Power EnJoy

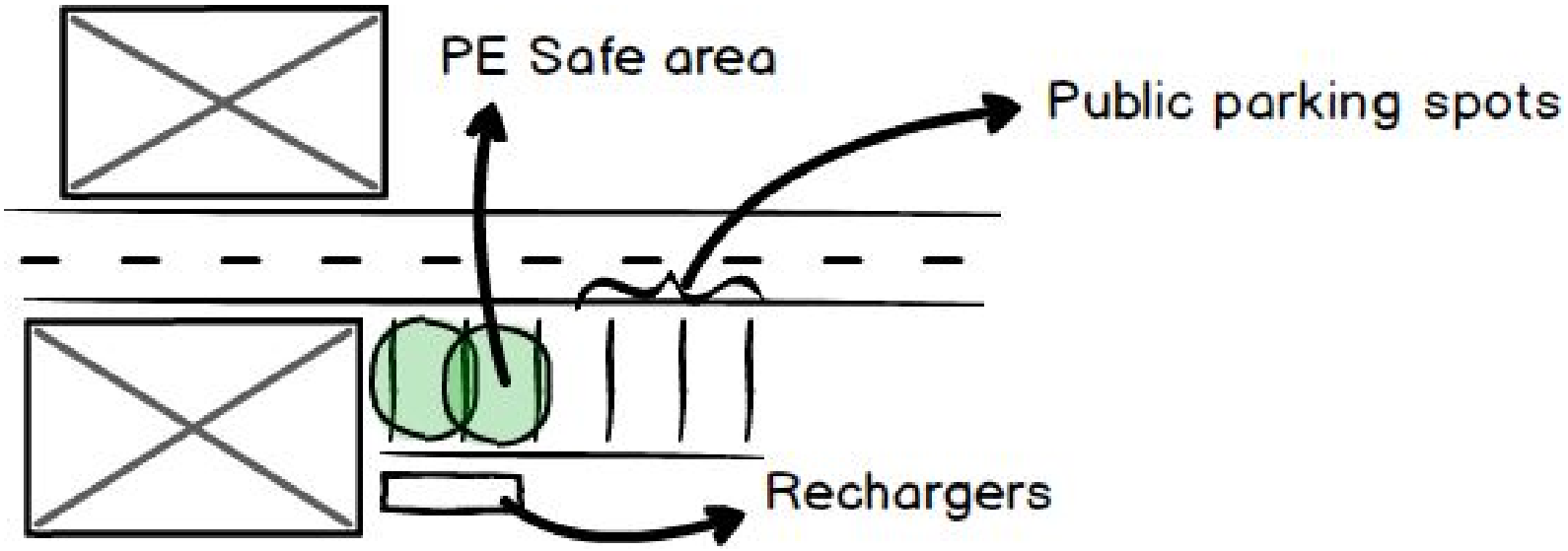
By William Di Luigi & Andrea Battistello

Agenda

- Scope and domain assumptions
- System overview
- Case of study: car handling
  - Requirements
  - Design
  - Testing

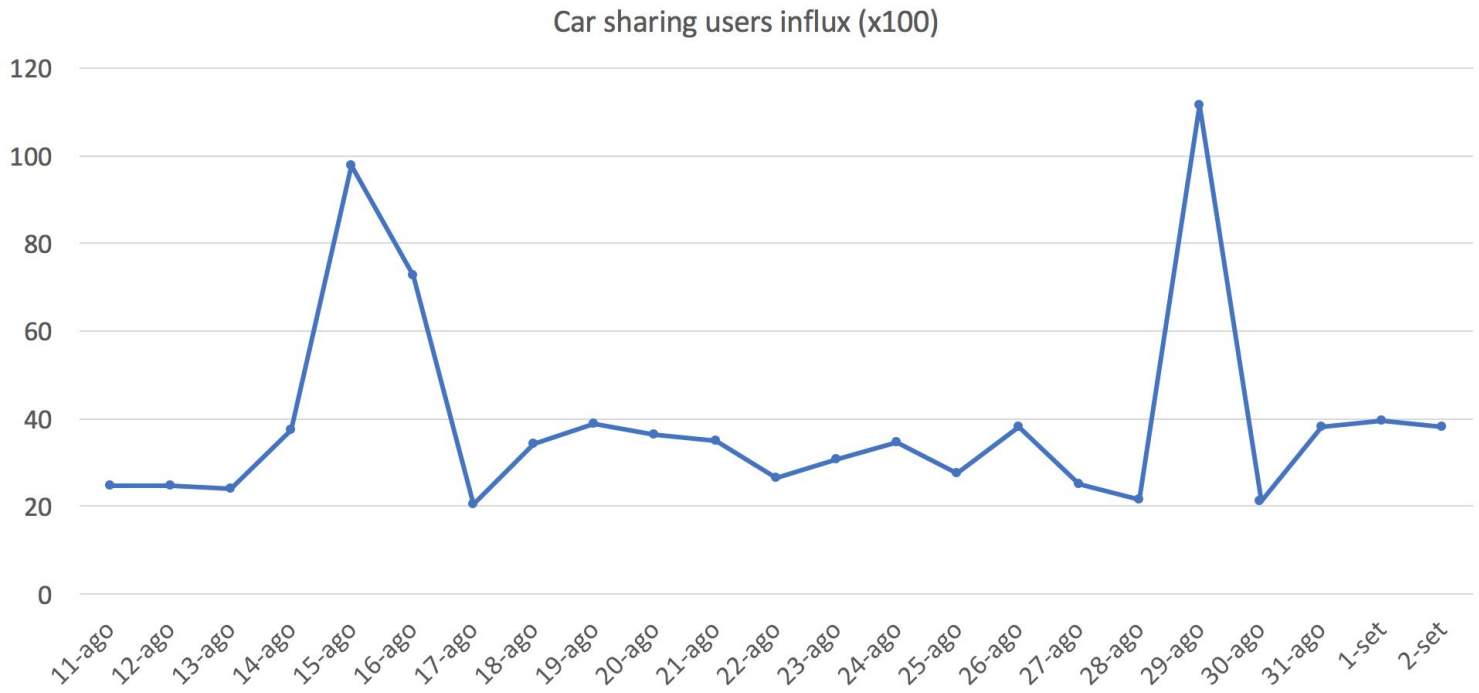
Scope and domain assumptions

Safe Area



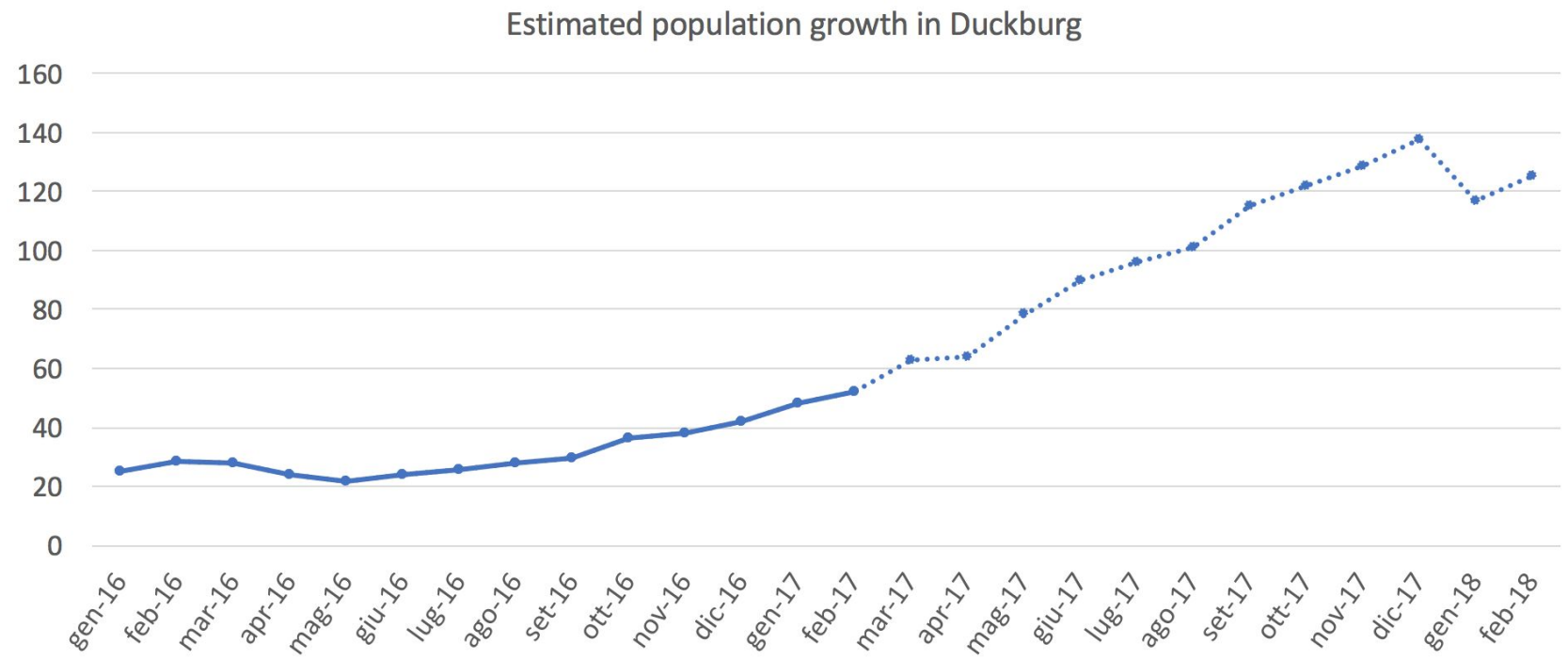
Types of users

- **Type A**:
  - Routinely uses car sharing systems
  - Uses the service for everyday activities
- **Type B**:
  - Sporadically uses car sharing systems
  - Typical case: when there is a public transport strike



