun up on any machine with a single command `docker compose up` .
- `N07.02` : All the code should be versioned via `git` standard and `GitHub` platform.
- `N07.03` : The commits should follow the structure presented in Section 4.3.
- `N07.04` : The fine-tuned model weights shall be saved in a `.pt` format and (eventually) versioned using Data Version Control protocol.

4. Project Architecture

This section is meant to describe the overall architecture of the project, from training to deployment and monitoring; a special focus will be given on the interactions between the components and the MAchine Learning lifecycle management.

4.1 High-level Architecture

The system follows a client-server architecture, where the client is the frontend webapp and the server is the backend API.
Three main components are identified:

1. Client Tier (Frontend): A web application developed using React and TypeScript, responsible for user interaction, image upload, and displaying recipes.
2. Application Tier (Backend): A Python web server (FastAPI) acting as a central orchestrator, managing
    - User Authentication & Session Management

    - Business Logic (Recipes Creation & Management)
    - API Gateway functionality for ML components.
3. Data Tier: Relational MySQL Database are used to store user data, metadata, and pointers to model versions.

4.2 ML Training & Validation

Regarding ML models, only a fine-tuned version of Ultralytics YOLO11n is implemented. \ The LLM for the recipe creation is instead called via API, without any further process.

Data for the fine-tuning of YOLO models has been collected from a variety of datasets available online and shared through the most famous dataset platforms Roboflow.
It has been chosen because it provided ready-to-use models, natively exportable in a YOLO-like format.

- Fridge Detector Dataset - (21042 train, 2150 validation, 923 test)
- Fridgify Dataset - (17691, 1427, 1104)
- Computer Vision Food Dataset - (8754, 796, 574)
- Food Item Detection Dataset (10558, 3003, 1486)

The datasets have been merged and properly preprocessed in order to have a unique dataset with 99 different food classes. \

Fine-tuning has been performed using CUDA-enabled machines on Univeristy of Trieste cluster "Demetra".
The model has been fine-tuned for 100 epochs.
Key metrics tracked include:

- **Box-wise loss**: how well food boxes are identified in the space.
- **Classification loss**: how well food types are classified.
- **mAP@0.5**: mean Average Precision at IoU threshold of 0.5, measuring overall detection accuracy.
- **mAP@0.5:0.95**: mean Average Precision averaged over multiple IoU thresholds from 0.5 to 0.95, providing a comprehensive accuracy measure.
- **Accuracy, Precision, Recall**

All the metrics and the model weights have been logged using pre-defined Ultralytics YOLOv8 functionalities.
The final model weights have been saved in a `.pt` and in a `.onnx` format and versioned using Git LFS.

After fine-tuned, the model has been validated on a separate validation set (10% of the whole dataset) to ensure generalization capabilities, and finally tested on a test set (5% of the whole dataset) to evaluate final performance. \

The final results are the following:

- **mAP@0.5**:
- **mAP@0.5:0.95**:
- **Precision**:
- **Recall**:

Before final deployment, the model had to pass a sequence of tests, implemented to ensure high-quality results:

1. **Correct input**: check for test whether the model is able to deal with the following formats (`.JPEG` , `.PNG` , `.JPG` , `BMP` , `WEBP`)
2. **Black Image**: test with a black image; expects no objects found, so a `None` as output.
3. **Empty Fridge Image**: test with an empty fridge image; expects zero objects found, so a `None` as output.
4. **Blurry Image**: test with a blurry image (adversarial attack-like); expects no objects found, so `None` as output.
5. **Accuracy Detection Test**: test with a common fridge image, for testing model accuracy on finding and correctly classifying items in the fridge. Expects the correct number and the correct classes found in the image as output.
6. **Bounding Test**: test with a common fridge image, for testing whether the model is correctly able to define a suitable (as strict as possible) area around each food recognized. Expects the correct coordinates for the limits of the rectangle (upper-left corner and lower-right corner) as output.

4.3 Deployment

Regarding ML component, deployment is pretty easy, since we are actually only saving a `.pt` .\

Considering instead the whole web-app component, a Docker Image is the final output. Before the deployment of each component, also the webapp needs to pass a list of tests, for ensuring compatibility and full functionality:

1. **Frontend Working Test**: simple checks for ensuring that the frontend is able to start without errors.
2. **Frontend Type Check**: checks type errors in the TypeScript code.
3. **Backend Working Test**: simple checks for ensuring that the backend is able to start without errors.
4. **Docker Build Test**: tests whether each component is dockerizable, i.e., the `Dockerfile` s are correct and working.

