SmartCityAdvisor

POLITECNICO

MILANO 1863

ASSIGNMENT 1 – RASD

Requirement Analysis and Specification Document

A Software Engineering II Project,

made by Luca Favano 899224

V1.0 - 07/06/2018

Table of Contents:

1 - Introduction

1.1 Purpose

This Document is the Requirement Analysis and Specification Document thus its goals are to analyze the needs of the customer in order to model the system while describing it in terms of functional and non-functional requirements. It's a useful document to the developers since it can highlight the limits and constraints of the software by showing typical use cases.

1.2 Scope

SmartCityAdvisor is a service thought specially for the city of Milan that based on several sensors and actuators can limit the traffic throughout the city. Such service is mostly useful to contain the level of CO_2 in the air but may also be used in case of special situations that require it.

Such service is possible thanks to sensors that track the level of CO2 in the air, the number of cars that enter in the city center and the available spots in each parking lot of the city.

SmartCityAdvisor implemented a sophisticated actuators system that is able to divert the traffic away from the city center when needed, this is obtained by controlling the traffic lights and alerting the citizens through the mobile app and big displays that can be seen when entering the city.

1.2.1 Goals

[G1] Limit the traffic accordingly to current CO_2 levels in Milan.

[G2] Warns the citizens of a change of the current status

[G3] Provide users with up-to-date information retrieved from the sensors

[G4] Manage the traffic accordingly to special events taking place

[G5] Simplify the process of finding and reserving an available parking spot

[G6] Providing Superusers the possibility to manage events and emergencies

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Sensor: A component of that communicates with the system at regular intervals providing useful information for the system to work correctly.
- Actuator: A component that allows the system to have an impact on the environment based on choices that can be taken automatically or manually,

- City Center: An area of Milan which characteristics need to be preserved and kept constant over the time.
- CO_2 Level: Current amount of CO2 particles detected by the sensor and measured in ppm (particles per million).
- Superuser: A user of the app that has higher privileges compared to a "normal" user which is a common citizen that downloaded the app.

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document
- UML: Unified Modeling Language
- IEEE: Institute of Electrical and Electronic Engineers
- DBMS: Database Management System
- HTTP: Hypertext Transfer Protocol

1.3.3 Abbreviations

- [Gn]: n- Goal
- [Dn]: n- Domain Assumption
- [Rn]: n- Functional Requirement

1.4 Revision History

- V0.2: "First Version Uploaded", Released on 02/05/2018
- V0.9: "Complete structure that needs refinements", Released on 03/05/2018
- V1.0: "Complete Version", Released on 07/06/2018

1.5 Reference Documents

- Specification Document: "Assignment-for-the-second-project.pdf"
- Alloy Specification found on beep and google
- Past Project Examples taken from beep
- IEEE Recommended Practice for Software Requirements Specifications.

1.6 Document Structure

This document will be structured into five different sections:

- Section 1: The introduction, a brief description of the project, the terms that will be used and some useful knowledge.
- Section 2: Overall Description, more specific information about the service that will be provided and its functions, including its constraints and limits.
- Section 3: Specific Requirements, the complete requirements and constraints of the project. Both functional and non-functional requirements are discussed together with the Design Constraints and the attributes of the system.
- Section 4: Formal Analysis, a representation of the structure of the project obtained using the Alloy language.
- Section 5: Effort Spent + References, containing all the information used while writing the document and the time spent doing it.

2 - Overall Description

2.1 Product Perspective

The SmartCityAdvisor system will be developed will be developed from scratch, it will be server-sided and will also include an anonymous app and website.
This service can then be used on every mobile phone or computer with an active internet connection.

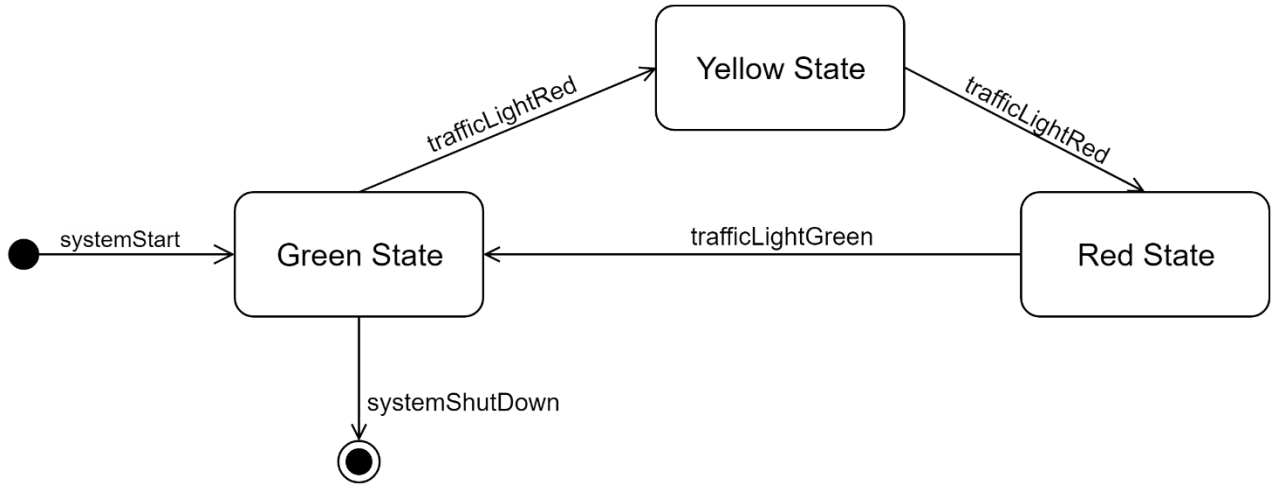
The system will interface with sensors, actuators, displays and the app while the app will just show the results to the customers without them needing to interact with it, if not to sort out the relevant results. The GPS in the mobile version could be used to obtain better and faster results on the go.

