Net Computing
Final report

Lars Doorenbos(s2507803)

April 5, 2018

Contents

1 Introduction

Finding a parking spot in a busy city can be a difficult task so we propose a software solution within the theme of the connected city to simplify this process. The Parking Spot Application will help car owners find a parking spot in the city that best suits their needs. In order for this to work properly all parking lots connected with the application will have to be equipped with sensors to track if there is already a car present in said spot. This real time data will be gathered and sent over a network to a server that receives and processes the obtained data.
A car owner will be able to use the Parking Spot Application both on his mobile phone and on a website, so that the user can find a free parking spot instead of driving through the whole city searching for one. Via this application the user can select a destination and it will then determine the best and closest available parking spot, complying with the constraints given by the user such as the maximum price that the user is willing to pay.

Note on running the program: running the main.java starts the area servers with all their sensors, while client.java is the way to request a parking spot in an area. The back end functionality for i.e. requesting a spot in a street is implemented, but not in the client at the moment.

2 Use cases

2.1 Use Case 1: Find an available parking spot that best suits your needs

- **Use case introduction:** In our parking spot application, a user can fill in his constraints with which the parking spot best suiting his needs can be determined. This use case describes the process of filling in the constraints of the user and the system then finding the best parking spot available.
- **Stakeholders:**
  - Person(s) looking for a parking spot: Wants to park as quickly and in a spot as closely resembling his needs as possible
  - City: Wants to have the least amount of people aimlessly driving around looking for a parking spot and clogging the traffic.
- **Main success scenarios:**

  (1). The person(s) looking for a parking spot enters his destination

  (2). The person(s) looking for a parking spot enters the maximum amount he wants to pay for his parking

  (3). Step 2 can be repeated for the other preferences.

  (4). The application calculates the best spot currently open

  (5). The user is presented with this best spot.
- **Alternative flow:**

  (5a). The parking spot is taken while the user is driving there

  (1). The system informs the user

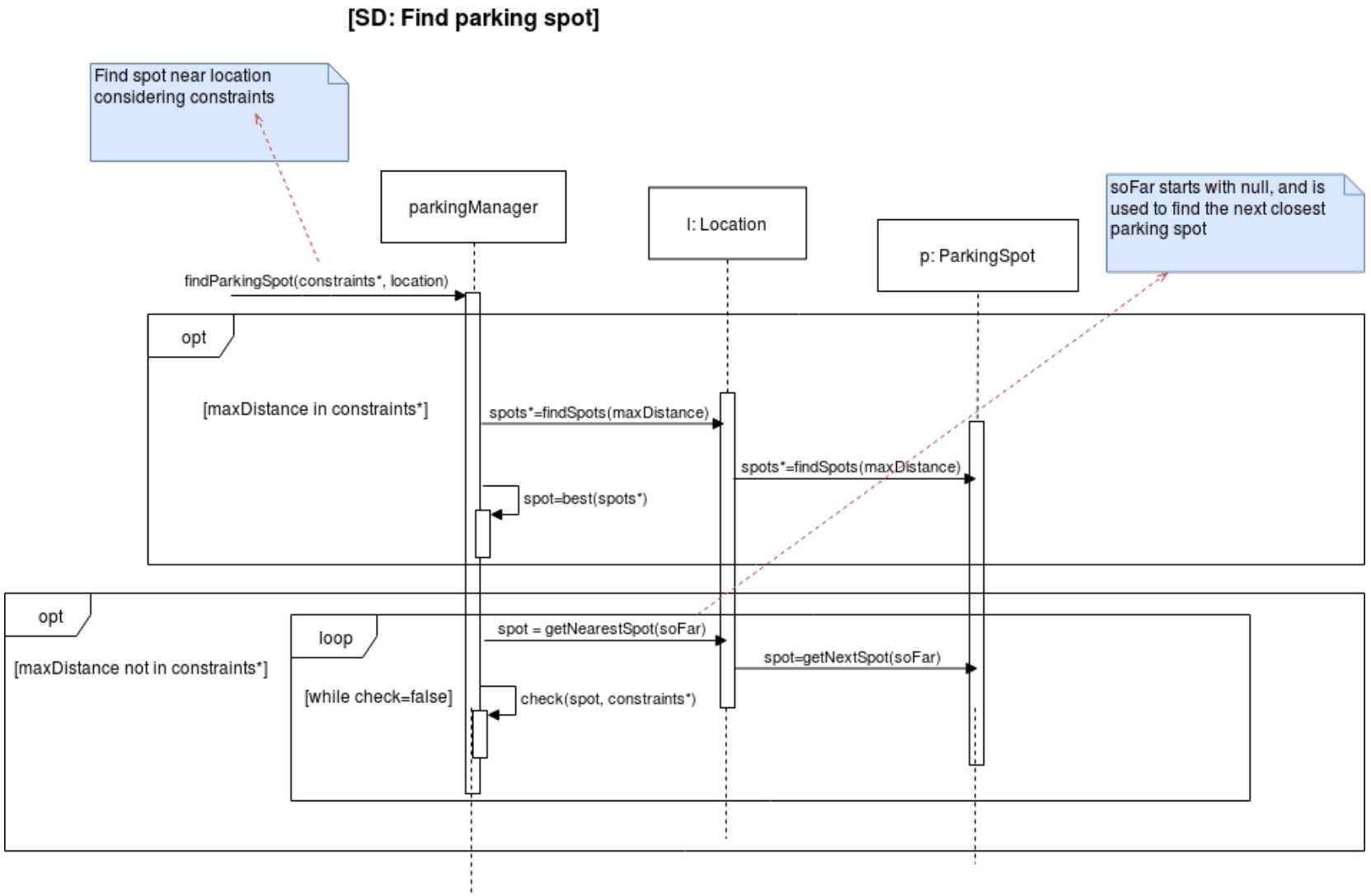
  (2). The system calculates the new best parking spot taking into account the preferences