Duckburg

- Poor public transportation
- Lot of traffic jams
- Increased pollution
- Number of inhabitants is expected to increase in the next years



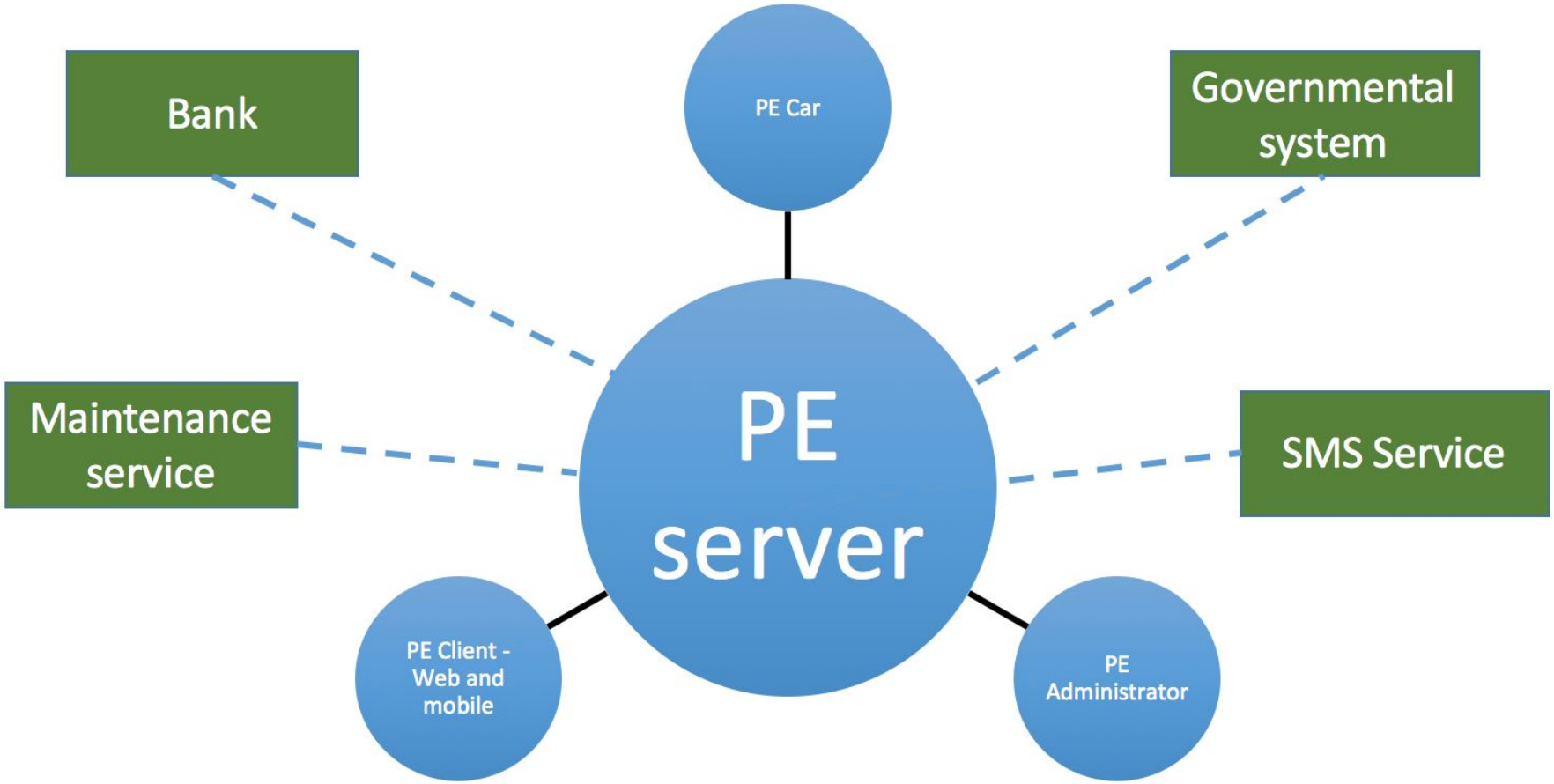
System overview

Goals

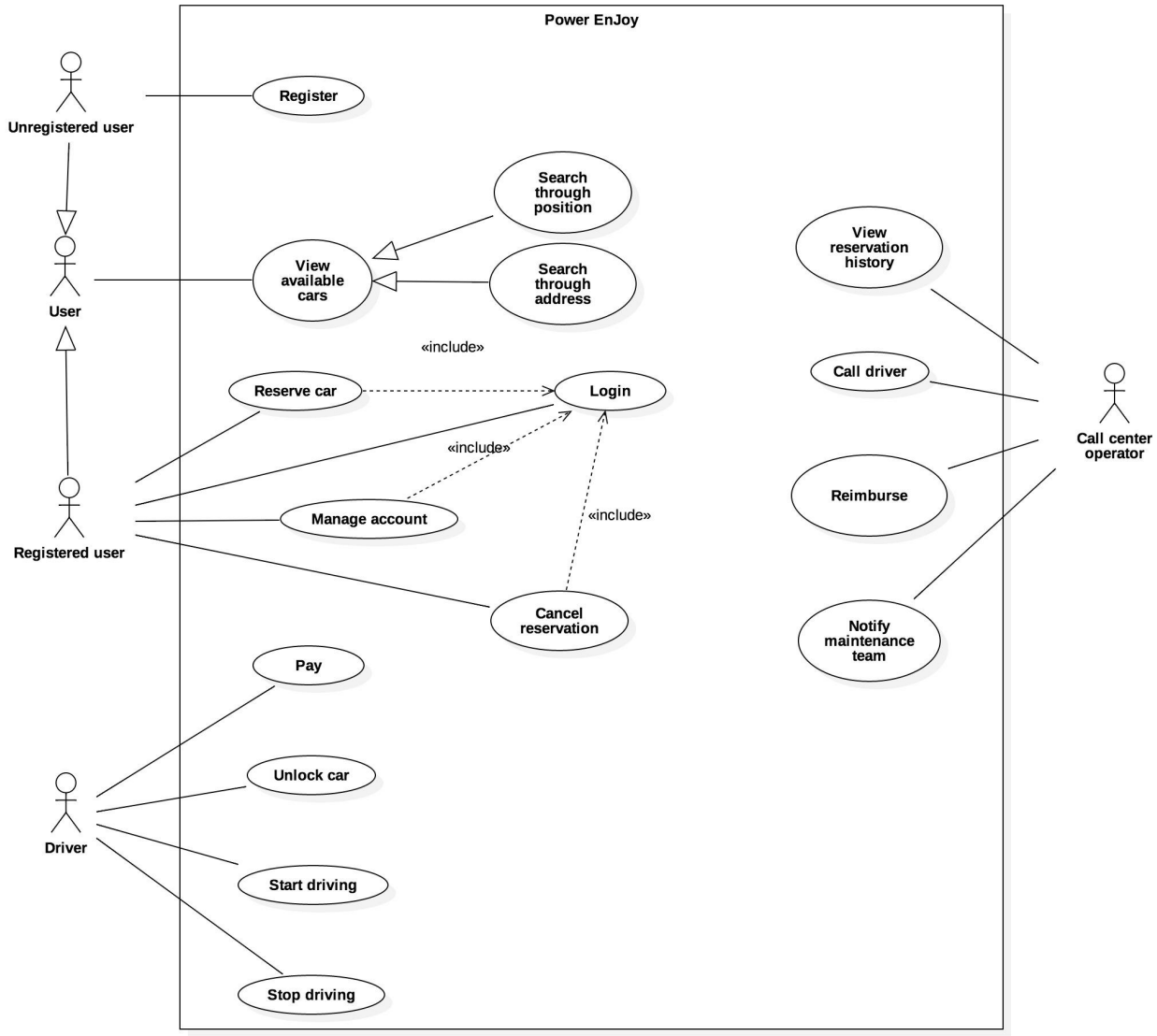
What we want to achieve

- ✔ Build a **reliable** service
- 👍 Encourage users to respect rules with **positive reinforcement**
- 🆘 Offer an efficient system of **customer assistance**

System overview



Use case diagram



Build a **reliable** service: view & manage cars

Functional and non functional requirements
Real time car monitoring
Efficient car handling
Verification and validation