The displays won't show all the information but only the ones most relevant to the road the displays are mounted on or the ones immediately linked to it.

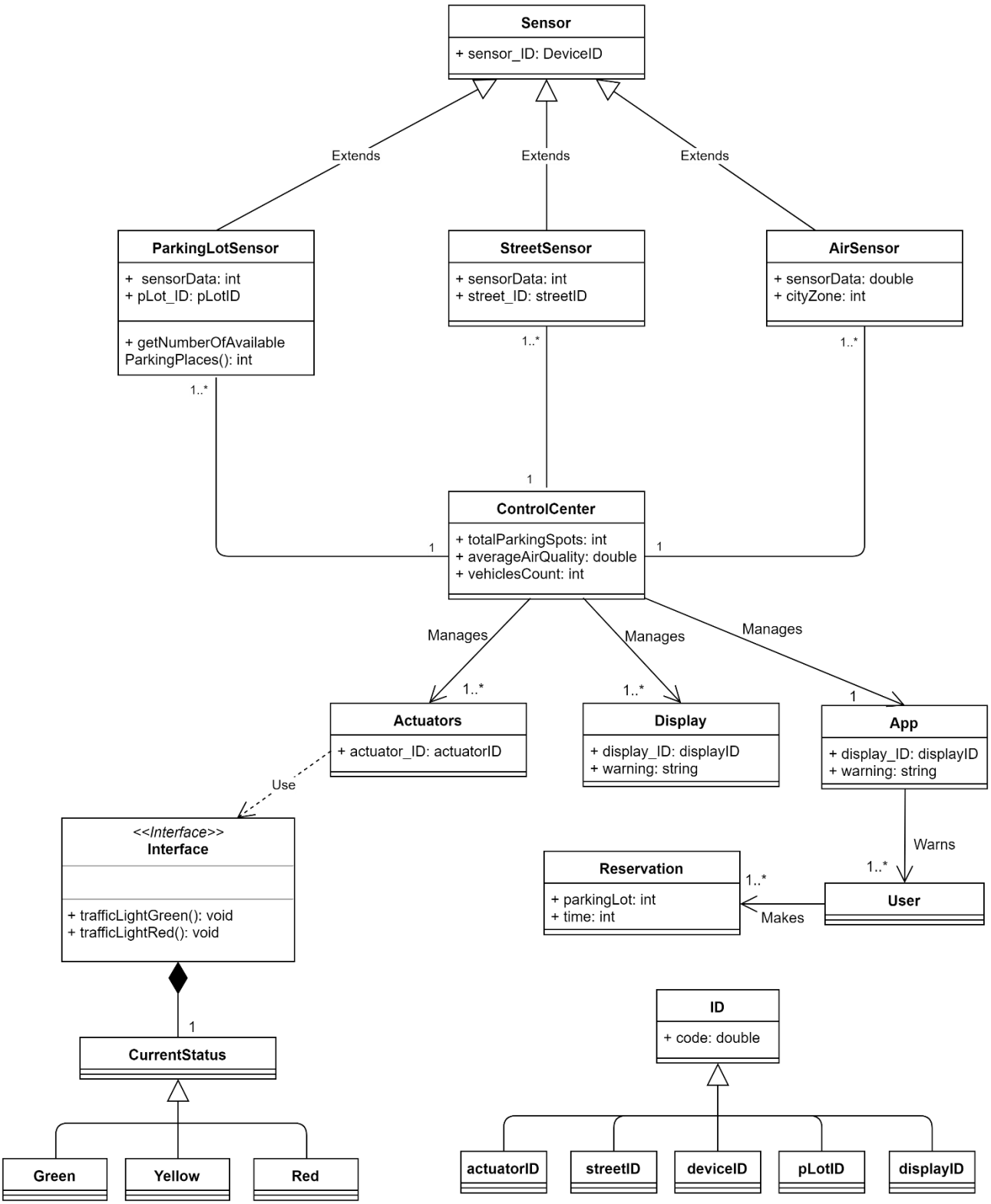
Having information about the CO_2 level and the availability of parking spots may also be interesting for the citizens and could be further used in future developments of this system.

2.1.1 Statechart

Here is the statechart related to the possible states that the system can assume:



2.1.2 Class Diagram



2.2 Product Functions

For this project we are asked to develop SmartCityAdvisor, a system that includes many sensors that in the beginning won't be used extensively, in fact we are also asked to contribute to the system by suggesting new functionalities that should be implemented.

2.2.1 CO_2-Based Traffic Management

The main function that the system should offer is a traffic management system based on the CO_2 level readings in the air that are taken by the sensors.

In this case the sensors send their data to the control center every two hours, when the data is found to be higher than a chosen threshold the traffic management system will change its status and start transitioning from a "green" status to a "red" one.
When in the red status the traffic will be deviated from the city center to other roads thanks to the actuators connected to the traffic lights.

