Software requirements specification сервиса "Ленивый поварёнок"

Введение

Ленивый поварёнок — это сервис для подпора рецептов блюд по имеющемуся списку продуктов. Сервис состоит из мобильного приложения на android для взаимодействия с пользователем и серверной части на базе микросервисной архитектуры. Особенностью сервиса должна стать возможность составлять список имеющихся продуктов не только с помощью текстового ввода, но и посредством фотографирования продуктов для их автоматического распознания.

Цель

Цель проекта в том, чтобы упростить мучительный процесс выбора блюда для приготовления, предложив людям сервис для подбора блюд. Требуется, чтобы пользователь имел возможность перечислить имеющиеся продукты с помощью выбора из огромного списка, по которому будет выполняться текстовый поиск, или, возможно, с помощью распознавания продуктов на фотографии. Далее у пользователя должна быть возможность нажать на кнопку, отправящую его запрос на сервер, и получить в качестве результата список "релевантных" блюд с рецептами. Формальное определение релевантности не зафиксировано, неформально это должны быть блюда, чей список ингридиентов в некотором смысле сильно пересекается со списком имеющихся у пользователя ингредиентов.

Соглашения о документировании

Оговоримся сразу, в данном сервисе очень много степеней свободы: существует много удобных и полезных функций, которые хотелось бы иметь, но которые не описаны в данном документе, а во многих местах можно применить более "умные" алгоритмы чем те, которые предлагается реализовать в данном документе. Однако чтобы не закопаться и не остаться ни с чем предлагается реализовывать продукт итеративно. Все перечисленные здесь детали реализации не обязаны во всех будущих версиях продукта быть реализованы и реализованы именно приведённым образом, однако данный SRS документ — это описание версии, которую предлагается реализовать в данный момент. Далее о данном подходе мы будем говорить "текущая итерация".

- Пользовательское приложение — приложение, с которым взаимодействует пользователь. Вообще говоря в текущей итерации это только android приложение, и разработка других способов взаимодействия с пользователем пока даже не планируется. Но для абстракции пользовательской части от серверных микросервисов может использоваться это слово, так как микросервисам не важно, с чем именно взаимодействует пользователь: android-приложение, ios-приложение, web-сайт, настольная программа,...

Ссылки

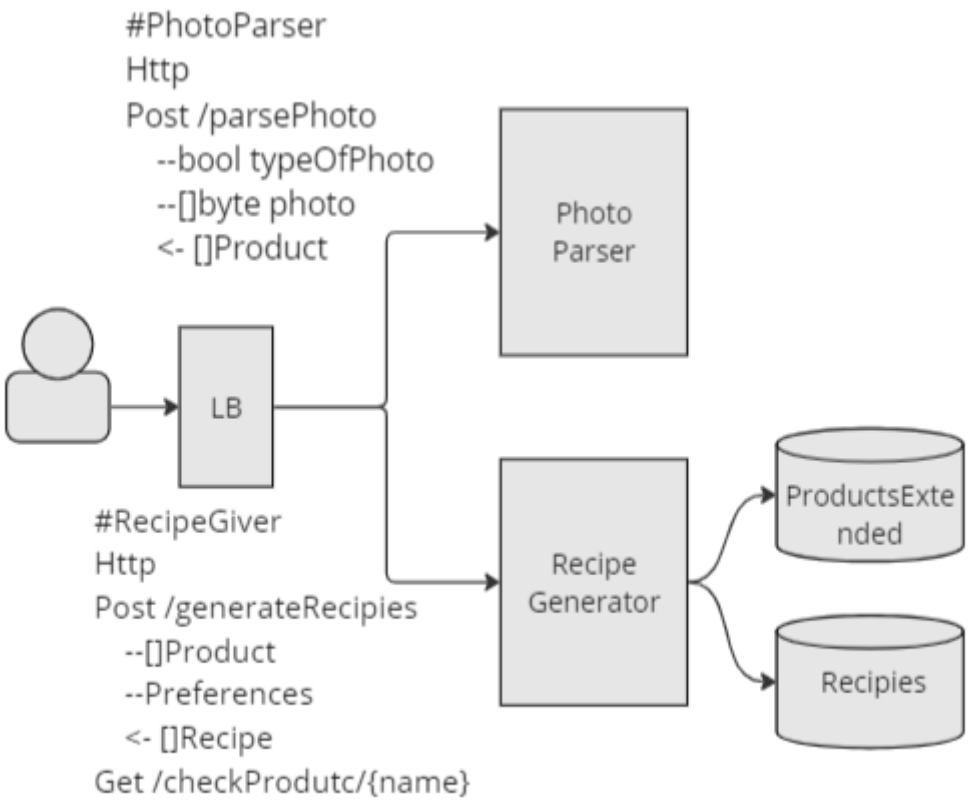
- База данных продуктов и рецептов, вариант 1: https://ru.openfoodfacts.org/data. Преимущества: много продуктов, рецептов и сопровождающих данных (интересные свойства блюд). Недостатки: нужно импортировать данные из MongoDB.
- База данных продуктов и рецептов, вариант 2: https://www.kaggle.com/datasets/coolonce/recipes-and-interpretation-dim Преимущества: довольно большая база, легко импортируется.