Functional requirements

[G1.1] View available cars nearby or in a given position

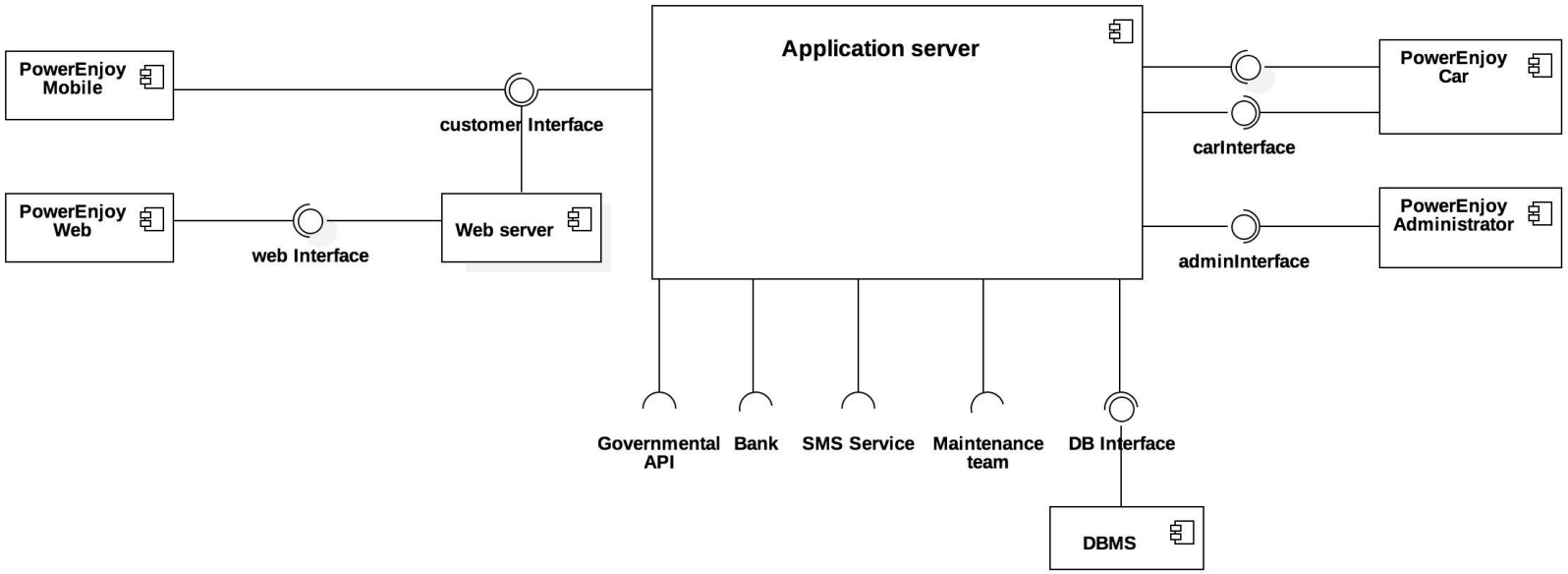
[R1.1]	PEM and PEW should be able to retrieve device position either with a GPS antenna or by using browser geolocalization after obtaining the user's permission.
[R1.2]	PE should be able to find the GPS coordinate given a specific address.
[R1.3]	Every car must respond with their GPS coordinate when it is asked by PE.
[R1.4]	The user should be able to specify the range of the area where to find available cars.
[R1.5]	The user should be able to visualize the battery level of available cars.
[R1.6]	Each car must have a fixed minimum battery level in order to be considered available.

Non functional requirements

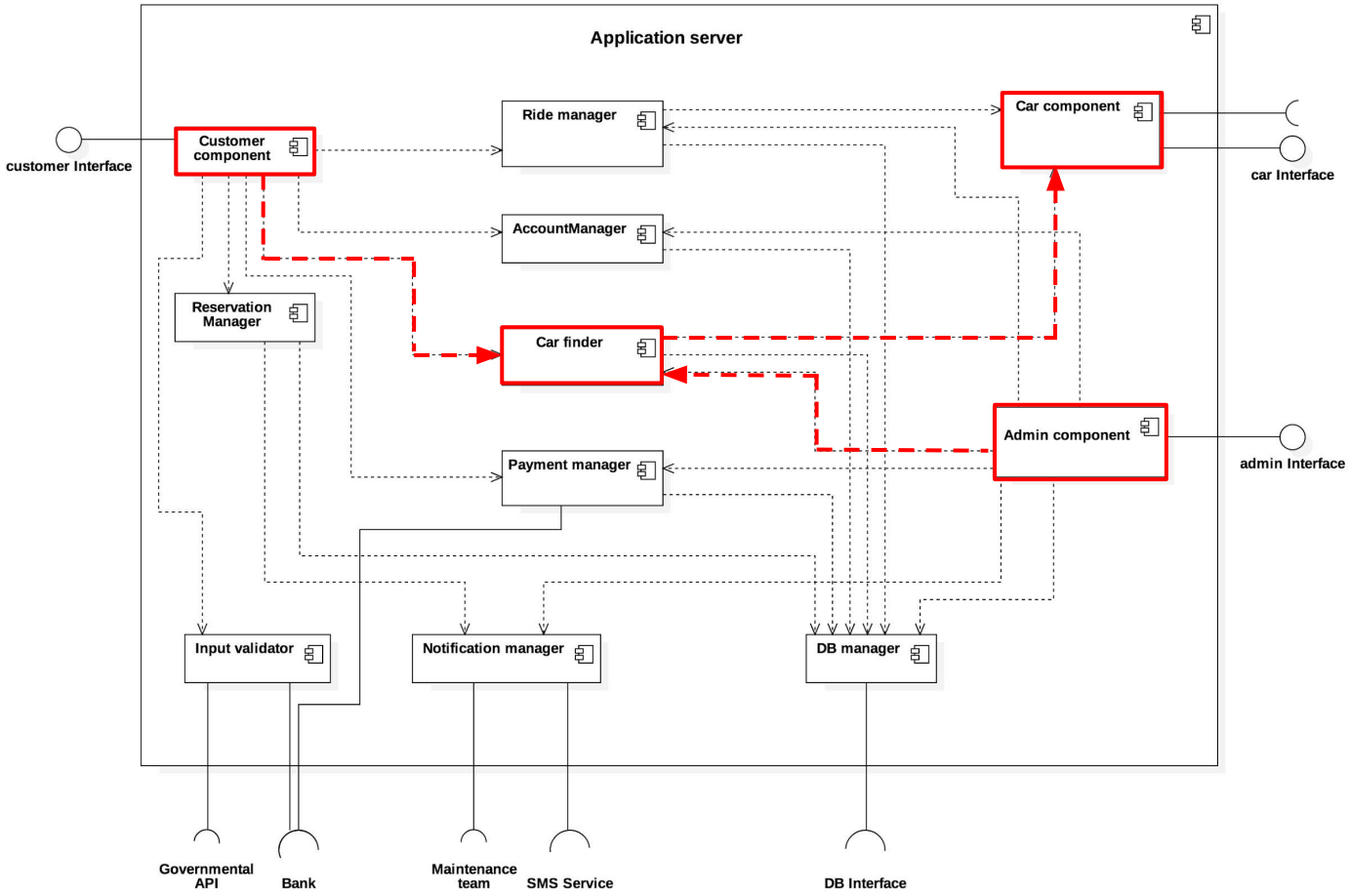
- Performance requirements
  - Should be able to handle many requests at a time and support real time car monitoring

- Availability and reliability
  - Should guarantee a good service when there are spikes in requests