  (3). The user is presented with this new best spot.

  (4a). No parking spot satisfying the entered constraints exists

  (1). The system notifies the user

  (2). Main success scenario continues at step 2.
- **SD:**


[SD: Find parking spot]
Find spot near location considering constraints
parkingManager
l: Location
p: ParkingSpot
soFar starts with null, and is used to find the next closest parking spot
findParkingSpot(constraints*, location)
opt
[maxDistance in constraints*]
spots*=findSpots(maxDistance)
spots*=findSpots(maxDistance)
spot=best(spots*)
opt
[maxDistance not in constraints*]
loop
[while check=false]
spot = getNearestSpot(soFar)
spot=getNextSpot(soFar)
check(spot, constraints*)

2.2 Use Case 2: Register an account

- **Use case introduction:** In our parking spot application, a user can register an account and save his preferences so he does not have to fill them in every single time. This use case describes the process of registering such an account and setting the preferences so that he does not have to fill them in every single time.
- **Main success scenarios:**

  (1). The user enters a name and password he wants to use

  (2). The user enters his preferences/constraints he wants to use every time (can be changed later)

  (3). The application registers the account with the given preferences/constraints

- **Alternative flow:**

  (aa). The user name is already taken

  (1). The system informs the user

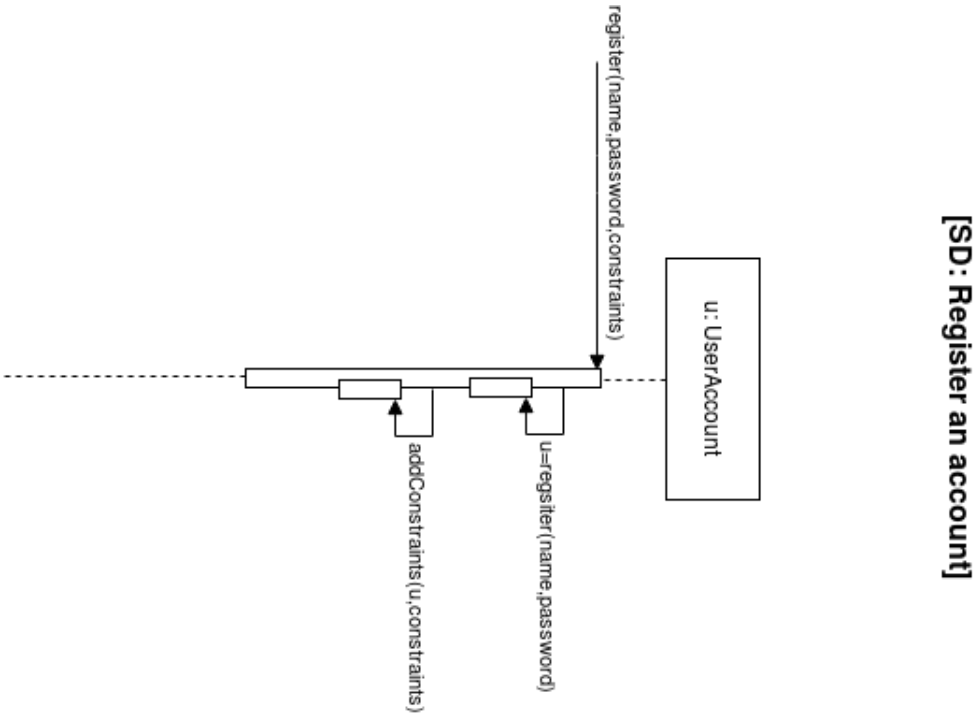
  (2). Main success scenario continues at step 1.