Общее описание

Функциональность продукта

Продукт должен состоять из нескольких взаимодействующих друг с другом модулей:

- Клиентское android-приложение
- Микросервис подбора рецептов (RecipeGenerator)
- Микросервис для работы с фотографиями продуктов питания (PhotoParser)
- Gateway-балансировщик



Перспективы развития продукта

В отсутствие аналогов среди android-приложений, продукт может занять свою нишу на рынке. Для этого продукт должен быть:

- Максимально простым в использовании пользователем.
- Легко масштабируемым.
- Открытым к развитию.

Фокус группы

Предполагаемый потребитель является русскоговорящим. Выделяются две большие группы пользователей:

- Желающий научиться готовить человек (неопытный повар).
- Умеющий готовить, но желающий разнообразить свой рацион человек (опытный повар).

Окружение системы

Клиентская часть продукта будет располагаться на мобильном-android устройстве потребителя. Доставка будет осуществляться через Google Play.

Серверная часть, а также база данных будут расположена на виртуальных машинах в Яндекс.Облаке. Операционная система Ubuntu 20.

Системные функции

Системная функция — `PhotoParser.parseProducts`, опознание многих продуктов на фото

Описание

front-end отправляет фотографию набора продуктов питания на endpoint микросервиса PhotoParser, который с помощью модели детектинга пытается определить, что на ней изображено. Назначение - помочь пользователю быстро составить список имеющихся продуктов.

Запуск

На этапе составления списка продуктов для генерации рецепта после нажатия на определённую кнопку пользователь загружает фото из памяти устройства.

Детали реализации

Клиент посылает http POST запрос на API-Gateway с заголовком "Content-Type":"multipart/form-data", в теле которого находится фотография, затем запрос проксируется на PhotoParser. Микросервис принимает фотографию, подготавливает, после чего подаёт на вход заранее обученной модели детектинга. Распознанные классы возращаются клиенту в формате json.

Системная функция — `PhotoParser.parseBar`, распознание штрих-кода продукта

Описание

front-end отправляет фотографию bar-кода товара на endpoint микросервиса PhotoParser, который определяет, что изображено на фото. Назначение - получить детальную информацию об имеющемся у пользователя товаре для более качественного подбора рецептов.

Запуск

На этапе составления списка продуктов для генерации рецепта после нажатия на определённую кнопку пользователь загружает фото из памяти устройства.

Детали реализации

Клиент посылает http POST запрос на API-Gateway с заголовком "Content-Type":"multipart/form-data", в теле которого находится фотография, затем запрос проксируется на PhotoParser. Микросервис принимает фотографию, подготавливает, после чего с помощью детерменированного алгоритма определяет код товара. Распознанный код возращется клиенту в формате json.

Системная функция — `Interface.product_list.take_photo`, ввод данных в список с помощью камеры

Описание

Позволяет пользователю использовать камеру устройства и отправляет полученное изображение на back-end, чтобы выделить продукты.