This system also includes a middle status that is the "yellow" one, when in such status the citizen will be warned of the imminent situation via the mounted displays and the app. This status could go on approximately from 30 to 60 minutes based on the current traffic situation, for an emergency this status could be skipped completely.

2.2.2 Complete Traffic Management

Given the availability of so many useful sensors an easy to identify function would be to make the system more complex and thus better.

Based on the other two sensors we could easily make an esteem of the cars that are currently circulating, in fact we have both the number of cars that enter the city center and the number of available parking spots, which is strictly linked to the number of parked cars.

This way we can't obtain an exact number but still give a concrete esteem. Based on this extra information we can make better choices on when to limit the traffic in the city center, especially when is it a good time to restore the "green" status.

2.2.3 Summarize and Provide System Data

By making the system data available to the citizens via the app we may provide useful services, on top of them the searching of a parking spot will be easier thanks to knowing the availability of spots for every parking lot of the city center.

All the data read by the sensors will then be aggregated and then made publicly accessible on the app.

2.2.4 Others (not implemented)

Another possible function would be to suggest the best alternative solution to the users in case of a sudden change of the system status, for example which public transport (if any) would allow them to arrive on time to their destination.

This last function won't be taken into consideration in this project due to the project group limited dimension.

2.3 User Characteristics

In this section will be discussed the actors that may interact with the system.

2.3.1 Actors

- Citizen: A person that is circulating in the streets of Milan and gets some basic information from the Display mounted around the city.
- User: A person that downloaded SmartCityAdvisor app and is able to retrieve more in-deep information about the current situations of the streets.
- SuperUser: A special user that signed in the app with special credentials and is now able to use extra functions such as making a new request for a special event happening in the future
- Control Management Worker: A person that is allowed to interact directly with the system to modify the current status in case of problems or emergency situations, send notifications to users or insert events in the DB

2.4 Assumptions, Dependencies and Constraints

2.4.1 Assumptions

For this system to works correctly we need to assume the good behavior of every citizen, it is implied that users should pay attention to the displays mounted on the streets or try to use the app so that they will have an adequate time to modify their plans and won't cause troubles to the circulation.

Another aspect that is assumed is that a red traffic light should be seen as a rule that can't be violated.

Lastly, we assume that the sensors make their readings in an anonymous way so that the privacy of the citizens won't be violated.

The following assumptions are also necessary:

- [D1] Sensors work with an adequate precision, providing readings at the right times.
- [D2] In case of problems with sensors, some backups are always available until the broken ones are fixed.
- [D3] Sensors are provided in the right number, checked by their ID and won't provide duplicated data.
- [D4] Actuators control the traffic lights in the wanted main intersections and not the ones in any different location.
- [D5] The "yellow" status is changed automatically after a given amount of time or manually by a worker in case of exceptions.

- [D6] Notifications to the users are sent and received immediately
- [D7] Superusers can't share their credentials with anyone else

2.4.2 Dependencies

No dependencies have been spotted up to this version of the system, but this may be updated in the future.

2.4.3 Constraints

Even if the app will be very light it still have some hardware limitations described as follows:

- The systems used to open the app shouldn't be too old, for a phone having iOS or an Android system is enough while a computer should use a modern browser.
- An internet connection is always necessary to retrieve the latest information from the system, it is care of the citizens to make sure the information they are reading are up-to-date and not some cached data.

Since the information provided by our system are very important for other apps to work properly (e.g. Google Maps) a public interface should be provided in the near future.

3 – Specific Requirements

Let's now analyze all the aspects seen in the section number two, such aspects might become useful to the development team.

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interfaces are strictly related to the app, which include only few goals of the system, while other goals that actually manage the traffic won't be shown in this section.

The following images represent the wireframes of some of the main pages of the SmartCityAdvisor app.

The obtained app is very light and simple, easy to use and thus user-friendly. Both the versions have the same functions with the same structure. It's divided into 4 main sections: Home, News, Data and Sign-in.

All the information needed can be obtained with few clicks and they are provided by the system automatically.

Other extra sections are omitted but could easily be added, for example the "contact us", "about us", "FAQ" sections.