Component view



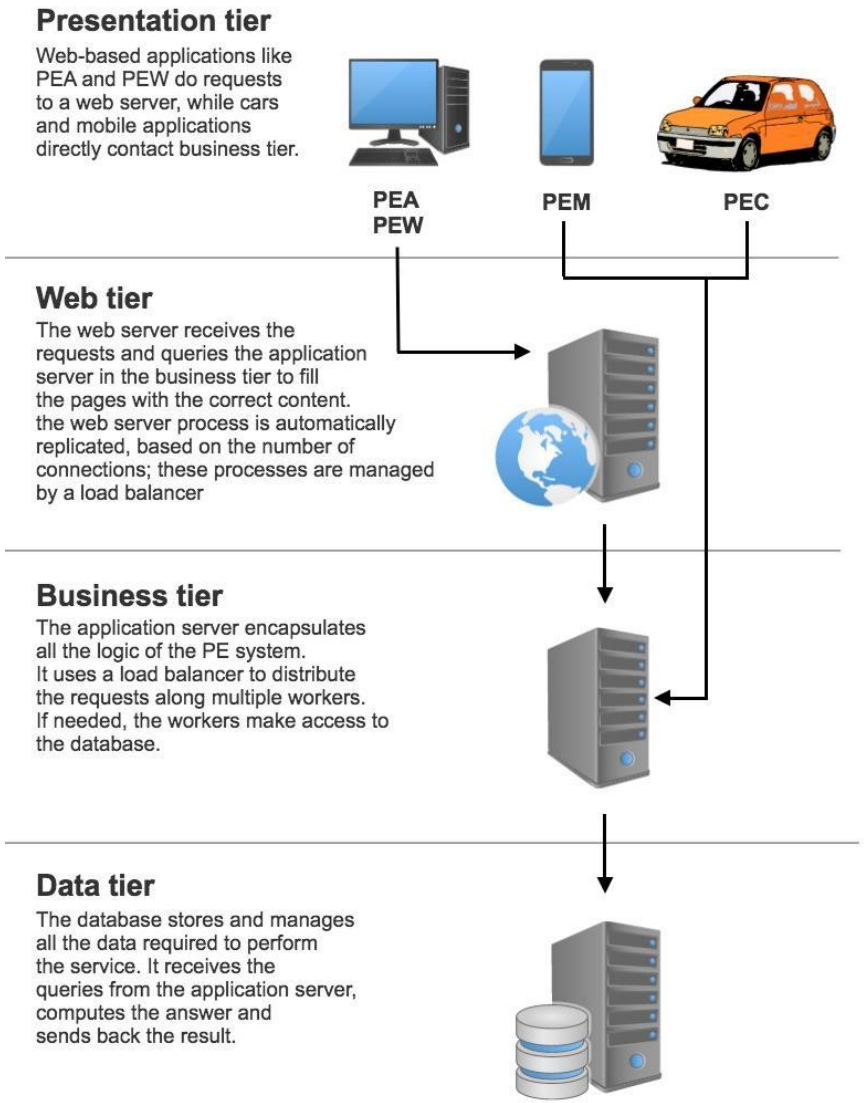
Application server



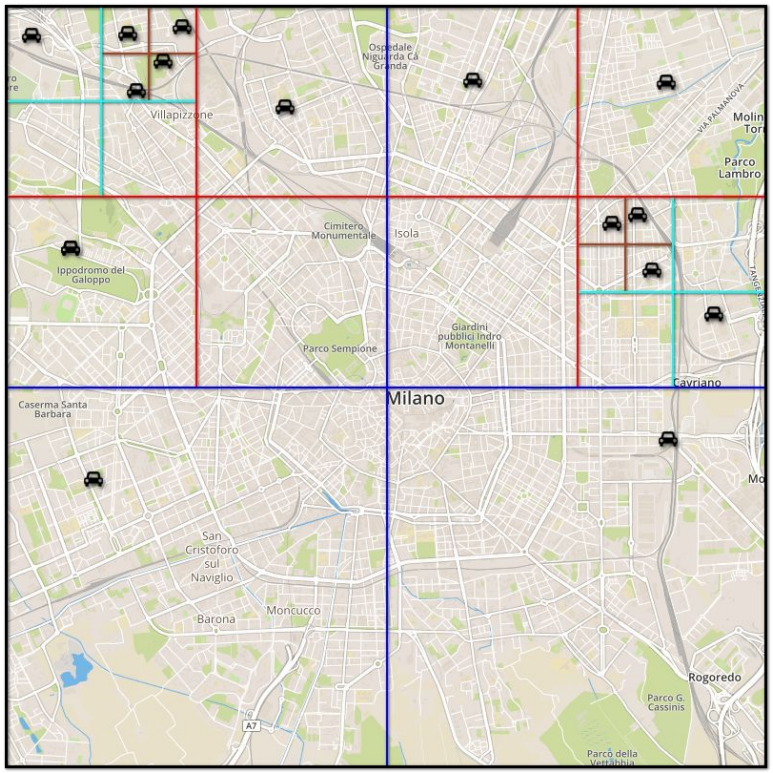
Tiers and layers

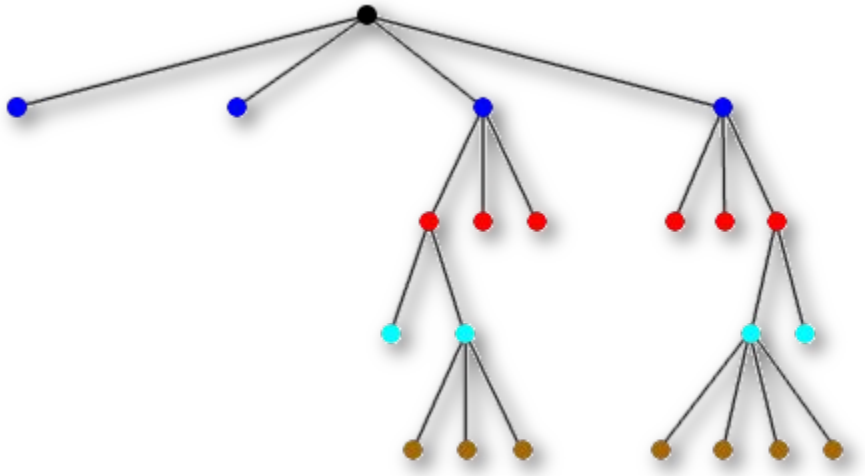
Layers:

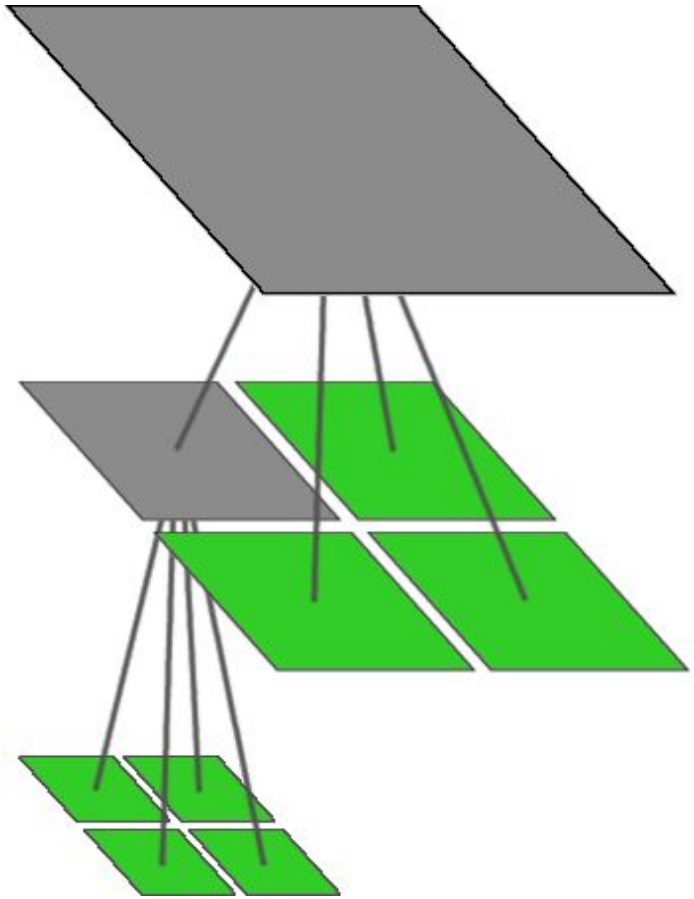
- Presentation layer
- Application layer
- Data layer



Quad trees


Ospedale Niguarda Ca Granda
Villapizzone
Parco Lambro
Cimitero Monumentale
Isola
Ippodromo del Galoppo
Giardini pubblici indro Montanelli
Parco Sempione
Cavriano
Milano
Caserma Santa Barbara
San Cristoforo sul Naviglio
Barona
Moncucco
Rogoredo
Parco G. Cassinis
A7