Запуск и реакция

- Запуск: Пользователь нажимает на соответствующую кнопку.
- Реакция: Запуск камеры устройства для съёмки фотографий.

Детали реализации

Опция использует камеру устройства для создания фотографии и передачи результата на дальнейшую обработку, добавляя полученный результат в список продуктов.

Системная функция — `Interface.product_list.keyboard`, ручной ввод данных в список

Описание

Позволяет пользователю вручную редактировать список продуктов в текущем списке.

Запуск и реакция

- Запуск: Пользователь выбирает элемент списка и/или нажимает соответствующую кнопку.

- Реакция: Изменение состава текущего списка продуктов.

Детали реализации

Опция должна поддерживать добавление, удаление и изменение элементов списка продуктов, при этом не изменяя состояния других элементов.

Системная функция — `Interface.gen_recipes`, создание списка рецептов по спику продуктов

Описание

Передаёт back-end'у текущий список продуктов и ожидает список рецептов.

Запуск и реакция

- Запуск: Пользователь нажимает соответствующую кнопку.
- Реакция: Приложение возвращает пользователю список рецептов.

Детали реализации

Ручка, отправляющая текущий список продуктов на back-end.

Системная функция — `RecipeGiver.get_ingredients_list`, получить список продуктов

Описание

Передать front-end'у список существующих продуктов питания. Компонент системы, с которым функционирует пользователь, должен иметь список всех существующих продуктов, чтобы проводить валидацию ввода пользователем названия продуктов на стороне пользователя. Имеет высший приоритет, так как блокирует работу front-end'а с пользователем.

Запуск

Это http ручка, пользовательское приложение будет дёргать её один раз при запуске, чтобы получить актуальный список продуктов. Список состоит из названий и id, которые создаются на стороне RecipeGiver.

Request

`<endpoint>/get_ingredients_list`, тело игнорируется

Response

В теле ответа json со следующими полями:

```
{
    "version": "строка; версия ответа, пока только 1.0",
    "ingredients": [{"name": "строка; название на русском", "id": "число;
id, выдаваемый RecipeGiver"},...],
    "timestamp": "число; время в которое список ингредиентов был таким",
}
```

Детали реализации

Список продуктов питания поддерживается в базе данных, вверенной микросервису RecipeGiver. В этой базе каждый продукт имеет уникальный числовой идентификатор, он и будет возвращаться. В случае изменения базы данных выданные id продуктов инвалидируются.

Системная функция — `RecipeGiver.get_recipe`, получить рецепт по списку продуктов

Описание

Пользовательское приложение передаёт микросервису RecipeGiver список имеющихся у пользователя ингридиентов и ожидает получить список релевантных рецептов. Имеет средний приоритет.

Запуск

Это http ручка, пользовательское приложение будет дёргать её всякий раз, когда захочет получить список рецептов по данному списку ингредиентов.

Request

`<TODO (я не знаю, что здесь)>/get_recipe`, тело содержит json со следующими полями:

```
{
    "version": "строка; версия запроса, пока только 1.0",
    "ingredients_ids": ["число; id продукта, выданное
RecipeGiver.get_ingredients_list",...],
    "ingredients_names": ["строка; название продукта от PhotoParser, его
множество значений не согласовано с базой продуктов напрямую",...],
}
```

Response

В теле ответа json со следующими полями:

```
{
    "version": "строка; версия запроса, пока только 1.0",
    "dishes": [
        {
```

```
            "name": "строка; название блюда",
            "id": "число; id блюда, выданное RecipeGiver",
            "photo": "строка; как-то сериализованные данные",
            "ingredients": ["число; id продукта",...],  // истинный
состав блюда
            "recipe": "строка; рецепт, инструкция по приготовлению",
            "attributes": [
                {
                    "key": "строка, название атрибута",
                    "value": "строка, значение"
                },
                ...
            ],  // прочие атрибуты, например, источник
        },
        ...
    ]
    "timestamp": "строка; время отправки",
}
```

Детали реализации

Подбор блюда состоит из двух этапов: идентификация продуктов, которые нашёл PhotoParser, и сам подбор.