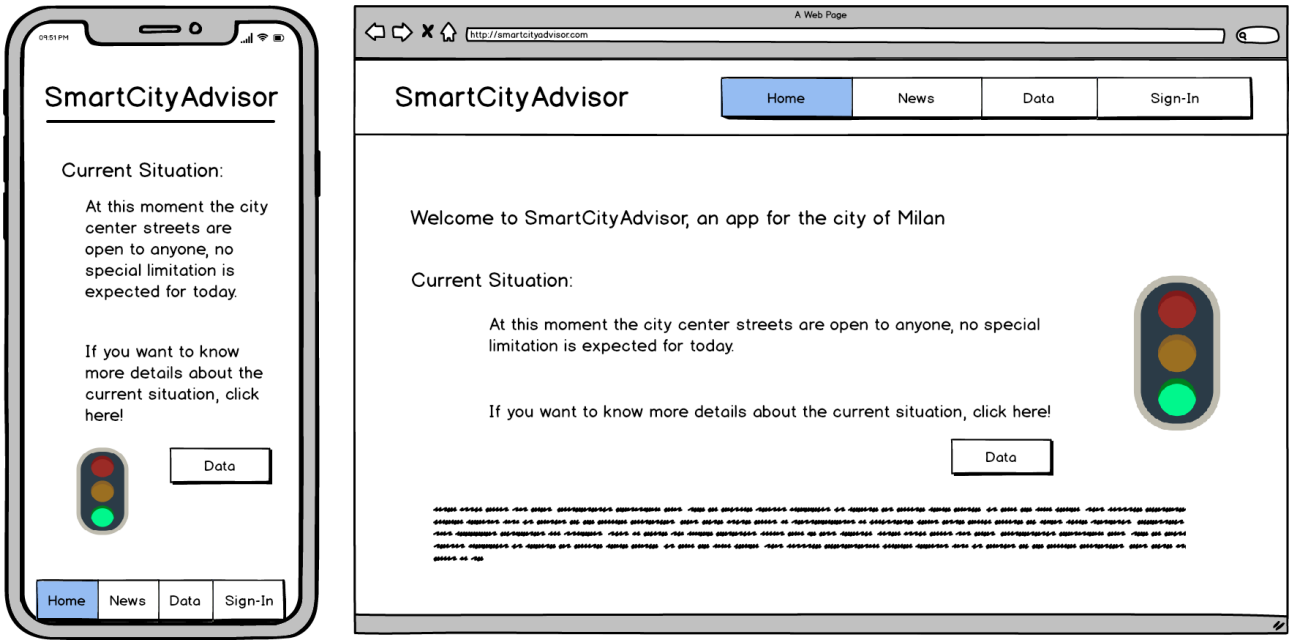


Figure 1: The landing page of the app shows the current traffic situation

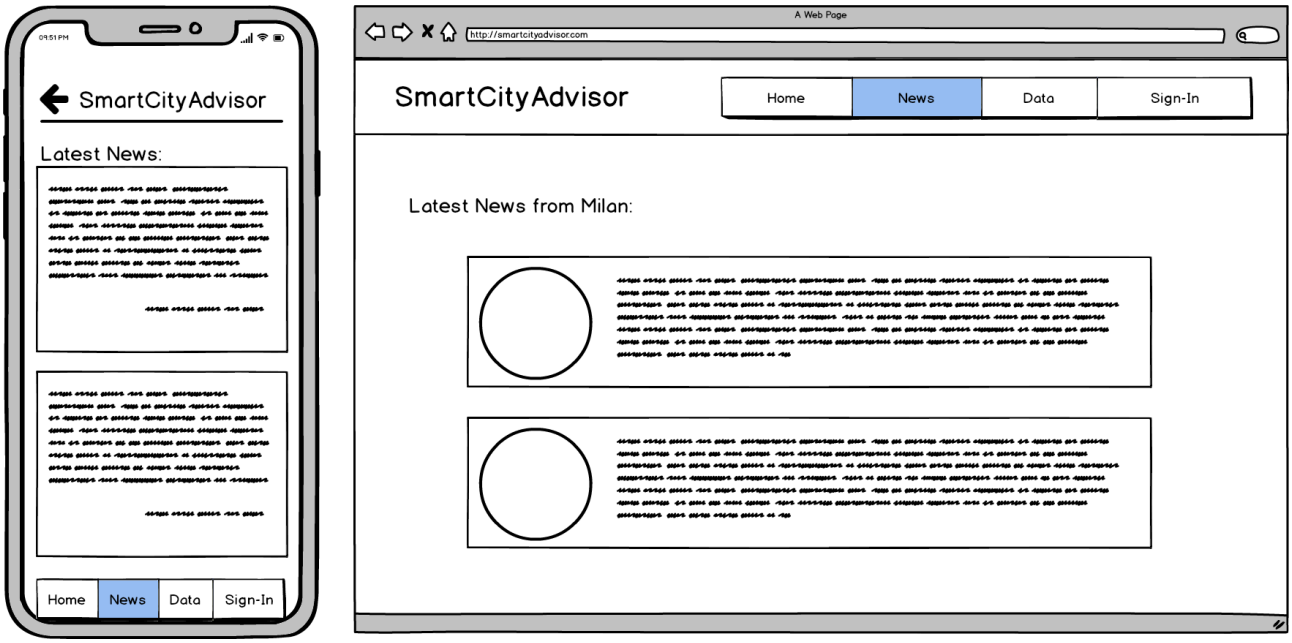


Figure 2: The news section shows the main events coming soon (e.g. A VIP that's coming to visit Milan, such event is known well before it takes place and citizens should be warned)