  (1b). The password is invalid

  (1). The system notifies the user

  (2). Main success scenario continues at step 1.

- **SD:**



3 System design

Our system consists of four main components, namely sensor to area server communication, communication between area servers, client to server communication and client login and storage. We will cover them in order:

3.1 Sensor to area server

The parking spots are equipped with pressure sensors to sense if there is currently a car parked. The sensors send this information via a network socket to the corresponding area server. If a change in pressure is monitored which means a car has been parked or has left a message is sent. For sending messages we will use TCP, and only send a message as soon as a change in pressure is monitored. We chose this over sending a UDP message periodically, as we assume changes do not happen so often that this is warranted. This thus reduces the load on the system. We send a SensorPackage, containing the spot and street name of the sensor and its availability to the area server, which keeps track of all its sensors and updates accordingly. On the area server side, a TCP Server continuously checks for incoming requests, and in the case of sensors data, it handles it by setting the values appropriately.

3.2 Inter area server

There are 2 kinds of communication between servers, one happens when a free spot is requested by the user, and the other is a backup triggered on changing of a sensor.
When a free spot is requested, the request is first send to the area in which the street the user wants to park nearby resides in. This is simply a string stating a parking spot needs to be found. The area then checks in its own spots whether an available spot exists, and if so returns the sensor package associated with the free parking spot to the user, as this contains both the street and the spot in that street. When this is not the case, it first requests what currently active areas are closest by from the database, returned as an integer array of indexes. It will then in that order request a free parking spot from these areas. It will also send a string to these areas to search a free spot, but so that the other area will only check its own spots and not also go through its own closest area list, as this will result in an infinite loop. In the end the starting area return the best parking spot as sensor package to the user.
As for the backups, each area server has a backup of one other area server. Every time a sensor sends an update, the area server also updates its backup by sending its sensor data array list to the area server keeping its backup. The sensors are sent so that in the case of the removal of an area, backup data is immediately stored somewhere else. Also if a parking spot sensor would be added or removed this would instantaneously take care of this. The backups are done using Rabbit MQ to ensure the backup data will be received and processed. There is thus an overlap of 1 in all area servers, this way when an area server falters the backup can still be found. As the data is not of vital importance, we deemed it unnecessary to increase the level of backup implemented. At first the program contained a central backup database, updated on change via UDP. The new way however better fits into the scope of the course.

3.3 Client to server

The user will use a http based client which will sent a request to the server containing the given parameters, such as their destination and a maximum price. This communication will be done using a RESTful web service, which sends the http requests to the server and makes sure the correct information returns. To make sure that no two users are assigned the same spot at a given time, the application has to assign a spot to a user for a given time, and keep this spot reserved. It then keeps track if the user actually parked on that spot, either by allowing access to GPS or having the user confirm he parked in said spot. If this is not the case within the given time the spot is available for others again. This reservation time should depend on traffic and the distance between the user and the parking spot. If the parking spot that was reserved is taken by another car it will have to notify the user and find a new spot. The data send back and forth between client and server would be in JSON format, as this is easy to process and create.

