TRUE NORTH

Date: November 20, 2018

Author: persianahuel@gmail.com

Version: 1.0

CODING CHALLENGE

# TRUE NORTH

Architecture Document

## 1. INTRODUCTION

This document describes the architecture developed for the Coding Challenge, the methodology used, and recommendations for hypothetical future iterations of this project.

## 2. CURRENT SITUATION

It is requested to develop software that is capable of:

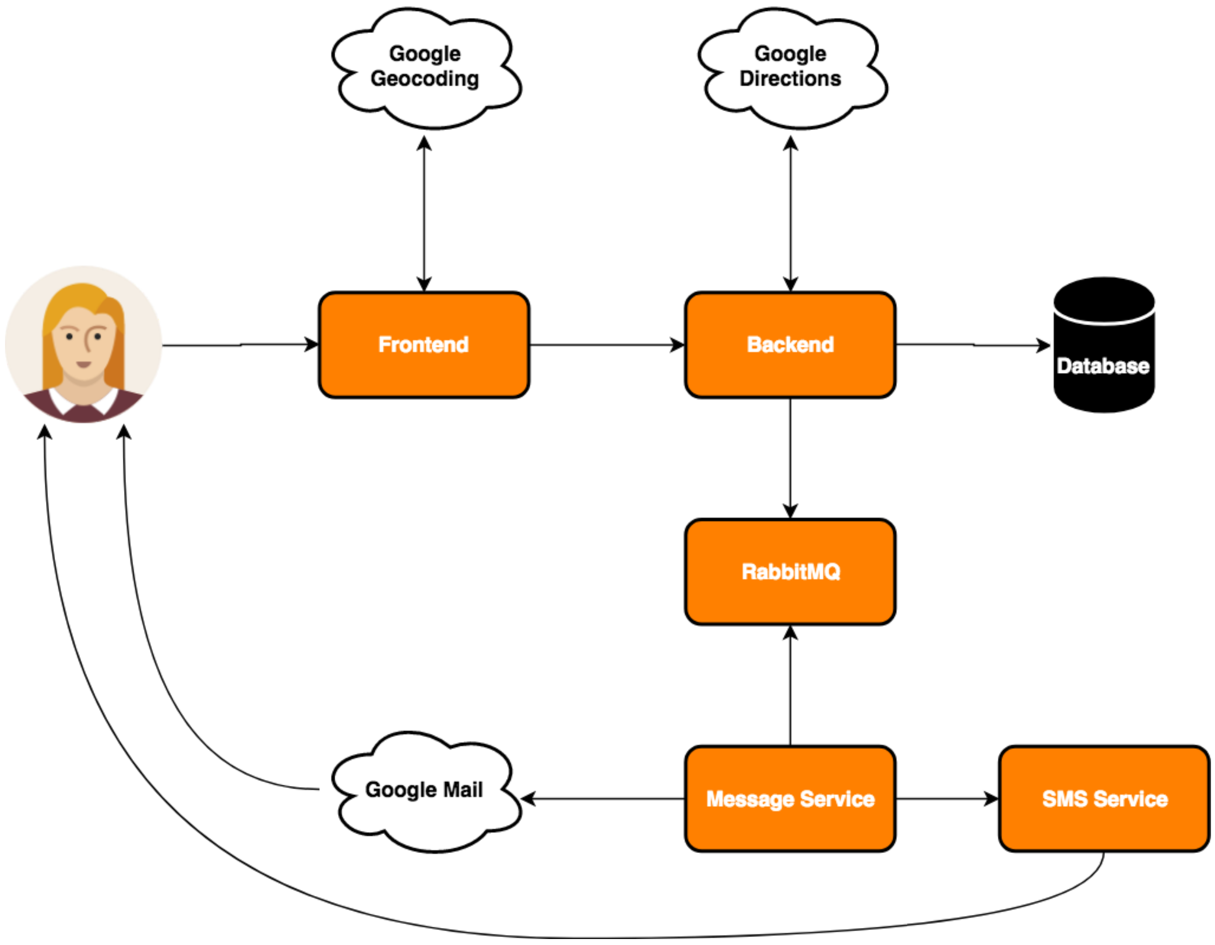
- Manage customer orders towards restaurants
- Create restaurants and their Meals

