Recipe Generator

Eduard Ruiz (s3687988) & Vicent Descals (s3736407)

December 11, 2023

1 System Requirements

- **Programming Languages:** Python, JavaScript, HTML, and CSS.
- **Libraries and Frameworks:**
  - **Models:** PyTorch, Transformers (HuggingFace), Diffusers (HuggingFace), MMDetection.
  - **Backend:** Django.
  - **Frontend:** Vue.js and Tailwind CSS.
- **Computational resources:** To run these models, we have tested them with an NVIDIA 3080 GPU with 10 GB of V-RAM and 48 GB of RAM.

NOTE: All the Python dependencies can be found on the *requirements.txt* and all the sub-modules for the frontend are listed on the *package.json*.

2 Visual documentation

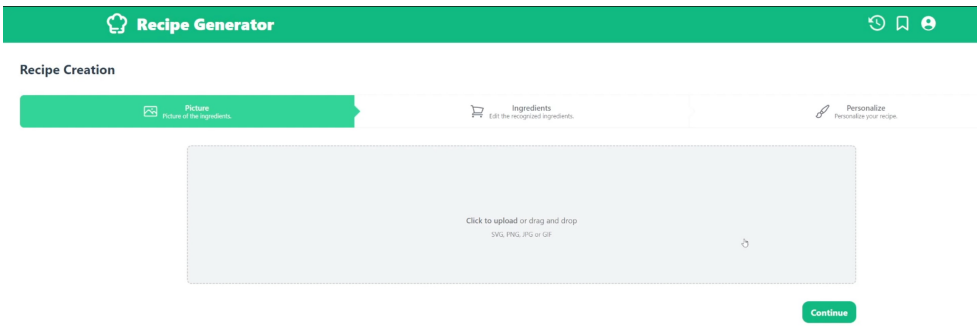

Figure 1: Home page to upload images that you will use in your recipe. Ingredients will be detected from images by Grounding Dino.

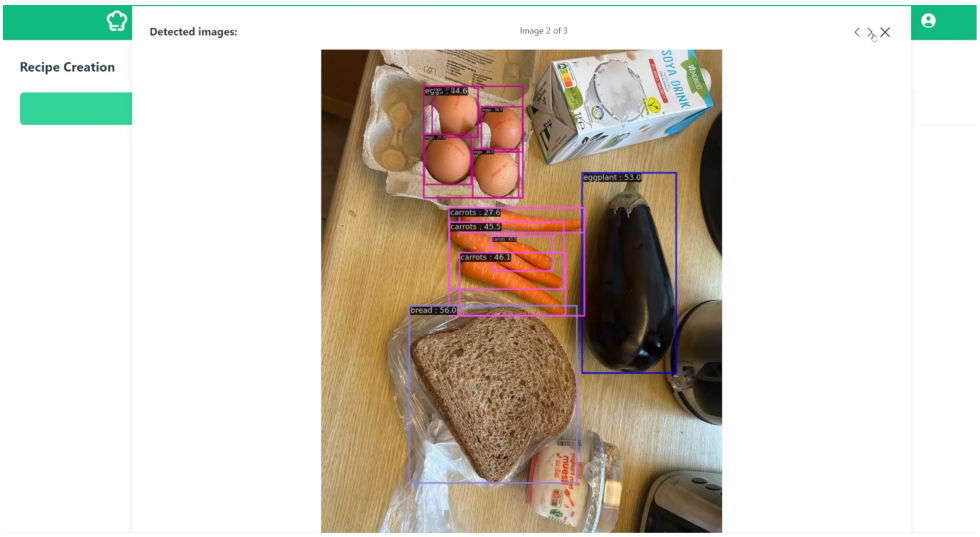

Figure 2: Detected ingredients on uploaded images by the Grounding Dino model. Available to the user to understand where the detected ingredients come from.