3.4 Client login and storage

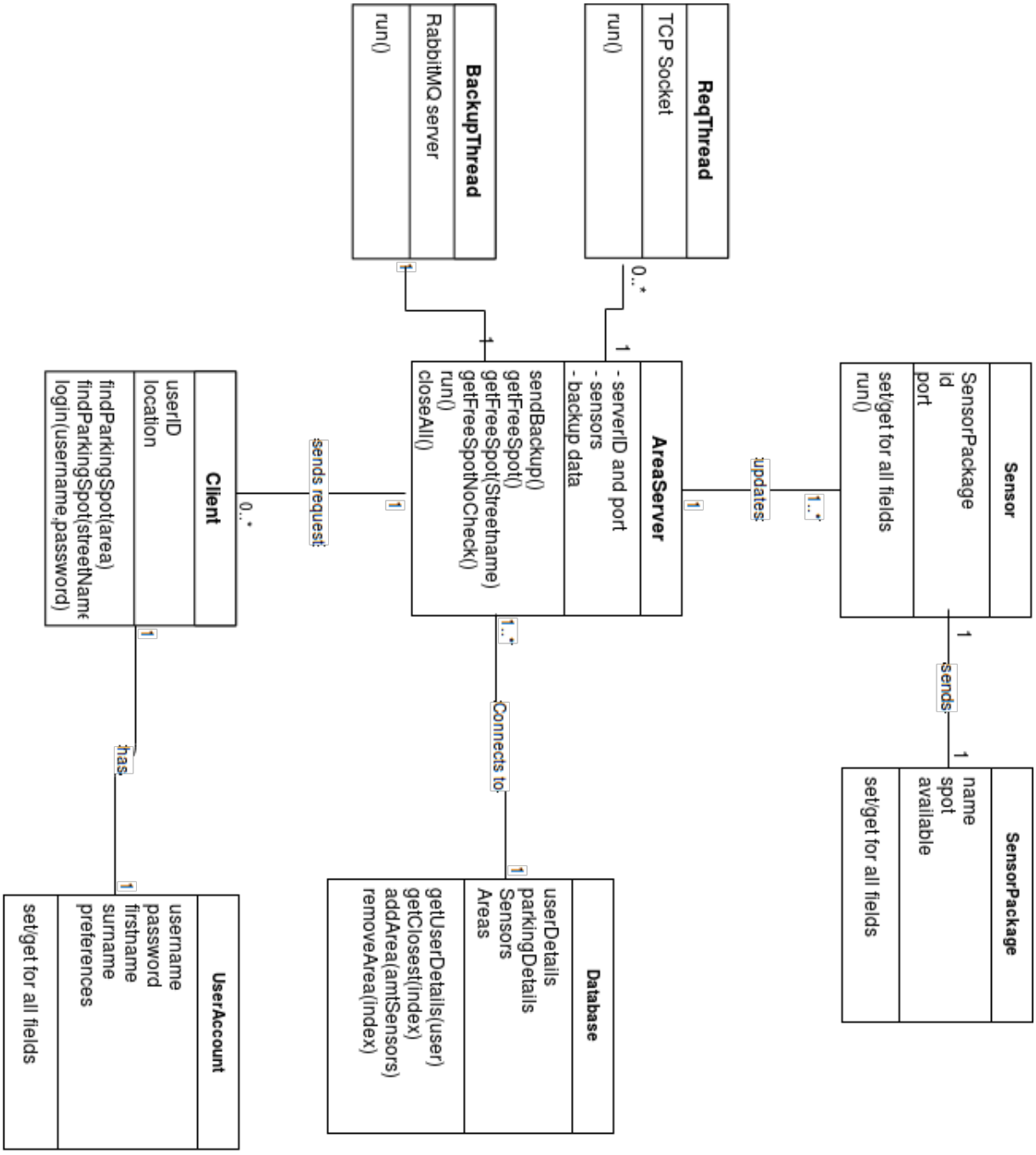
Users can login to the client with their username and password to give them extra options and store their personal preferences and information. This requires a database to store the users and their password. When the client tries to login to the server, it will check whether this username and password exist on the server. If this is the case the user is logged in. This would be done by posting the data to the database and inspecting

the returned value, which would either be fail or success.