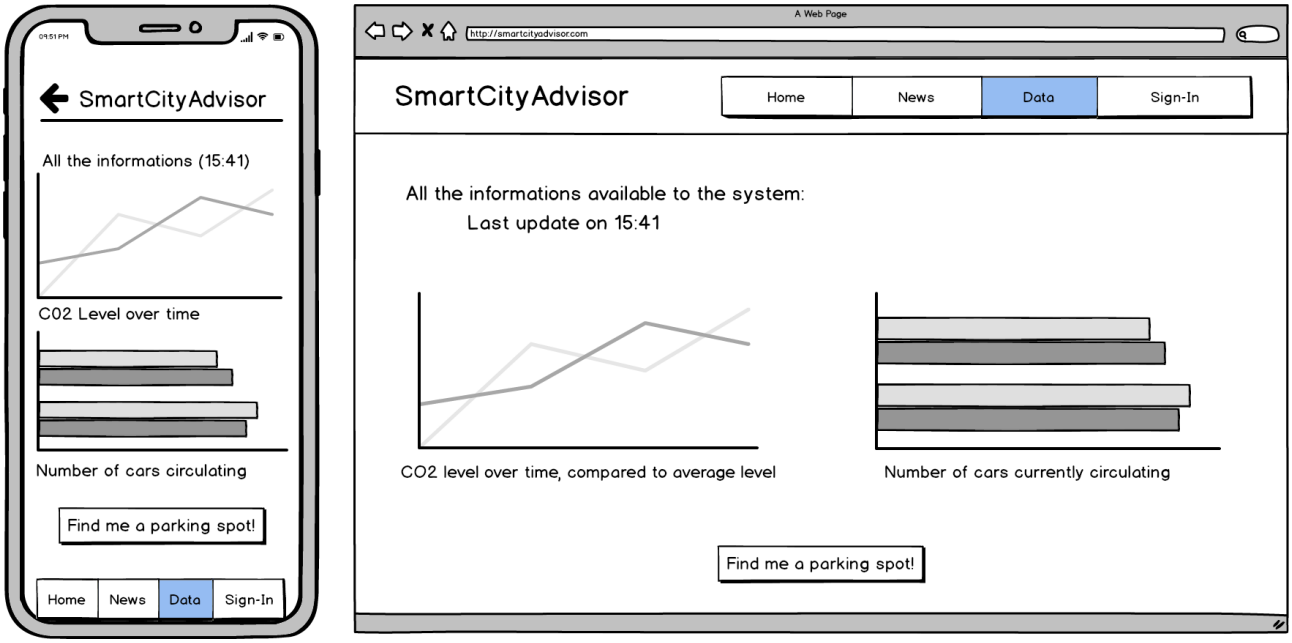


Figure 3: The data section shows the history of the sensors readings

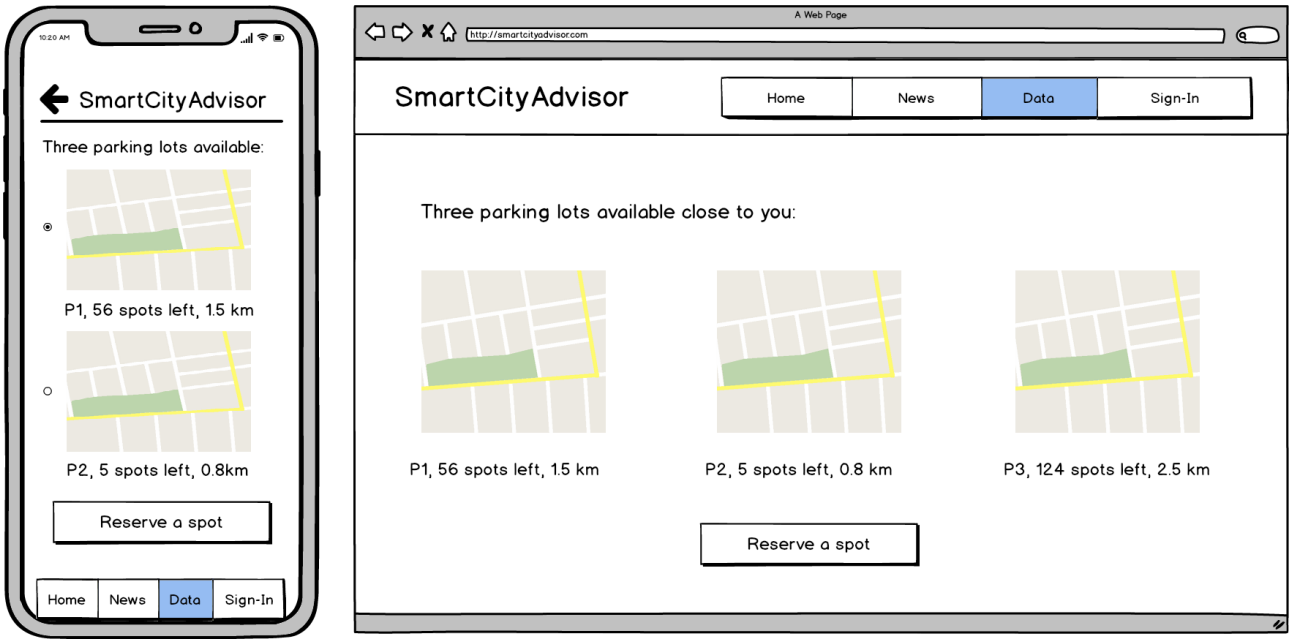


Figure 4: The data section also includes a function to find the nearest available parking spots, here some options are given, the user can then choose the one he prefers.