Once passed the tests, each component is dockerized using proper `Dockerfile` s; the webapp is then pushed to GitHub Container Registry (GHCR) for versioning and easy deployment, so that one can easily pull the latest version directly in his local machine. \

4.4 Monitoring

To ensure the system's reliability and performance post-deployment, a comprehensive monitoring strategy will be implemented. This includes:

1. **Application Performance Monitoring (APM)**: Grafana will be used to monitor server performance, response times, and error rates.
2. **Log Management**: Centralized logging using ELK Stack (Elasticsearch, Logstash, Kibana) to collect and analyze logs from both frontend and backend components.
3. **User Analytics**: Integration with Google Analytics to track user interactions, session durations, and drop-off points within the web application.
4. **Model Performance Monitoring**: Regular evaluation of the ML model's performance using a subset of user-uploaded images (with user consent) to detect any degradation in accuracy or response time.

5. Risk Analysis

This section identifies potential risks associated with the development and operation of the Recipe Suggester Tool. Risks are categorized by their nature (Technical, Machine Learning, or Privacy) and assessed based on their **Likelihood (L)** and **Impact (I)**.

Legend:

- **L (Likelihood):** 1 (Rare) to 5 (Almost Certain)
- **I (Impact):** 1 (Negligible) to 5 (Catastrophic)
- **Risk Level (R):** L × I (Low: 1-9, Medium: 10-19, High: 20-25)

5.1 Technical & Infrastructure Risks

ID	Risk Description	L	I	R	Mitigation Strategy
R01	**External API Unavailability** The LLM provider (e.g., OpenAI) or the Auth provider (Google/Meta) goes offline or times out.	3	5	**15 (Med)**	Implement a **Circuit Breaker** pattern. If the LLM times out, fail gracefully with a user-friendly message ("Our chefs are busy"). Implement exponential backoff retries.
R02	**Latency Constraint Violation** The total response time exceeds the 15s threshold due to high traffic or LLM generation slowness.	4	3	**12 (Med)**	Use a faster, smaller LLM model (e.g., GPT-4o-mini) for standard requests. Implement **Redis Caching** for common ingredient combinations to skip generation entirely.
R03	**Database Data Loss** User profiles or saved recipes are lost due to storage corruption.	1	5	**5 (Low)**	Enable automated daily backups (snapshots) of the MySQL volume. Use a managed database service (e.g., AWS RDS) if deployed in production.

5.2 Machine Learning & MLOps Risks

ID	Risk Description	L	I	R	Mitigation Strategy
R04	**Model Hallucination (LLM)** The LLM generates a recipe using ingredients *not* present in the user's fridge or creates unsafe cooking instructions.	3	4	**12 (Med)**	**Strict Prompt Engineering:** Use "System Prompts" that explicitly forbid adding outside ingredients. Set LLM `temperature` to 0.2 to reduce creativity/randomness.
R05	**Data Drift (YOLO)** Real-world user photos (dark, blurry, weird angles) differ significantly from the high-	4	4	**16 (High)**	**Continuous Monitoring:** Log the confidence scores of detections. If the average confidence drops below 60%, trigger an alert for re-

ID	Risk Description	L	I	R	Mitigation Strategy
	quality training datasets, causing accuracy to drop over time.				training. Collect user feedback (corrections) to build a "Fine-tuning Dataset".
R06	**Bias & Sensitivity** The model suggests meat recipes to a vegan user or fails to respect listed allergies.	2	5	**10 (Med)**	Inject user constraints (Allergies, Diet) directly into the **System Prompt** as strict rules (e.g., "CRITICAL: User is Vegan. Do not include meat.").
R07	**Adversarial Attacks** Users upload malicious images (noise patterns) designed to crash the inference server or consume excessive resources.	2	3	**6 (Low)**	Validate input image headers and resize/normalize all images to a standard resolution (e.g., 640x640) before passing them to the model.

5.3 Privacy & Compliance Risks (GDPR)

ID	Risk Description	L	I	R	Mitigation Strategy
R08	**Sensitive Data Leak (Images)** Personal photos (e.g., selfies in the fridge reflection) are accidentally stored or leaked.	2	5	**10 (Med)**	**Stateless Architecture:** Images are processed in RAM and strictly deleted after the transaction. No permanent storage of raw images is allowed.
R09	**API Key Exposure** The system's or user's API keys are exposed in logs or GitHub commits.	3	5	**15 (Med)**	Use `.env` files and `git-crypt`. Configure the logger to mask sensitive fields (e.g., `sk-proj-***`) before writing to the log files.

6. Appendices & Resources