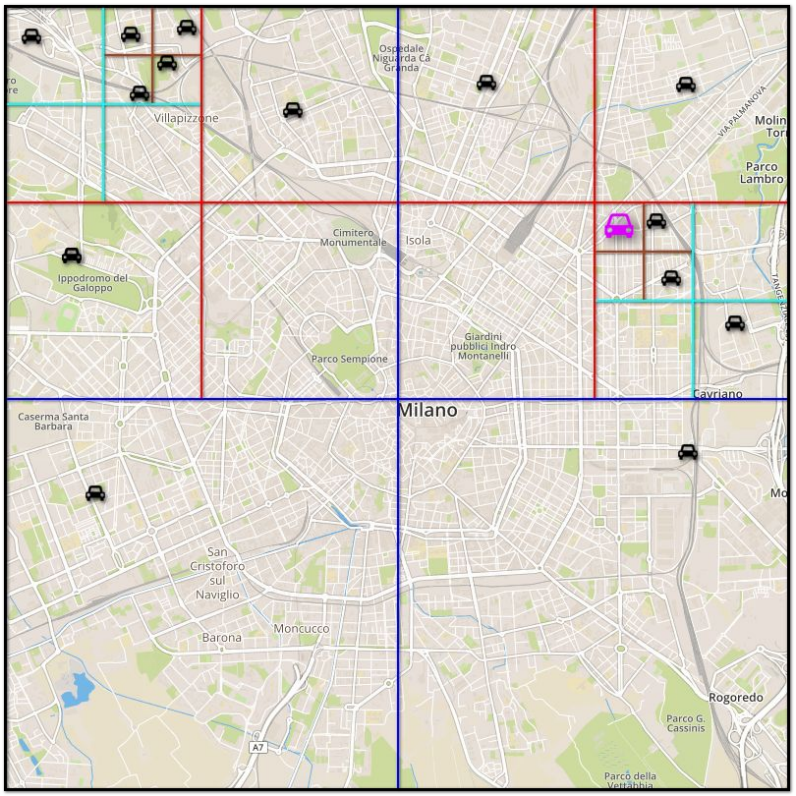


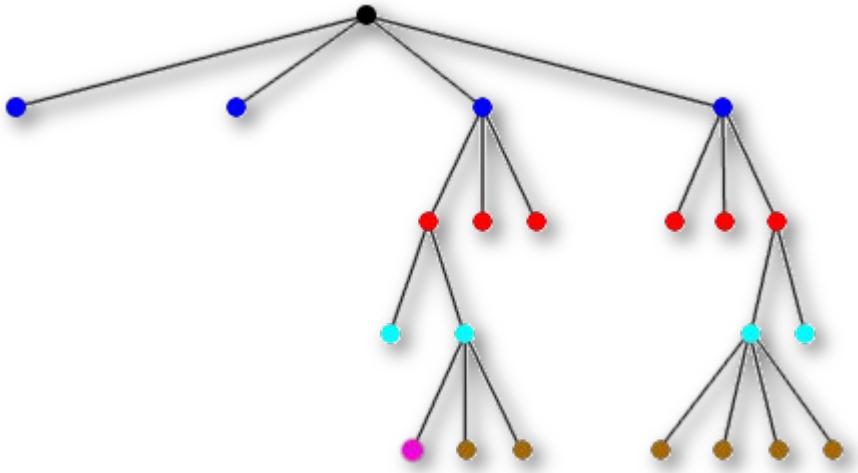


Insert

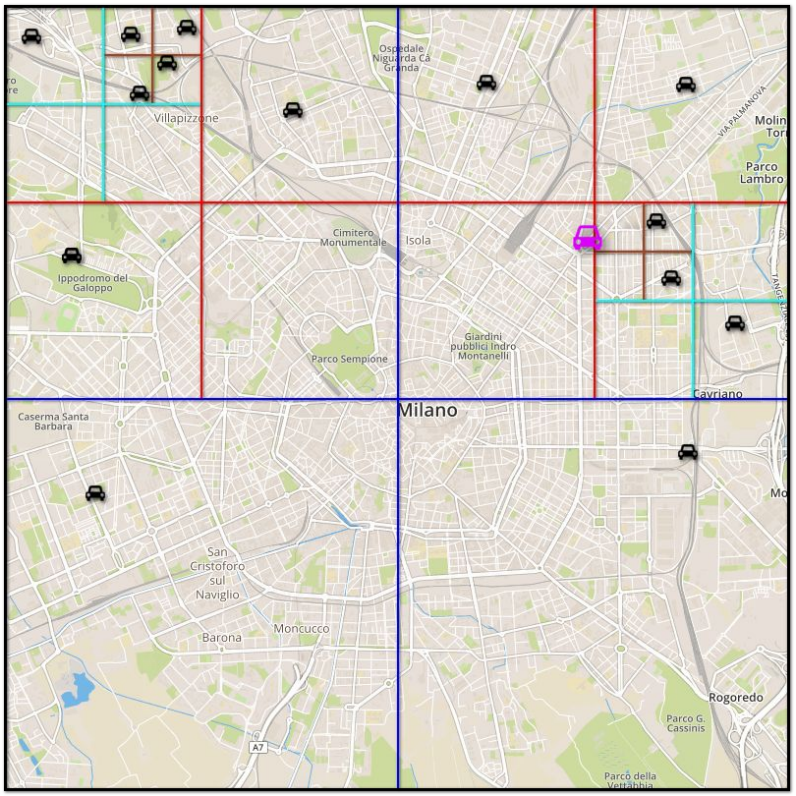
```
procedure QuadInsert(C, X)
    // Try to insert car C at node X in quadtree
    // By construction, each leaf will contain either
    // 1 or 0 cars
    if the subtree rooted at X has more than 1 car then
       determine in which child Y of node X the car C is in

       // Y is either the top left, top right, bottom left,
       // or the bottom right "quadrant"

       QuadInsert(C, Y)

    else if the subtree rooted at X has exactly 1 car then
       // X is a leaf
       create four children for node X in the Quadtree

       // X is not a leaf anymore
       move the car in X into the child in which it lies


       let Y be child in which car C lies
       QuadInsert(C, Y)
    else
       // X is a leaf
       store car C in node X
    endif

end
```