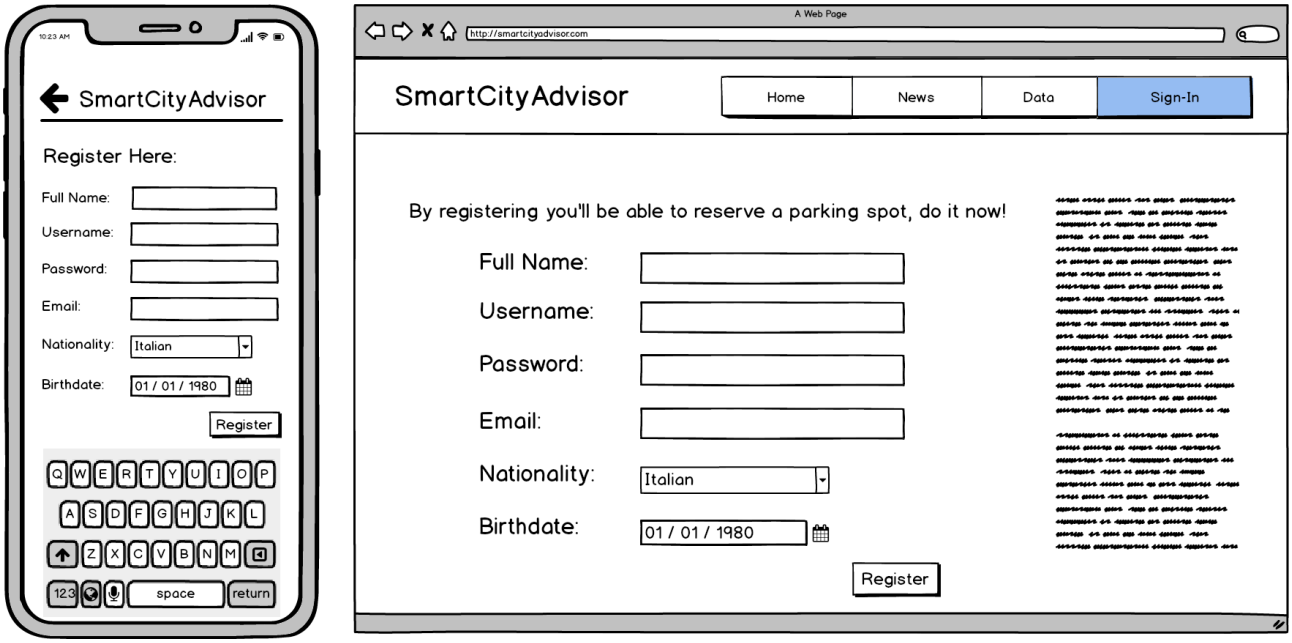


Figure 5: To reserve a parking spot the user has to register by filling a short form, by doing this he will become a registered user but not a superuser.

Figure 6: The sign-in section allows superuser to login into their special accounts, by inserting username and password

Figure 7: Once that a superuser logon onto the website they have access to a special set of functions that directly interact with the system

3.1.2 Hardware Interfaces

The application could run on any device with an internet connection, a GPS is also preferred (especially for the “find me a parking spot” function) but not required. Possible devices would be iOS phones, Android smartphones, tablets and Computers thanks to the responsivity of the developed app.

3.1.3 Software Interfaces

The system depends heavily on server-side operations that achieve the traffic management.

Popular APIs could be used to reduce the development cost, google maps API could easily be integrated as shown in the wireframes. This API could also be useful to retrieve real-time traffic information that could be integrated in our app.

To suggest an alternative path in case of an emergency we also would need the APIs offered by the Milan companies that provide public transportation services.

Some software requirements may be Android 4.0 or higher and iOS 7 or Higher.

3.1.4 Communication Interfaces

To communicate and send notification to the app users we use HTTP and XMPP protocols in order to send messages to GCM server, up to 1000 messages with a single HTTP request

3.2 Functional requirements

3.2.1 Requirements Definition

[G1] Limit the traffic accordingly to current CO_2 levels in Milan.

- **[R1.1]** The system will automatically register the reports provided by the sensors
- **[R1.2]** The system will monitor the data and automatically make an action when moving after a threshold
- **[R1.3]** The system may change its traffic status in any moment, in any possible status

[G2] Warns the citizens of a change of the current status

- **[R2.1]** After a change in the system status citizens will be notified immediately and automatically by the system
- **[R2.2]** The displays text will be changed and show relevant information
- **[R2.3]** The app users will receive a notification on the app sent by the system

[G3] Provide users with up-to-date information retrieved from the sensors

- **[R3.1]** After data is received from the sensors it is categorized and added to a DB
- **[R3.2]** Relevant data is automatically loaded from the DB to the website every few hours
- **[R3.3]** The user can see the data by opening SmartCityAdvisor and navigating to the right section

[G4] Manage the traffic accordingly to special events taking place

- **[R4.1]** The user has to own superuser credentials
- **[R4.2]** The user has to login on the website using the right credentials

- **[R4.3]** The superuser has to navigate into its personal area and add a special event
- **[R4.4]** Superuser has to fill the form, the provided information will be then added to the DB
- **[R4.5]** The system will send an email to the superuser to confirm if the operation was successful

[G5] Simplify the process of finding and reserving an available parking spot

- **[R5.1]** User has to be logged in
- **[R5.2]** If the user is not logged in he needs to register by navigating to the right section and filling the forms
- **[R5.3]** The system will send a link to the email provided to conclude the registration
- **[R5.4]** The logged in user must look for a parking spot and choose which one to reserve
- **[R5.5]** The system will take the reservation and send it to the desired parking lot system
- **[R5.6]** The system will confirm the successful reservation by sending an email to the user

[G6] Providing Superusers the possibility to manage events and emergencies

- **[R6.1]** The Superuser has to be logged in
- **[R6.2]** The superuser has to reach it's personal area by navigating to it
- **[R6.3]** Special forms to report emergencies, add/remove events can be found in the personal area

3.2.2 Defining Scenarios

1) A foreign president will visit Milan:

When the appointment is confirmed and the news start spreading around the SmartCityAdvisor administrative team will take care of warning the citizens.