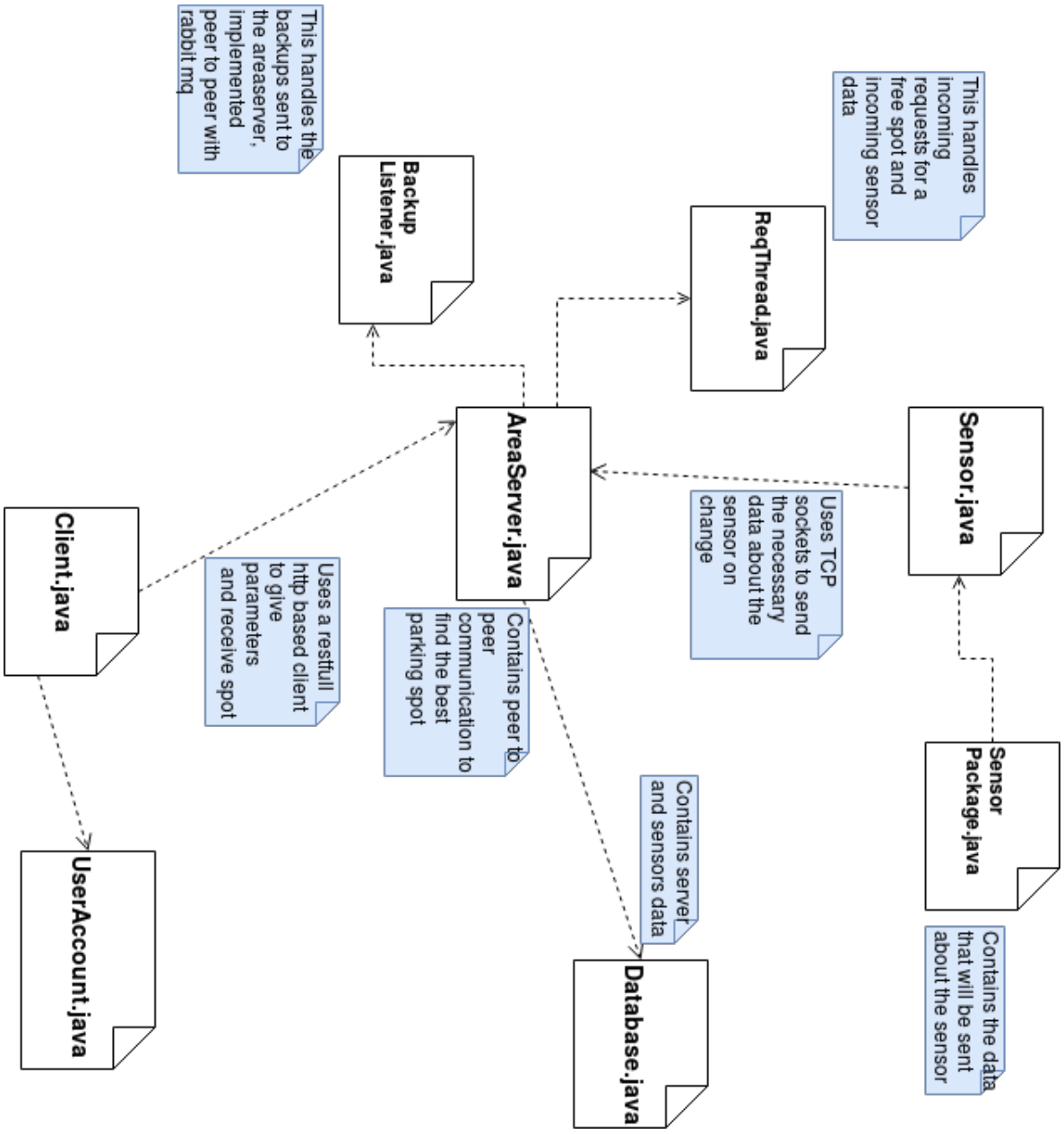
3.5 General decisions

We decided to change our architecture to be more P2P by having not one server that contains all the information, but several area-based servers that contain data only relevant to that area.
As for our simulation, we use a random number generator to determine whether a sensor will change its state or not. Currently there is a 5% chance every second to change state for every sensor. Originally the backups would also only have some chance of sending, but as we now only store the data on one other area server, that is removed.
I implemented rabbit MQ in the backup communication instead of the client-server or sensor-server communication as in those cases the communication is two-sided, for which sockets lend themselves better. Both the TCP socket connection and the rabbit MQ connection ensure no data gets lost and everything is handled properly.
The area servers are now running as separate threads in the same program, this has in our case the same effect as if they were separate programs, but it makes it easier to test and debug.

4 Class Diagram



5 Component Diagram



6 Evaluation & Conclusion

The REST part and user database are completely missing, because of the inability to contact my teammate while he said he had it done but not having shared the code anywhere. I have tried to explain it as best as I can in the decisions section, but to start coding all of his part as well in 2 days was too big a task and I already had taken most of the other group partners work on me as well. The report has been my main focus.

To simulate the client I have made the client.java class, which will connect with the application and in which you can enter the area in which you want a parking place. Obviously, this can use improving as of right now it is mainly for testing.

After realizing I was on my own, I decided to ignore constraints as maximum price and optimization of finding the best spot, as this has less to do with net computing as with a field such as operations research.

All in all, the application works as it should, but it is very minimal and needs improvement before it can actually be put to use in a real world, connected city situation.

Github: https://github.com/PJEilers/NetComputing