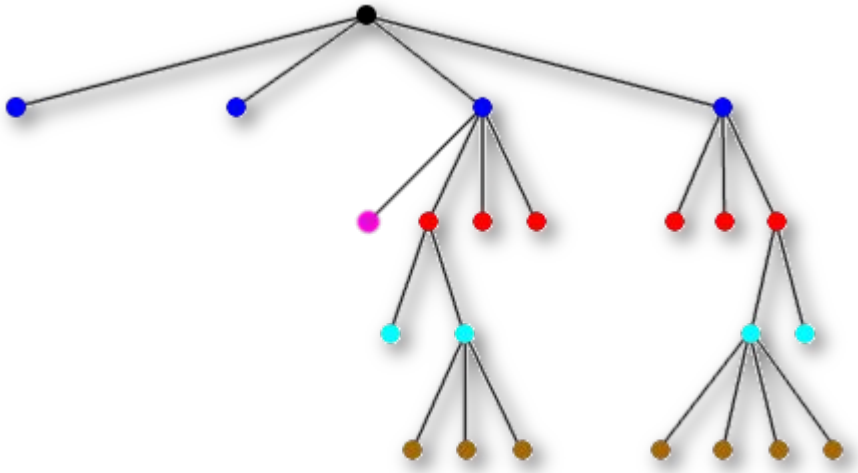
Update car position



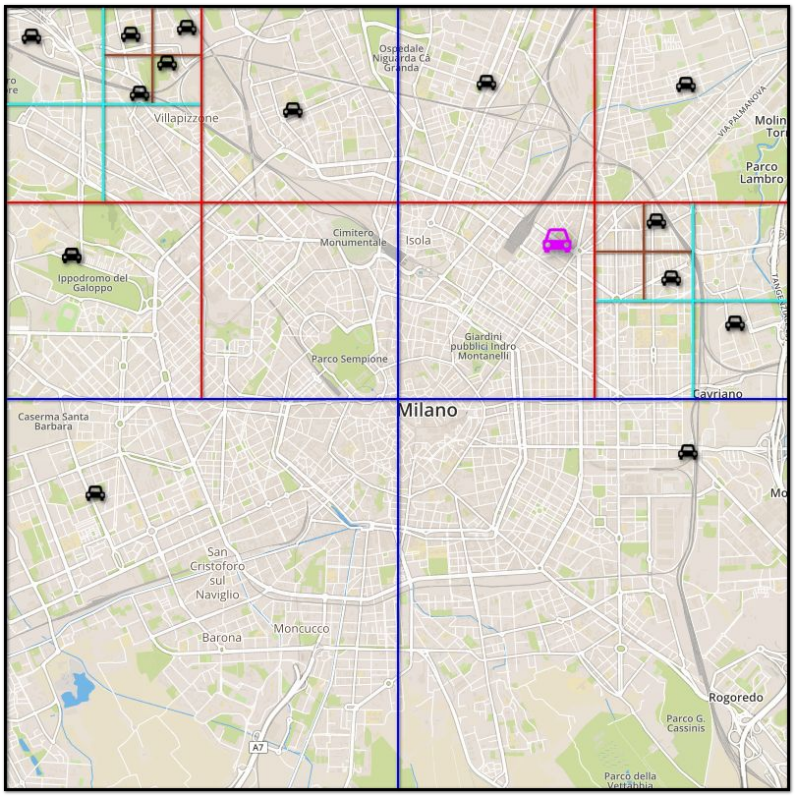


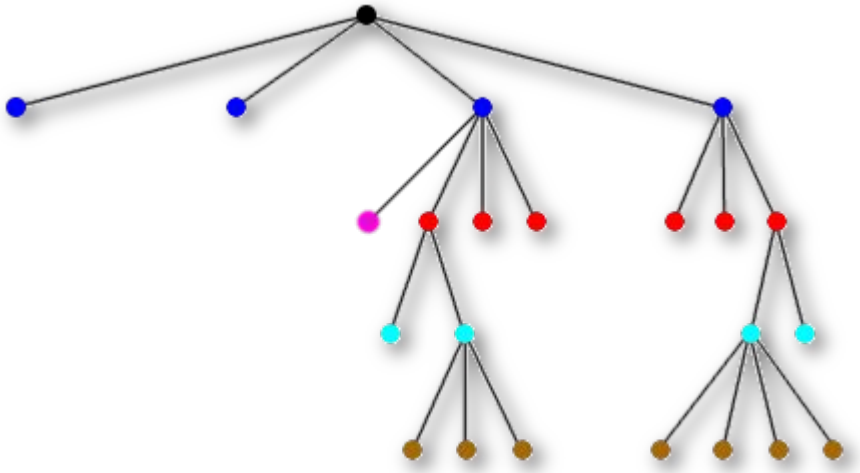
Update car position



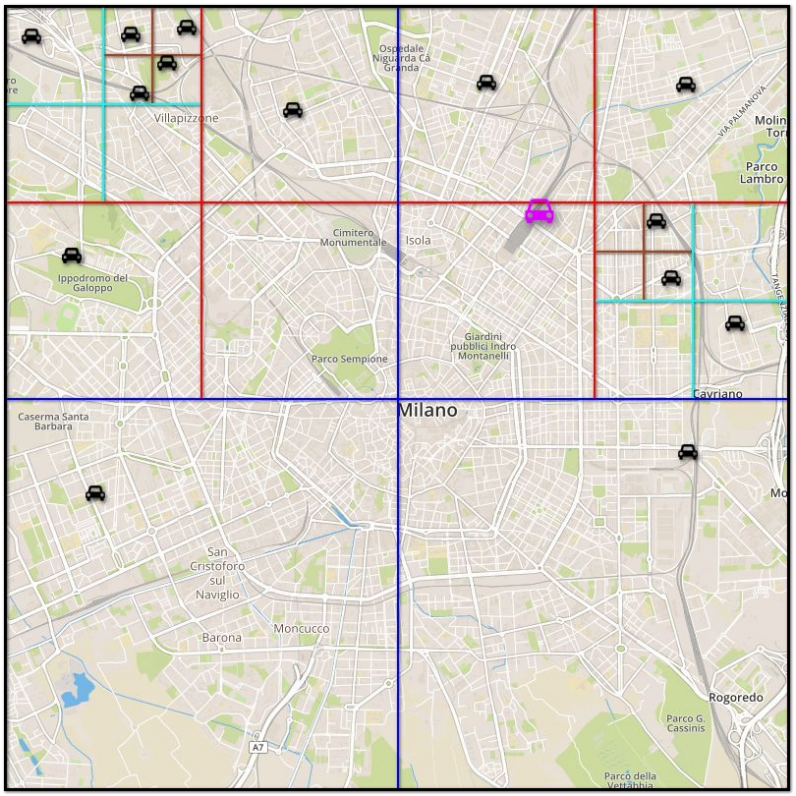


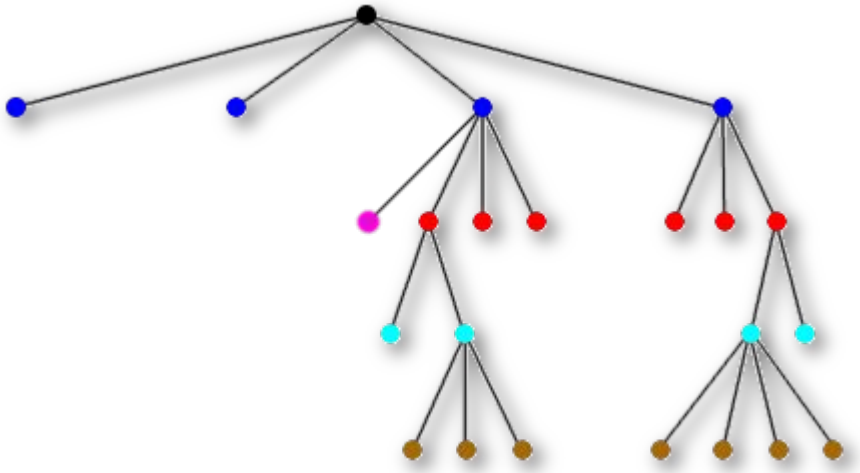
Update car position



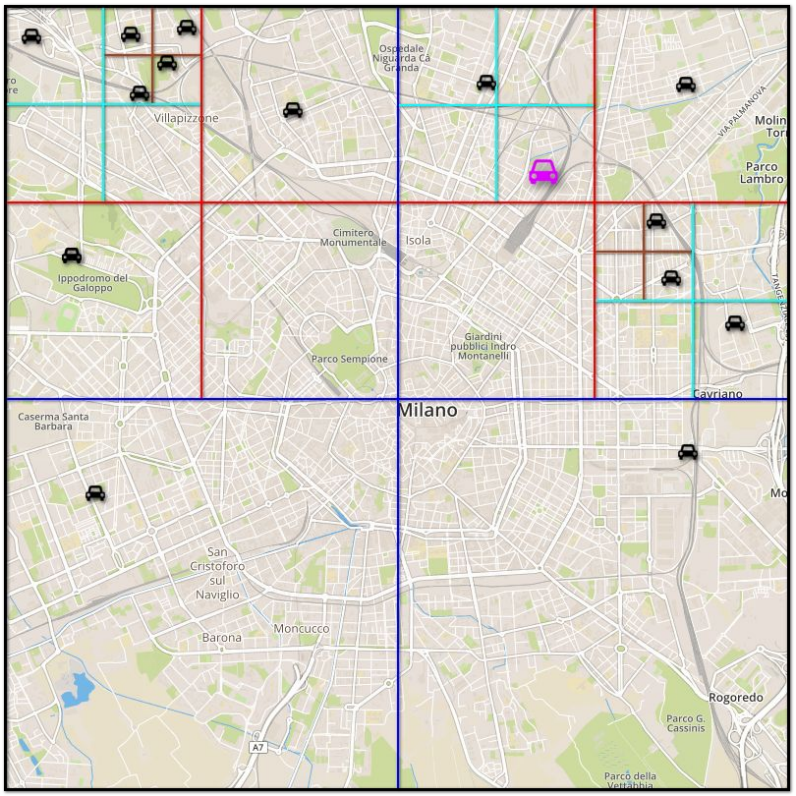


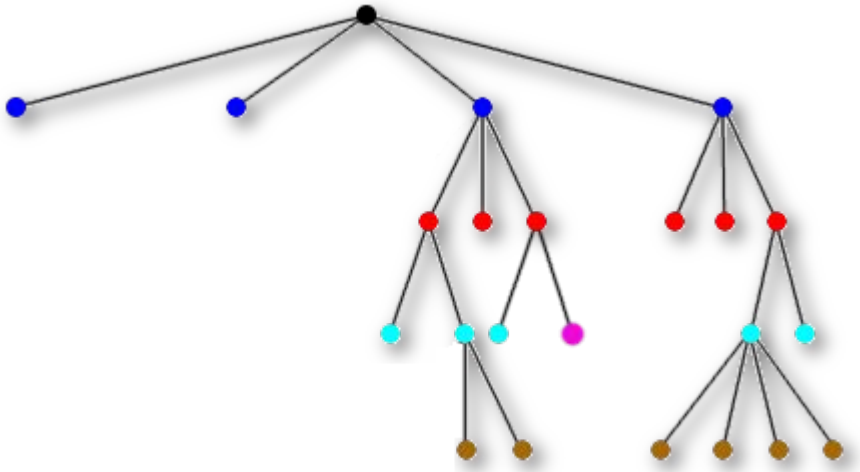
Update car position



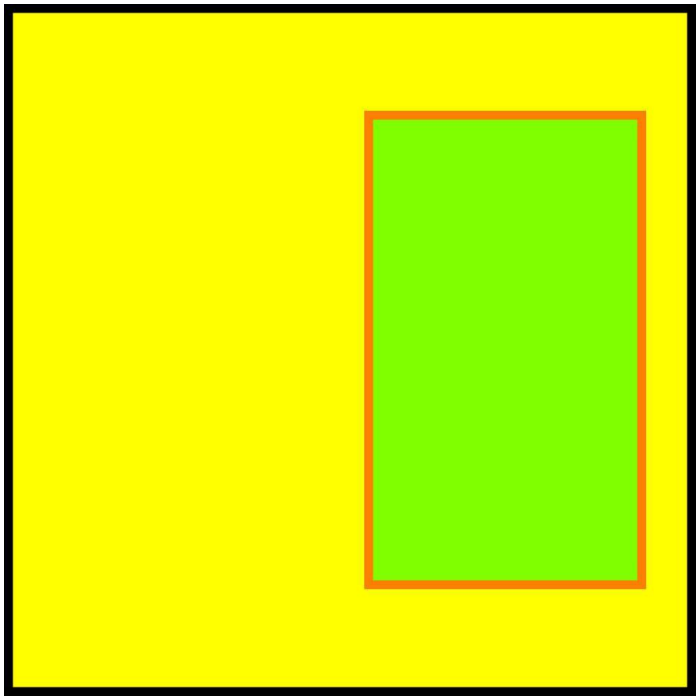


Update car position

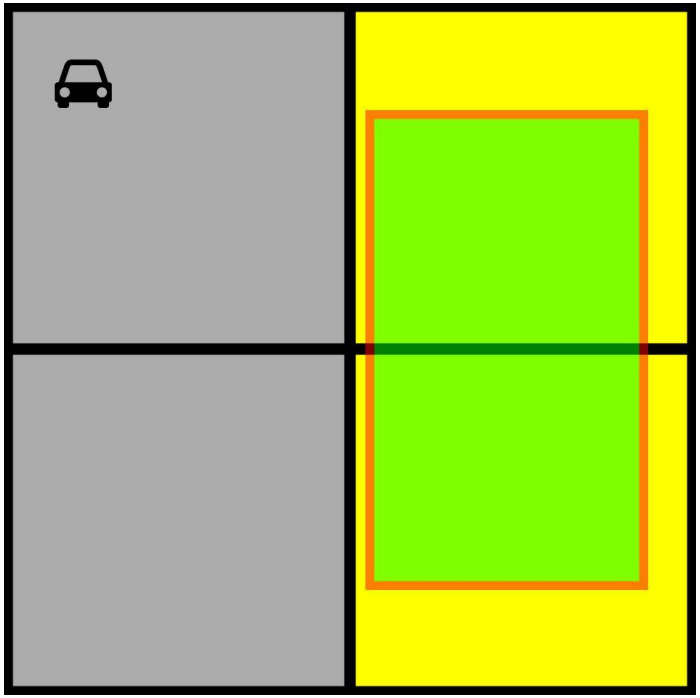




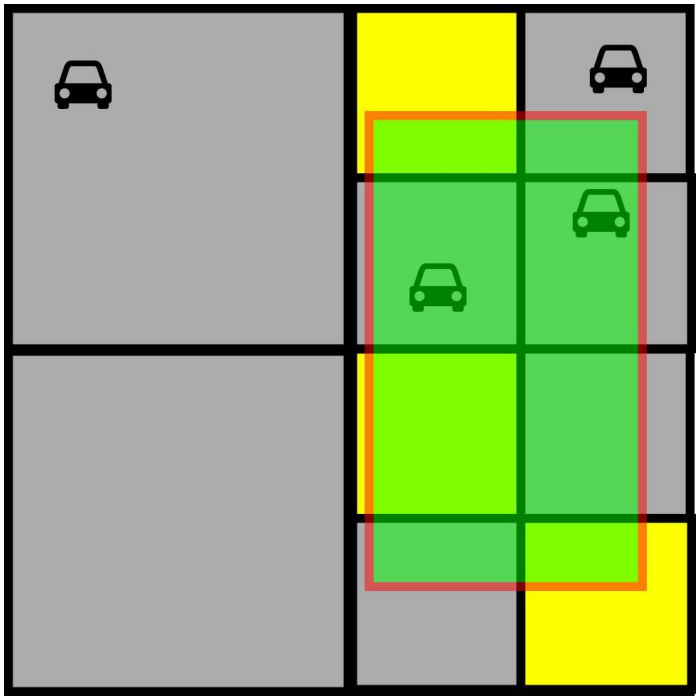
List all cars in a given area – Step 1



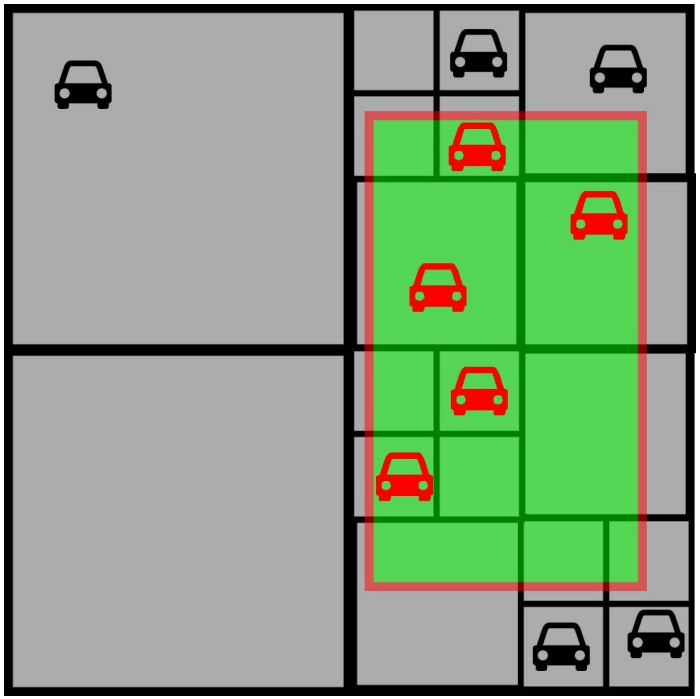
List all cars in a given area – Step 2



List all cars in a given area – Step 3



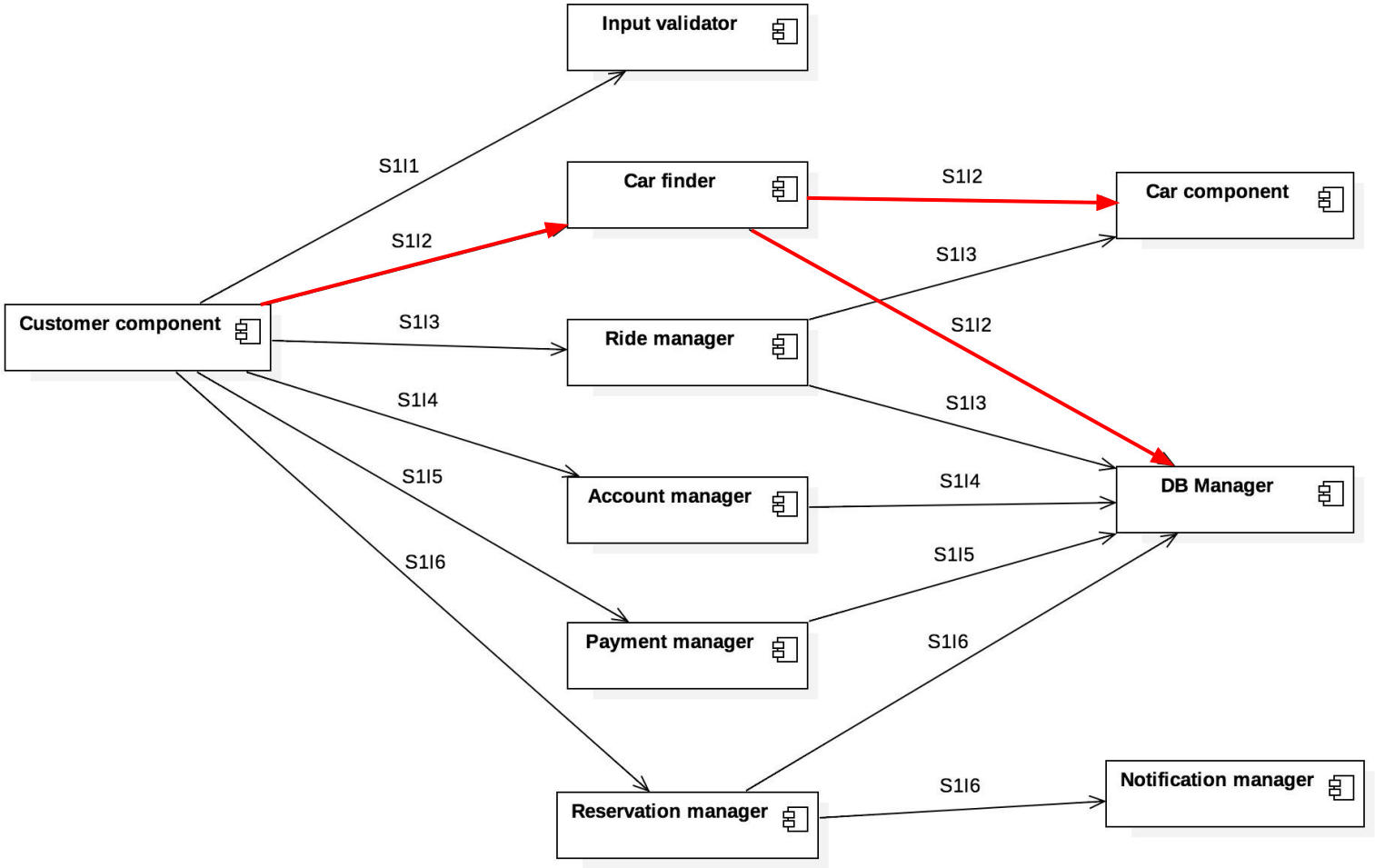