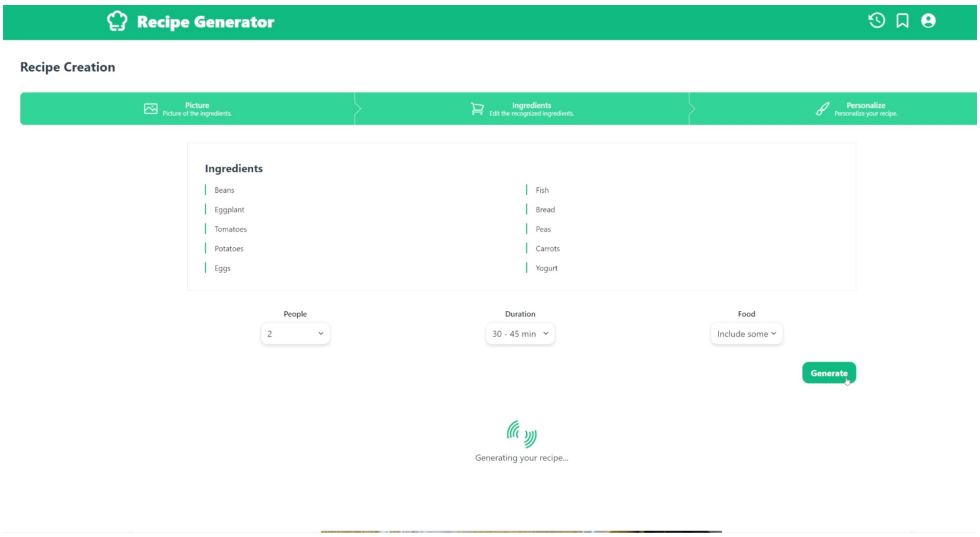

Figure 3: List of all detected ingredients. The screen include personalization selectors to select the number of servings, duration of the recipe and which ingredients to include.

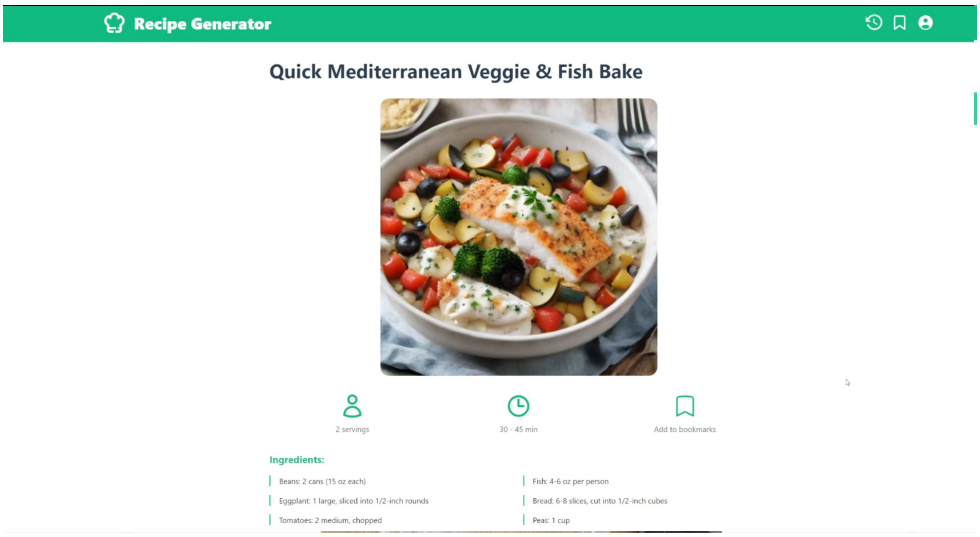

Figure 4: Generated recipe view, with the title, ingredients and its amounts, the generated text and the image generated by the Stable Diffusion model.