To do so a staff member will open the SmartCityAdvisor mobile application and move into the sign-in section with a click. After entering this section, he will insert his username and password and then submit. If the submit is successful (the credentials should be already present in the system DB) he will have access to some special features.

Special features include the option to add an event that will require the city center to be closed for a certain amount of time. After he filled the form with the required information, such information will be added to the DB and users will be notified.

2) There’s an emergency in the city center

A bad event happened in Milan and many citizens reported such situations to the authorities, based on an agreement between the police stations in Milan and the SmartCityAdvisor team, a policeman immediately warns the staff about it.

After being warned the staff may access again the “superuser” features but this time will select the “emergency” option. Such option will suddenly block the accesses to the city center (without keeping the “yellow” system state) and users will be notified.

3) I need a parking spot!

A citizen that already downloaded the SmartCityAdvisor app in the past is now traveling with his family around Milan, they are late for their appointment but unluckily they can’t find any parking spot. Their choice is to open the app, navigate to the “Data” section and press the “find a parking spot” button.

The system will automatically suggest him the nearest parking lots with available spots, the user will then choose the one nearest to the place they have to reach and drive to it.

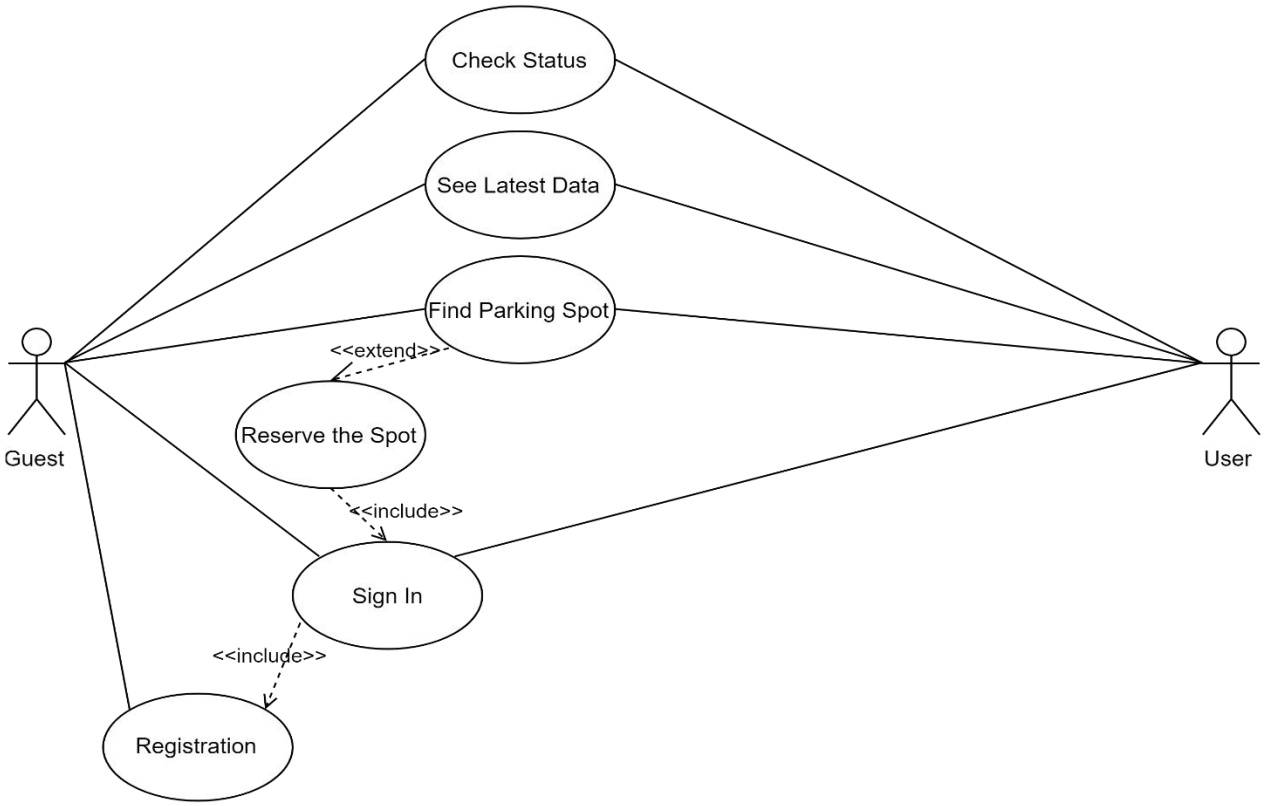
4) Why is the street closed?!

A woman is going to the city center to have some shopping with her friends, when she’s almost there she finds out that the access to the center is not possible. The woman has no idea why is this happening since she has ignored the displays mounted on the street. Luckily she remembers that a friend suggested her this useful app called SmartCityAdvisor, she immediately download the app in the homepage learns that the city center is closed.