List all cars in a given area – Step 4



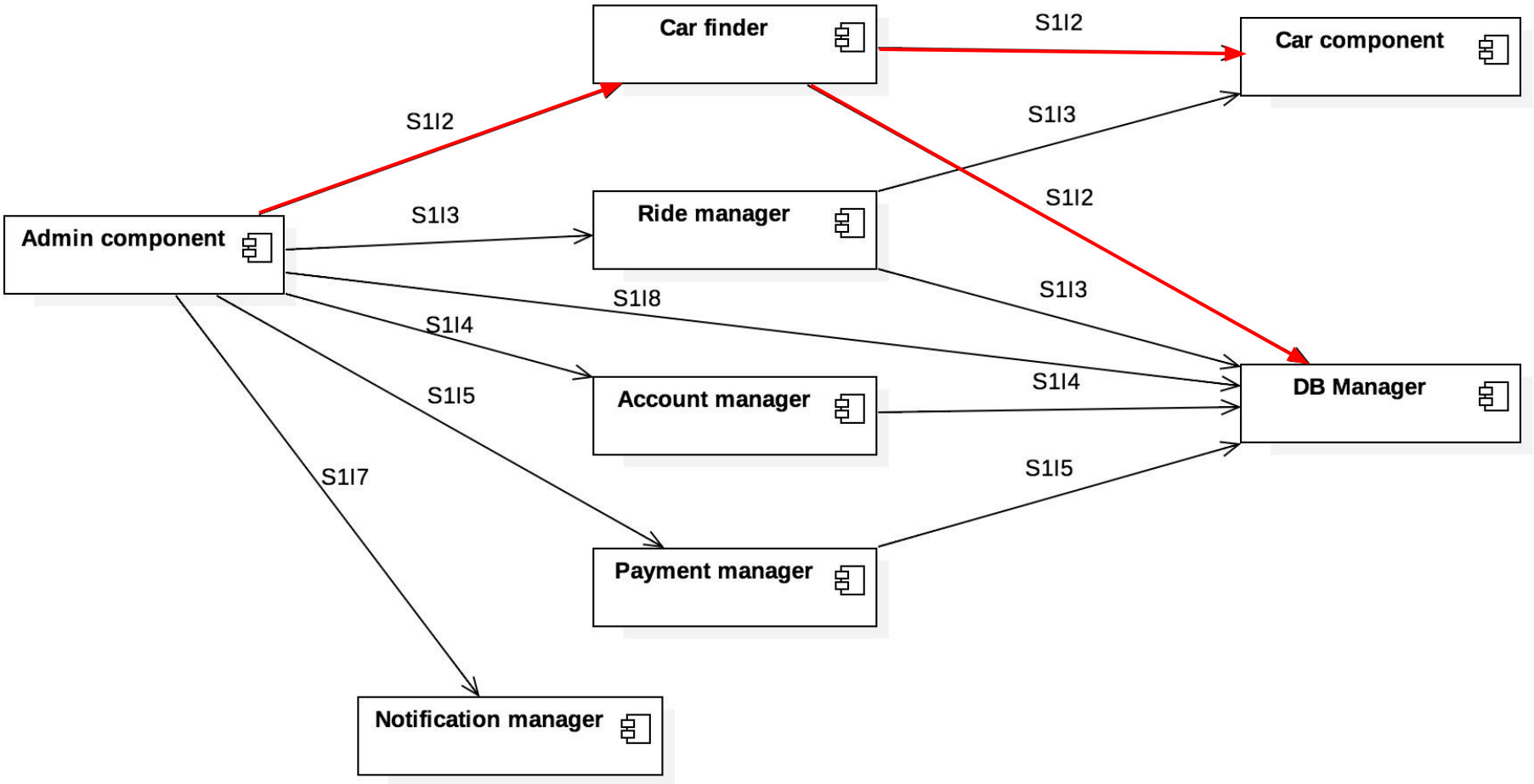
List all cars in a given area

```
procedure QuadList(S, X)

    // S is the “query square”, that is: the interesting area
    // X is the root node, initially is set to root
    answer = [] // empty list

    Y = S ∩ area(X) // intersection between S and the area of X

    if Y ≠ ∅ // non-empty intersection

       if X is a leaf

          answer += [all cars that are inside Y]   // 0 or 1 car

       else

          answer += QuadList(S, top left of X)
          answer += QuadList(S, top right of X)
          answer += QuadList(S, bottom left of X)
          answer += QuadList(S, bottom right of X)

       endif

    endif

    return answer
```