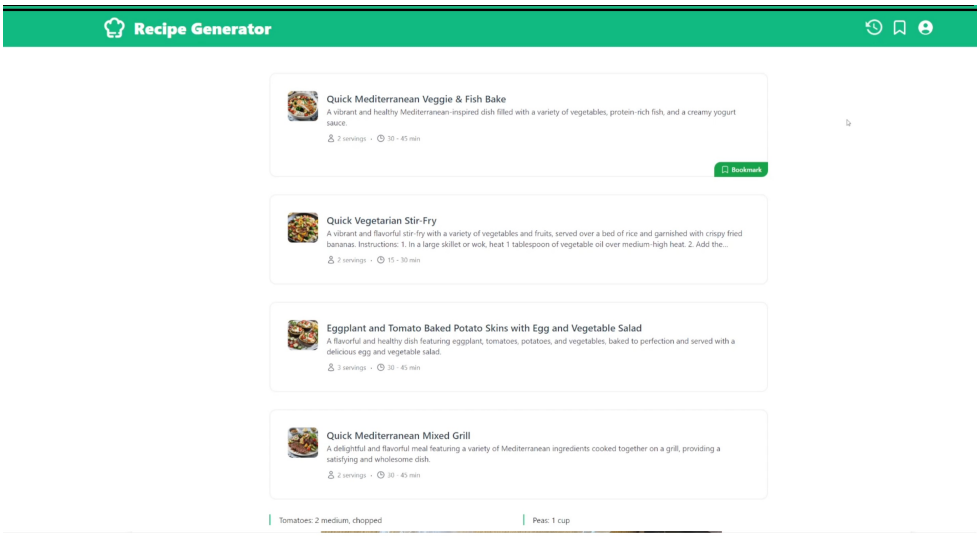

Figure 5: Historical list of generated recipes with bookmarked recipes highlighted. There is an option to filter only the bookmarked recipes.

3 Source code

In our project, we have written all the code, ensuring it meets our specific requirements. While we have developed the core programming ourselves, we have integrated open-source models, utilizing their pre-trained weights. This approach results in a combination of both systems which result more effective. The result is a unique web application which combines open-source models with software development techniques, allowing us to leverage the strengths of existing technologies while adding our own unique functionalities.

Our source code is available in a GitHub public repository: https://github.com/eduruiz00/mms-project.git.

4 System functionalities

Overview

The system is designed to leverage advanced Artificial Intelligence models to facilitate a comprehensive and interactive culinary experience, by generating recipes. It integrates several functionalities, focusing on image recognition, recipe generation, image generation, and recipe storage. The combination of all individual components and their interplay within the system results in the final web application, which produces a recipe based on ingredients detected from images.

Image Recognition and Ingredient Identification (Grounding Dino)

At the forefront of the system's functionality is an image recognition module, referred to as "Grounding Dino". This module utilizes a state-of-the-art machine learning algorithm to detect objects in images. In our case, we provided certain ingredient labels that we are interested in detecting. Upon processing an image, it can accurately identify and list the ingredients present. This process involves deep neural networks trained on a vast dataset of images (including food items), allowing the system to discern a wide array of ingredients with high precision.

Other models were also tested, such as YoLo, with less accuracy.

In case one or many ingredients were wrongly detected or are missing, the list can easily be edited by removing ingredients or adding extra ingredients.

Interactive Recipe Suggestion (Llama2)

Following ingredient identification, the system engages the "Llama2" module. This component is responsible for suggesting recipes based on the identified ingredients. It operates on an extensive database of recipes and employs algorithms to match the available ingredients with potential recipes. This module is designed to adapt to user preferences (number of servings, duration, or ingredients to include), making the recipe suggestions highly personalized and relevant.

Advanced Recipe Generation

Upon selecting a base recipe or a set of ingredients, the system activates its recipe generation functionality. This involves the use of "Llama2", an open-source language model, which crafts detailed recipes. The model considers the ingredient list, and personalized criterion, generating a step-by-step recipe guide together with the required ingredients and their amounts. This process ensures that the recipes are not only feasible with the given ingredients but also align with culinary best practices.

Enhanced Visualization (Stable Diffusion)

To augment user experience, the system incorporates the "Stable Diffusion" module for recipe visualization. This feature transforms the text-based recipe description (generated previously by Llama2) into a more engaging, visually rich format. It employs advanced graphic rendering techniques to create images that represent the final dish, providing users with a visual expectation of the prepared meal.

Recipe Archiving: Historical and Bookmarked Recipes

A key aspect of the system is its ability to archive previously generated recipes. This functionality includes both historical recipe tracking and bookmarking capabilities. Users can access their past recipes in a historic list. With the bookmarking feature, it enables users to save favorite recipes for easy retrieval. This archival system is designed to be intuitive and user-friendly, encouraging users to explore and revisit their culinary experiences.

Conclusion

In summary, this system represents a comprehensive integration of AI-driven modules tailored for an enhanced culinary experience. From ingredient identification through image recognition to personalized recipe generation and visual representation, each component works synergistically. The system's ability to archive recipes adds an additional layer of user engagement, making it a versatile tool for both novice cooks and culinary enthusiasts.

Exclusions from the System

This system does not incorporate a professional culinary checker. Given the integration of multiple modules, occasional misinterpretation of data by one module may lead to inaccuracies in ingredient quantities, instructions, or generate images unrelated to the intended recipe. As AI systems are susceptible to errors, users are advised to review and validate the recipe before proceeding to cook, identifying and rectifying any potential significant discrepancies.