## 3. PROPOSED SOLUTION

### 3.1. DESCRIPTION

To carry out this development, a micro-services architecture implemented in containers was chosen.
Since this software had several components from the request, the use of Dockers facilitated the development, the communication between the components and their installation.
It should be noted that with the addition of Docker Compose, a primary layer of orchestration was obtained (far from being able to be compared with solutions such as Kubernetes or OpenShift), but sufficient to facilitate control over the components.

### 3.2. ARQUITECTURA



#### Frontend

Web application developed in ReactJS. User interface, which allows:

- to customers, place order orders
- to administrators, create new restaurants and meals

It has a small integration with the Google Geocoding service to calculate at the time of loading the address, latitude and longitude.
This application integrates with the Backend through the REST API to send all the necessary information.

#### Backend

Aplicación desarrollada en Python con el Framework Flask.
Esta aplicación sirve de toda la información necesaria al Frontend y se encarga de gestionar todas las comunicaciones con la base de datos.
Se integra con el servicio Google Directions para calcular los tiempos de viajes entre un restaurant y la dirección del cliente que realiza la orden.
Adicionalmente, se integra con la cola de mensajería para enviar información del pedido tanto a restaurants y clientes

#### RabbitMQ

Messaging queue in which two queues were implemented:

- For restaurant orders
- For confirmations to users

#### Message Service

Application developed in Python with the Flask Framework.
Responsible for taking messages from the messaging queues:

- If you are from the restaurant order queue, send an email to the restaurant with the order detail
- If they are from the queue of confirmations to the user, send an SMS to the user with the detail of the order

#### Database

PostgreSQL Database
It contains all the information related to restaurants, meals and orders.

#### SMS Service

Simple mockup developed in Python.
It is responsible for emulating the sending of SMS to the client.

#### Google Geocoding

Google Geocoding is a web service for converting addresses into geographic coordinates.

#### Google Mail

It is the google mail service, this is used to send orders to restaurants.

### 3.3. TECHNOLOGIES

ReactJS: this framework was used develop the frontend app.

Python 3 with Flask: for the rest of the developments, was used Python 3 for coding and Flask framework for expose REST API.
PostgreSQL: this is the database selected to save all information, like restaurants, meals and orders.
RabbitMQ: it is the message queue where it's saved orders and notifications
Dockers: It is the technology used to implement the all solution, each develop and infrastructure component was implemented in a isolated container. For simplicity, in this solution it was implemented docker-compose for quicks deploys.

## 3.4. ASSUMPTIONS AND CONSIDERATIONS

- The software uses integrations with google to calculate positions and delivery times. The end-to-end operation of the solution requires access to these services.
- It is necessary to have an internet connection, unlimited and without restrictions.
- The deployment of all the components was done with Dockers version 18.09.0 and docker-compose 1.23.1
- All the components were developed and tested in the same computer, the correct operation is not assured with their containers distributed in different environments.
- The Class dbManager (backend service) was selected to do the unit tests, but to run it you must do it in a external way, because it is not included in the docker container

## 3.5. POSSIBLE IMPROVEMENTS FOR FUTURE DEVELOPMENT EVOLUTIONS

- When displaying restaurant and meal listings, we could limit the maximum amount and add a page functionality in the frontend to avoid saturating the database.
- This is my first experience developing a frontend. Any improvement that can be made to the developments in React JS would be correct.
- Docker compose has a functionality to control the order of container deployments, but this feature does not control whether the container is ready. For this challenge, I put a 30-second sleep in the message service to wait for the message queue to be ready. In a production implementation, I must implement health checks:
    - Liveness probe: to see that the container is ready to work.
    - Readiness probe: see that the container is working.
- In this challenge, I use the included application server in Flask, but it is not recommended to production. For a production environment i will used GUnicorn for example.