Integration test



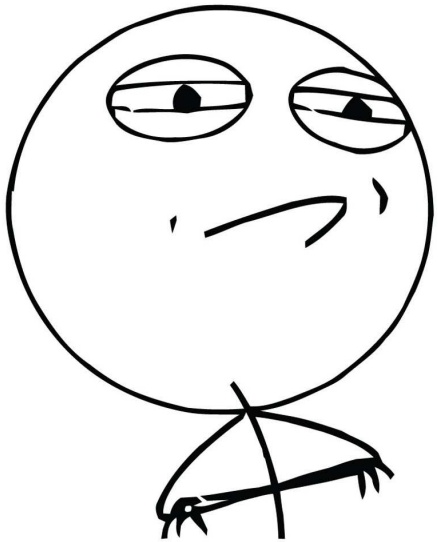
Integration test



Integration test

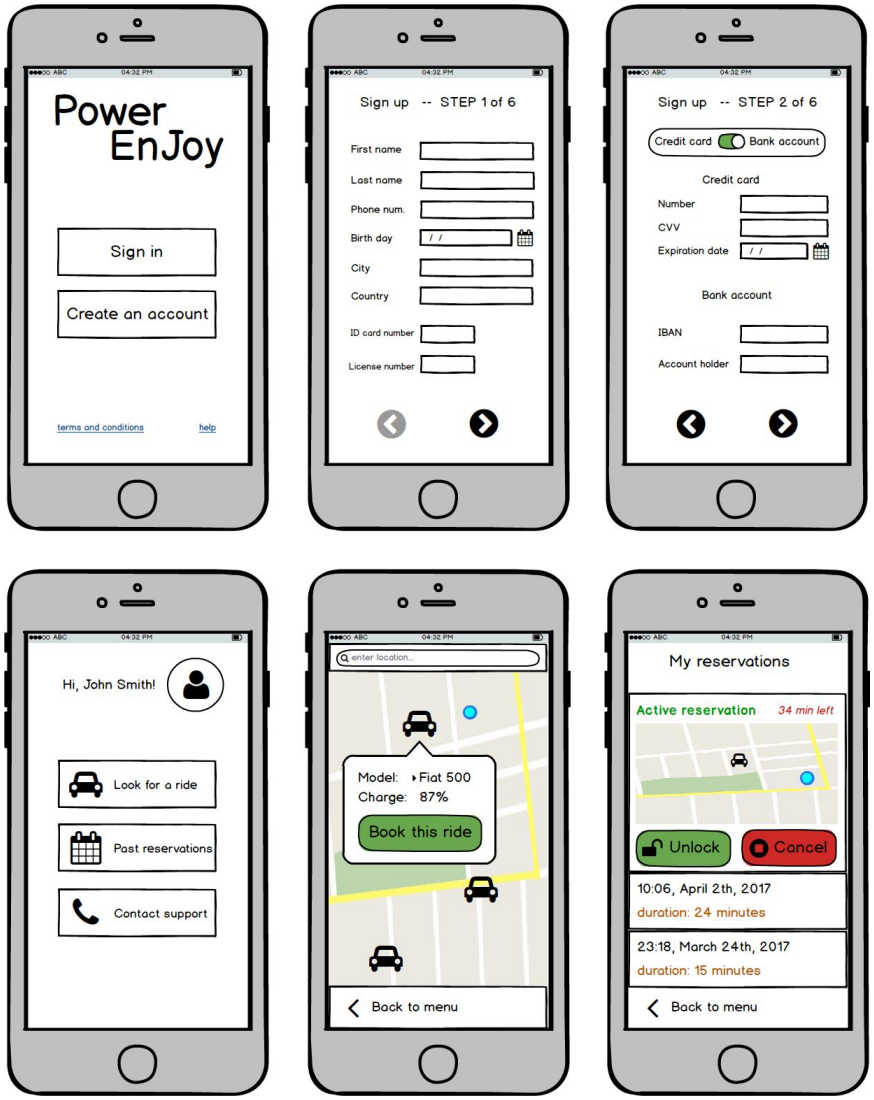
Test case identifier	**S1I2-T1**
Test item(s)	Customer comp., Admin comp., Car finder → Car comp., DB manager
Input specification	Typical car search query, either via address or specific position with increasing range from 10 meters up to 50 km in suitable growing steps.
Output specification	Check if the components access the proper elements in the database and in PE Car stub.
Environmental needs	PE Car stub

We're done.

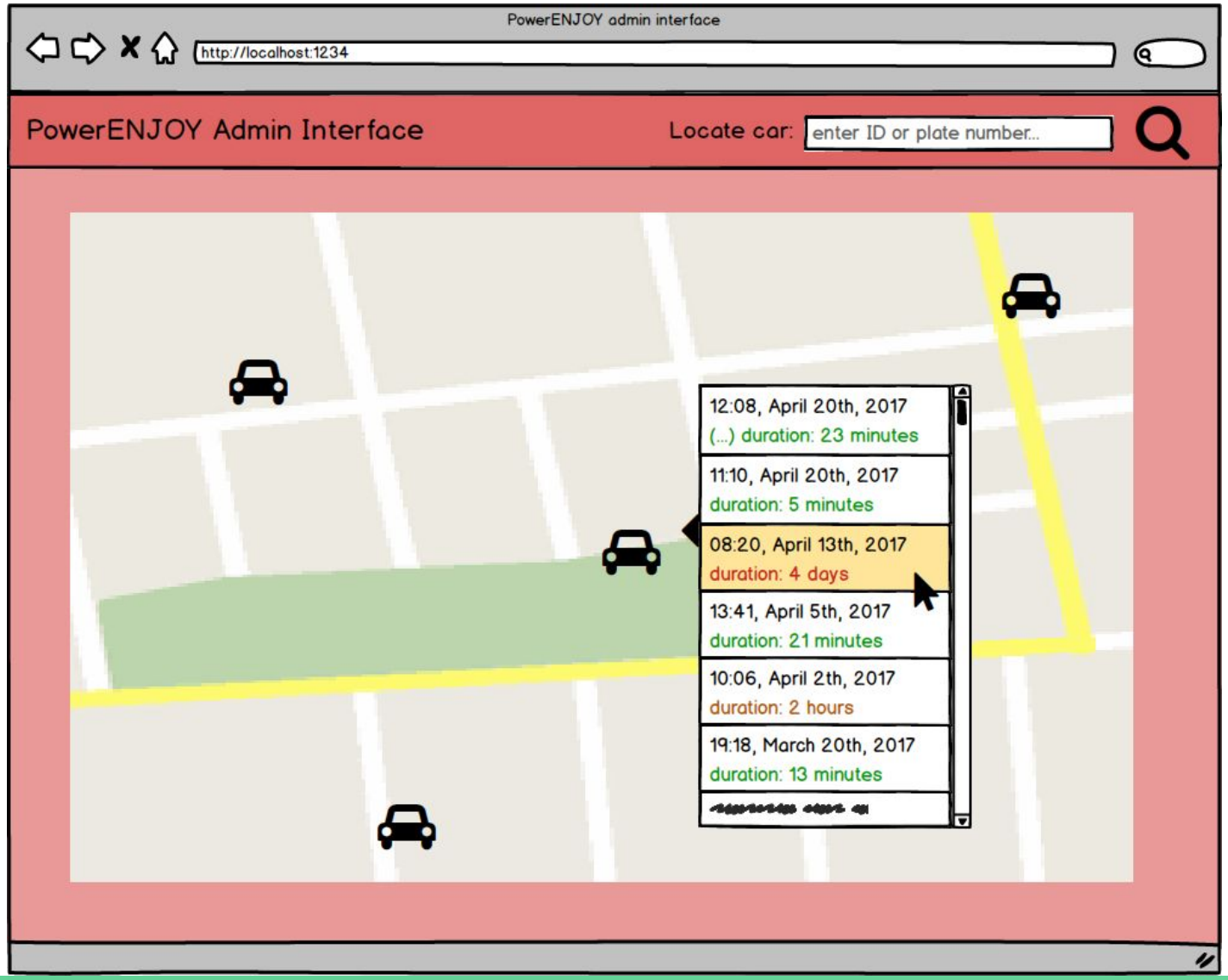


Any questions?

User interfaces



User interfaces



Sequence diagram