Additionally, the system does not automatically accommodate culinary restrictions (such as vegetarian or vegan preferences), potentially resulting in inconvenient recipes. Nevertheless, users can manually edit the ingredients to align with specific culinary requirements.

Furthermore, the current system lacks the functionality to share recipes with other users. Incorporating this feature in future iterations would enhance user experience by facilitating the exchange of cooking experiences and fostering a community around the platform.

5 Main challenges

During the project, we encountered several main challenges that we had to overcome:

- Finding a suitable object detection model that could accurately detect and label ingredients. Since some ingredients were packaged, detecting the packaging added complexity and decreased the system's accuracy. We initially evaluated YoLo, but the results were not as good as expected. Ultimately, we found another system called Grounding Dino, which performed better. Grounding Dino is trained for object detection tasks in general, including detecting food.

- Finding a suitable GPT model for recipe generation. We considered different options, but Llama2 was the only one suitable for our project because it is open-source. Although GPT-4 is more accurate and allows API calls, it is not open-source and was not within our budget, with the idea of developing a zero-cost system.

- Considering an image generation system to provide users with an idea of how the recipe should look. We ultimately selected the Stable Diffusion model, which is also open-source and generates high-quality images.

- Designing a user-friendly interface that allows users to combine the Machine Learning models and view results dynamically. We achieved this using the Vue.js framework, which served as a single-page application that made HTTP requests to the backend API.

- Developing the backend and database for the web application. We needed a system that could call the models and return their responses. The easiest way to achieve this was with a Python backend, as it allowed for better integration with the models. We chose Django as the REST API framework.

This backend was connected to a SQLite database that stored information that we would need to retrieve, such as user data, generated recipes, and bookmarks.

- Once we had all the components in place - the ML models, the front-end, and the back-end - we needed to connect them all. We accomplished this through a REST API system, where the front-end made HTTP requests to the back-end, which then executed the models for inference and returned the output after some post-processing steps. The end result is a user-friendly web application.

- Designing a consistent prompt for the GPT and Text-to-Image system. To ensure a consistent structure for the output of the GPT model, we developed a prompt that requested output in a specific structure for proper interpretation. Additionally, the GPT model generated a brief description of the recipe, which was then passed to the Text-to-Image system (Stable Diffusion) to generate an image based on that description.

6 Application Setup Process

In this section, we detail the setup process for our application, optimized for use on a computer equipped with GPUs, such as a Windows machine with an RTX-3080 GPU with 10GB of VRAM. The primary software prerequisites include Node.js, Python (version 3.11.5 or later), Conda for environment management, and PyTorch with CUDA support. These components are fundamental to the application's architecture and can be downloaded from their respective official websites.

The setup involves configuring both the back-end and front-end. For the back-end, navigate to the dedicated directory and install the required Python dependencies from the 'requirements.txt' file. This step includes manual installations of the Auto-GPT for the Llama2 model and the *mmdetection* library, essential for the application's functionality. In the front-end setup, move to the corresponding directory and execute `npm install` to integrate the Node.js dependencies listed in the 'package.json' file.

Following the environment setup, the next phase is to prepare the models. This involves downloading pre-trained weights, specifically for the Grounding Dino model, and placing them in the designated directory within the back-end. The weights of the rest of the models will be downloaded directly when executing the application.

To run the application, start the front-end and back-end services. Initiate the front-end by opening a terminal window, navigating to its directory, and executing `npm run serve`, which provides a URL, usually hosted on the localhost. Concurrently, for the back-end, open another terminal window, navigate to the back-end folder, activate the Conda environment, and launch the Django API with `python manage.py runserver`. This process also generates a URL, signaling the back-end service is active.

Accessing the web application is done through the front-end URL in a web browser. If not redirected automatically, manually go to the login page by appending `/login` to the localhost URL. The application uses an admin account for initial login, with 'admin' as the username and '1234' as the password. After logging in, the user is directed to the initial page for uploading images and generating recipes. The application also features a section for accessing recipe history and bookmarks on the top navigation bar, enhancing user interaction.

This guide assumes a basic understanding of terminal operations and directory navigation. For setup issues, refer to the documentation for Node.js, Python, Conda, the operating system in use or the Django documentation.