Smart Service Development - Exercise 2

Wagner, Lukas 01430206
Kissich, Meinhard 11771817
Weinbauer, Klaus 11721179

April 29, 2021

1 Service Idea

Fast short-range goods delivery seems to be a fast-growing market with numerous examples out there. As in many cases, Amazon is one famous example. It is now possible to receive some goods within just a couple of hours after ordering when living in a large city[1]. But it does not end with Amazon. Since the corona crisis forced governments worldwide to close restaurants, food delivery got a boost in popularity[2]. Still it does not end with food. Also, the ambulance can be seen as some short-range, fast delivery service - with some special treatment, of course. However, all these examples combine the same key features: (mostly) short-range, short job duration, source and destination of some matter, (extremely) fast scheduling, many workers ("agents") that do the transportation, optimization of resources, etc. Now, we think that we can provide a service that can do better than others. (Or at least, we want to have a piece of the cake). The provided service aims to give an interface for such transportation jobs, schedules the right resource to the right job for various optimization strategies, and monitors the open missions after assigning the resource. Besides, we also serve the agents on the job with great features - the selection of suggested jobs, automatic route planning, and many more. The main advantage is that the provided service is extremely versatile over many sectors and applications within the same API.

2 Research

The services consist of three parts as depicted in section 3.3. Some of these parts heavily depend on other existing services and provides an abstraction layer to introduce features or exchange the hidden service. This can range from

[1] https://tinyurl.com/9sfuh3dc
[2] https://tinyurl.com/4xbzz96d

introducing caching to pay for less traffic, adding some annotations or data, or exchange the service completely when moving to a cheaper supplier. However, other parts of the service are entirely self-implemented, like database management. Not only because of data privacy issues but also to have complete control over data and - if legislation permits - use data analysis to optimize the service on demand.

2.1 Existing Services

On-demand logistics increased in popularity over the last few years. Especially due to the COVID-19 pandemic, the demand for delivery services has reached a new high, and companies are starting to establish in this market. Although there are already some suppliers that serve this need, it is still an exciting and promising area to start a business. The predictions for crowdsourced delivery could not be any better. According to an article in *SupplyChainBrain*, they estimate that 90% of retailers expect to use crowdsourced delivery by 2028[3]. A popular supplier in this new area of logistics is *GoShare*. *GoShare* has more than 65000 customers and more than 8000 drivers and is mainly present in the United States and some states in South America (Brazil, for example)[4]. Although the competition in America is already quite challenging, the European market is still underpopulated. This is where our service will establish and, due to the predictions of the market, develop sustainable growth with a solid customer base shortly.

2.2 Dependencies of our Service

Our service's most significant dependencies will be a geolocation service like Google Maps or Open Route Service. Since we provide a logistic service, the requirement for geographical awareness is inevitable. Because of convenience and the fact that the mentioned providers offer an already feature-rich API, we choose to use one of them. For the first step of our system design process, this will be the only dependency on external suppliers. Maybe in a later stage, when we encounter issues with logistic algorithms or job distribution, we might consider including additional dependencies to dedicated services for those tasks. But at this stage, we did not plan anything in this direction.

[3] `https://www.supplychainbrain.com/articles/30887-crowdsourced-delivery-is-here-to-stay`
[4] `https://goshare.co`

3 Initial specification

3.1 Terminology

First we want to establish certain terms that will be used throughout the rest of the document.

- A **job** is a delivery of a certain subject or object from point A to point B, e.g. transporting a person from one address to another, or delivering a pizza from a certain restaurant to a given address.
- A **customer** is the issuer of a task. Since our service targets other service providers, such as restaurants or taxi services, this must not be confused with the end-user, in other words the customer of our customer. Customers will typically use our customer API from their own back-ends. In other words, there will be no direct communication between end-users (the customer's customer) and our service API, our customers will act as an intermediary.
- An **agent** is the entity that fulfills a job, for example a taxi driver or food delivery driver. Agents will typically use our provided API using a smartphone app.

3.2 Primary Data Sources

Our service only requires one external data source, which is used for geocoding (resolving street names/addresses) and route calculation. There are multiple services that could be used here, for example Google Maps[5] or Open Route Service [6]. We will likely choose the latter.

3.3 Provided services/Data Transformers

Our current plan is to split our service into three micro-services:

- **Job service**. This service is used by our customers to create, update and delete jobs. Also, customers can query the state of a job they have created (open/assigned/done). Agents can use this service to ask for a list of jobs that they can take, accept a job and mark jobs as done.
- **Agent service**. This service is used by the agents to update their status (position, availability, etc.). The last known state of an agent is stored in a database. Customers can query information such as the current number of available agents, proximity of the next available agent, and also the position of an agent if and only if that agent is currently executing a job created by this particular customer.

[5] `https://developers.google.com/maps/`
[6] `https://openrouteservice.org`

The motivation behind splitting this service from the Job service are mainly separation of concerns and scalability. In general, there are much more agents than there are customers. Also, agents will update their state quite frequently due to position updates. Thus, the split between job service and agent service allows us to spawn a large number of agent service instances (horizontal scaling) to deal with the traffic.

- **Geocoding/Routing service**. The main purpose of this service is to provide an abstraction for third-party providers such as Google Maps or Open Route Service. The benefits of creating an abstraction layer are as follows: First, we can simplify the rest of our application, because we don't need to know about any details about the concrete providers and also because we are only using a very small subset of the functionality provided by those services. Secondly, we can easily switch providers if we have to. If we, for example, discover that the data from openrouteservice is insufficient for our application, we can simply rewrite our abstraction to use Google Maps, without the need for any modifications in the rest of our code base. Lastly, we can introduce **caching** to reduce the use of the provided APIs which might traffic and thereby cost.