Gift Idea Generator

Gifting made easy
Graded Miniproject

AI Foundations

Fabio Zahner & Silvan Lendi

Eastern Switzerland University of Applied Sciences

1 Introduction

Introduction

In this document we will describe the process of implementing the Miniproject required for the AI-Foundaitions module. It is expected to generate at least 3 ideas that make use of the GPT-assistant and build an application around that idea.

The first step will describe the generation and thinking process for the ideas and the selection of one idea based on evaluaiton. We will then continue on refining the idea and create specifications that the applcation must implement.
additional text

additional text

We wanted to implement an idea that could actually have a practical usecase and one that we would personally use. We did some brainstorming together and came up with things that we struggle with regularly and where a Large Language Model could be of help. Within a short amount of time we came up with three ideas:

- Food recipe generator that returns recipes based on what kind of food you have at home (would be the user input)
- Since Christmas is arriving soon: gift idea generator based on what properties the gift receiver has
- Belt balancer generator for the game Factorio [1]

Immediately, we thought about the practicality and implementation side of things. All three of the ideas were well usable, however one stood out on the complexity and possibilty of implementation. A belt balancer in factorio serves the use of distributing X number of input belts or conveyors (that transport materials) to Y number of outputs. In the game, you can import structures such as a belt balancer in the form of a blueprint string. This string looks like glibberish to the human eye because the game first decodes the string using base64 and afterwards uses zlib inflate to finally get the json representation of the individual strucutres that will be placed in the game to complete the balancer. To many, this will sound like an application where a Large Language Model will not work well to generate these complex and very error intolerant string, and you are correct! We generated such a string 10 times and not once did the string import work in the game itself. This idea was not going to work well so it is eliminated.

We decided on the following criterias for evaluating which idea to choose:

- practical use (weight 30)
- ease of implementation (weight 5)
- beneficial (weight 20)
- originality (weight 10)

idea	practical use	ease of implementation	beneficial	originality	total score
Recipe generator	5 * **30**	8 * **5**	7 * **20**	5 * **10**	**400**
Gift Idea Generator	7 * **30**	8 * **5**	9 * **20**	8 * **10**	**510**

Since the Gift Idea Generator received more points, this is what we chose for the project.

Before starting to implement the idea, we first decided on specifications that the application schould meet. We discussed ideas and boundries and agreed on the following: The application should take three parameters as inputs from the user: Gender, Age and personal interests It should be available on mobile to be easaly useable when you are on your way out. When the user presses the button to generate the ideas upon entering the parameters, he should receive 5 gifting ideas in text form.

[1] Not affiliated with the product however you should try it anyway

2

Implementation: Backend

To integrate ChatGPT into our application, we utilized the OpenAI API to set up a specialized "assistant" named “fabio_silvan_giftassistant”. This assistant is specifically to generate gift ideas based on user-provided preferences and constraints that come from the Frontend. By configuring system instructions tailored to our requirements, we ensured that it would effectively parse input constraints and consistently return a list of five gift suggestions.

For the model, we selected gpt-4-turbo due to its compatibility with JSON Schema formats, which enables structured outputs and ensures an easy integration with our application’s data parsing processes. Using JSON Schema has proven especially helpful in guiding the assistant's response structure, making the parsing of returned messages easier.

While additional tools like file search, code interpreter, and external function calling are available, we found them unnecessary for this project. The assistant's core design and clear output schema provide all the functionality needed to meet our specific gift recommendation requirements.

3

Implementation: Frontend

To create an application that integrates into daily life, we will use the Flutter technology stack to develop software accessible across multiple platforms. For development, we have focused on optimizing the application for Android mobile phones. However, if we decide to expand the application to other platforms, it would require minimal additional work.

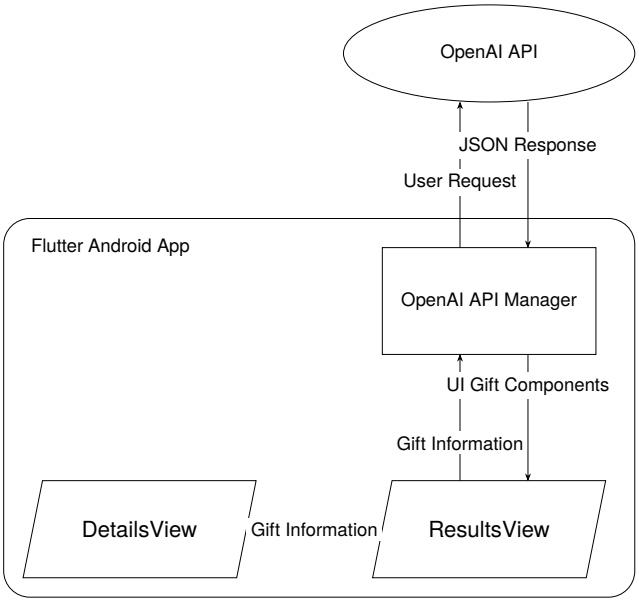


Figure 3.1: Planned architecture of the App

Figure 3.1 describes the planned architecture of the Flutter App. It has two main screens, which the User is guided through:

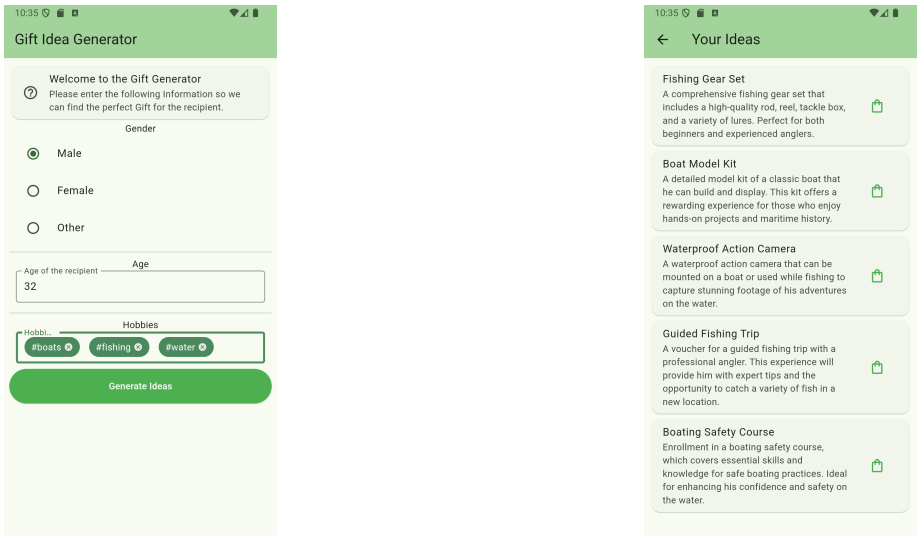
1. In the `DetailsView`, the User can specify different attributes about the Person they would like to gift something to (furthermore called the Recipient)
2. As soon as the User submits the Information, all Gift Information is Passed to the `ResultsView`.
3. The `ResultsView` passes the received Information to the API Manager, which converts it into a prompt, which is then sent to the OpenAI API.
4. The OpenAI Assistant processes the Request and returns a Response in Form of a JSON Object. (For more details see Section **??**)
5. The API Manager receives the JSON Object, extracts the relevant Information and sends it to the `ResultsView` in Form of prebuilt widgets, where they are displayed to the User.

User Interface

For the Design, we wanted a simple but clear User Interface. We used the Material Design Guidelines to ensure a consistent Design. At first, we created a basic UI to test the app's core functionality. After developing the initial version of the app, we proceeded to validate the API Manager's ability to convert JSON responses into visually appealing Widgets.

Once the functionality had been confirmed, we focused on refining the User Interface. While not the primary objective of our project, we recognized that an improved UI can significantly enhance the overall user experience. We therefore implemented the following changes:

- A small infobox which instructs potential first-time users.
- Improved display of input fields for age and hobbies.
- The option to specify the Gender when selecting "Other".
- Added titles for each Gift Idea to improve overview.
- A button to instantly search for a gift on `galaxus.ch`



(a) `DetailsView` **(b)** `ResultsView`

Figure 3.2: Final Version of the Application

OpenAI API Manager

The `OpenAI API Manager` is the backend logic which allows the Application to communicate with the OpenAI Assistant. It provides the functionality to convert User Input into a usable text prompt, as well as convert the received JSON into a user-friendly interface.

```
    factory GiftResponse.fromJson(Map<String, dynamic> data) {
        GiftResponse gr = GiftResponse();
        for (Map<String, dynamic> idea in data['presents']) { // Extract "present" objects
          gr.ideas.add(Idea(idea["title"], idea['description'])); // Fill information into new Idea
              () Object
        }
        return gr; // Return a GiftResponse() Object containing Ideas()
    }


    List<Widget> cards(Set<Idea> ideas) {
        List<Widget> cards = List.empty(growable: true);
        for (Idea idea in ideas) { // For all Ideas in the List
            cards.add(Card( // Add a new widget (Flutter Card Widget in this Case)
            child: ListTile(
                title: Text(idea.title), // Title
                subtitle: Text(idea.description), // Description
                trailing: IconButton( // Shopping Bag Button
                    onPressed: () {
                        _launchUrl(
                            "https://www.galaxus.ch/de/search?searchSectors=0&q=${idea.title}");
                    },
                    icon: Icon(Icons.shopping_bag_outlined, color: Colors.green,))),
            ));
        }
        return cards;
    }
```

Figure 3.3: Code section responsible for converting JSON into Widgets

4

Conclusion

In this project, we successfully developed a Gift Idea Generator, demonstrating the practical application of Large Language Models in solving real-world challenges. By leveraging the OpenAI Assistant through a Flutter-based mobile application, we created a tool that simplifies the often challenging task of finding the perfect gift.

During Developement, we encountered the following challanges:

- Prompt engineering: Creating effective prompts to generate the desired JSON
- JSON parsing: Handling and processing JSON data received from the API, with Error handling in case returned JSON is not valid.
- User interface design: Creating an intuitive and visually appealing user interface.
- API integration: Integrating the OpenAI API into the Flutter app.

While we successfully developed a functional Gift Idea Generator, we recognize that we didn't fully exploit the potential of the OpenAI Assistant. More advanced features like Code Interpreters or external API integrations could have significantly enhanced the app's capabilities. Although, given the inherent complexity of AI integration in this relatively straightforward use case, we are satisfied with the current outcome.

Future Work

While our current implementation focuses on Android mobile platforms, the Flutter framework allows for easy future expansion to other platforms. Potential future improvements could include:

1. Expanding the range of input parameters
2. Implementing more sophisticated filtering of gift ideas
3. Adding user feedback functionality to improve recommendation accuracy
4. Supporting multiple languages

A

Source Code Appendix

```
{
    "name": "present_response_schema",
    "strict": true,
    "schema": {
    "properties": {
        "presents": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "description": {
                        "type": "string",
                        "description": "A detailed description of the present."
                    }
                },
                "required": [
                "description"
                ],
                "additionalProperties": false
            }
        }
    },
    "additionalProperties": false,
    "required": [
        "presents"
    ],
    "type": "object"
    }
}
```

Figure A.1: json schema