1. На первом этапе для каждого названия продукта из массива `ingredients_names` подберём ближайшее название из известных RecipeGiver названий продуктов, в качестве метрики используем расстояние Левинштейна. В случае если минимальное расстояние больше некоторого порога (для определения которого ещё необходимо провести небольшой ресёрч), проигнорируем название продукта, в противном случае добавим соответствующий id в список айдишников имеющихся у пользователя продуктов. Вообще говоря расстояние Левинштейна — так себе метрика, в следующих итерациях можно будет заменить её на более умный поиск.
2. Сердце всего RecipeGiver, по какому правилу подбирать блюда --- вопрос сложный. В текущей итерации разработки RecipeGiver будет возвращать список блюд, эквивалентный списку, получающемуся по следующему правилу:
    - Пусть A — множество имеющихся ингредиентов, а "R_i" — множество необходимых ингредиентов для приготовления $i-$го блюда, тогда
    - удалим из списка блюд те, у которых $\frac{|A \cap R_i|}{|A|} < 0.75$;
    - отсортируем оставшиеся блюда по неубыванию величины $|R_i \setminus A|$;
    - вернём 10 первых блюд из списка, если их меньше — вернём сколько есть (в том числе ноль).

    Алгоритм глупый, но для первой итерации в самый раз. Практически весь этап состоит из запроса к БД рецептов.

Требования к внешнему интерфейсу

Пользовательский интерфейс

Пользовательский интерфейс должен:

- Демонстрировать пользователю текущий список продуктов
- Позволять редактировать список
  - С помощью камеры устройства, считывая продукты с фотографии
  - Вручную добавлять и удалять продукты из списка
- Обладать элементом взаимодействия, с помощью которого активный список продуктов конвертируется в список рецептов

Software interfaces

Серверные компоненты должно быть возможно скомпилировать как бинарник или интерпретировать на платформе Linux x86_64.

Аппаратные интерфейсы

RecipeGiver будет использовать СУБД PostgreSQL, требуется, чтобы PostgreSQL-сервер можно было запустить на Linux x86_64 (представление в памяти БД, управляемой PostgreSQL, зависит от платформы). В качестве запоминающего устройства должно быть возможно использовать как SSD, так и HDD.

Коммуникационные интерфейсы

Для сервиса RecipeGiver должен использоваться REST API, подробнее см. Системные функции.

Сервис RecipeGiver будет обращаться с помощью соответствующих хранимых процедур к базе данных с рецептами.

Нефункциональные требования

Требования к производительности

Так как наш проект пока находится на начальном этапе, мы предполагаем небольшую загрузку наших мощностей. Измеряя в численных метриках порядка 100 RPS (Requests per second).

Требования для обеспечения консистентности (Safety requirements)

Будет производится валидация данных на каждом из этапов обработки, а также везде, где возможно возникновение ошибок, будут написаны обработчики с логированием.

Требования для обеспечения качества программной системы (Software quality attributes)

Конечно к каждому сервису необходимо будет написать юнит и интеграционные тесты для базовой проверки их на работоспособность. Для оценки сложности кода планируется обсчитывать цикломатическую сложность при помощи соответствующих пакетов/библиотек для каждого из языков, применяемых в проекте. В качестве архитектуры системы планируется использовать вариант монолитной архитектуры. При разработке будем стараться придерживаться принципов SOLID. Написанный код будет приводиться к стандартам оформления, если таковые есть, как, например, PEP8 для Python, для Go же просто будем применять `gofmt` к каждому из файлов ради принятого единообразия, для Kotlin - `ktfmt`.

Требования информационной безопасности (Security requirements)

Конкретных требований по безопасности данных мы не предъявляем, поскольку никаких авторизаций со стороны пользователя наш продукт не предполагает, человек просто сразу начинает сканировать и мы даем ему результат. С нашей же стороны мы используем базы данных, наполненные информацией из открытых источников, поэтому также к себе не предъявляем каких-либо особых требований.