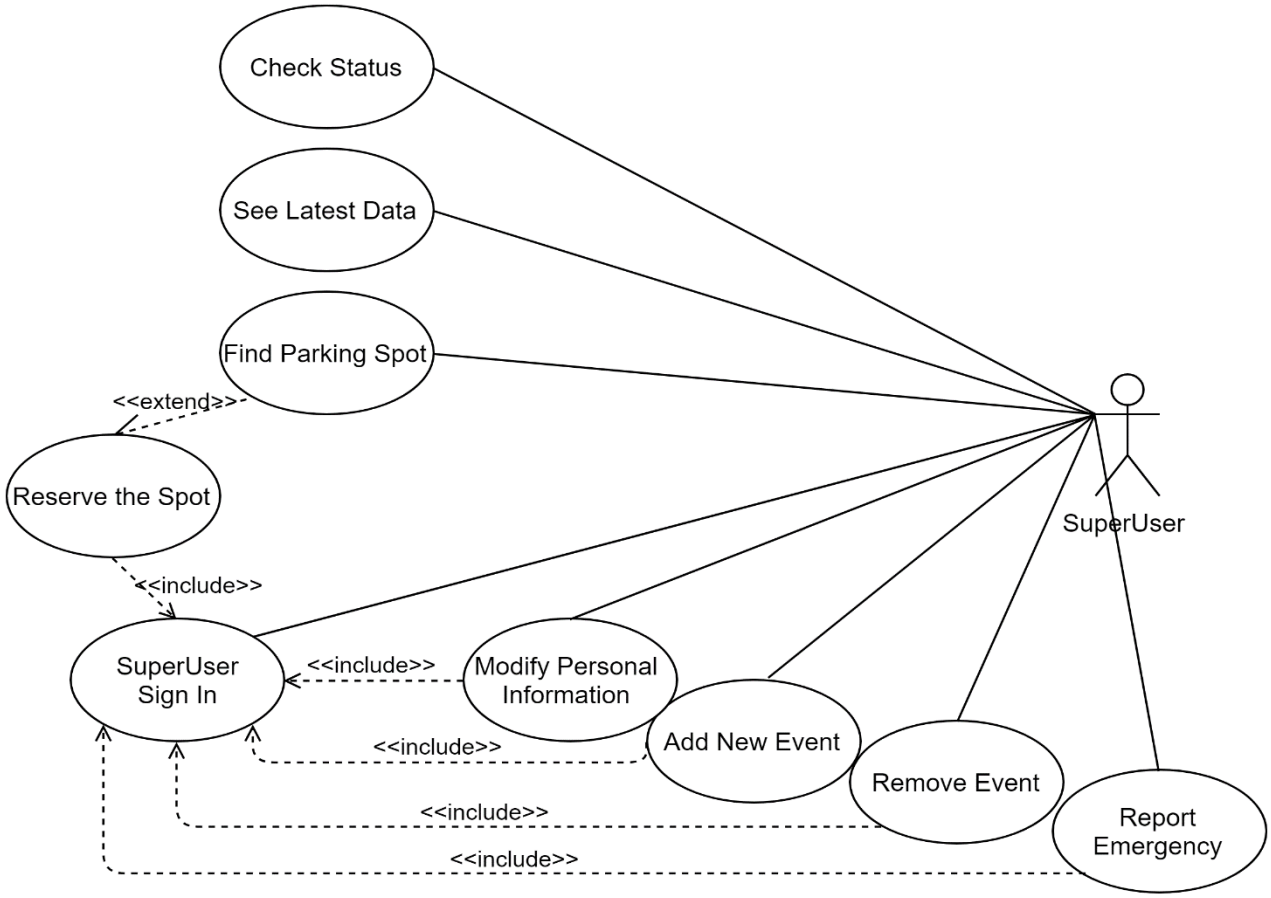
She wants more information and navigate to the Data section, here she finds out that the CO2 level is too high, to not endanger the population the access to the center is open only to the residents.

3.2.3 Use Case Diagrams

Use case for “normal” app users, that can be guests or become user after registration:



Special use case for the superursers that have more advanced functions:



3.2.4 Use Case Descriptions

User Registration

Actor:	Guest
Input Conditions	The guest is on the homepage, haven't registered yet
Events flow	Guest clicks on the Sign In section Guest clicks on "register" Guest fills in the required information and sends them The system saves the information The system sends a confirmation email Guest follow the instruction on the email
Output Conditions	The process ends successfully and the guest is redirected on the sign in page
Exception	The guest already had an account The guest doesn't follow the instruction on the email The guest inserted invalid information The system is not reachable
Reached Goals	[G5] [G2]

User Login

Actor:	Guest
Input Conditions	The guest is on the homepage, have already registered
Events flow	Guest clicks on the Sign In section Guest insert username and password Guest send the information to the server The system evaluates the received credential The Guest is now upgraded to User
Output Conditions	The process ends successfully and the user is redirected to the personal area.
Exception	The guest doesn't have an account The guest filled wrong credentials
Reached Goals	[G5] [G2]

Park Spot Reservation

Actor:	User
Input Conditions	The user has already logged in, the user is in the homepage
Events flow	User clicks on the Data section User chooses the “find me a parking spot” function The system automatically receives and analyze user data The system provides few alternatives based on the user data The user chooses which alternative fits best and reserves it The system sends a request to the parking lot The system sends an email to the user to confirm
Output Conditions	The process ends successfully and the guest has an email as proof of the reservation
Exception	The user closes the website before the process is complete The user changed some information without warning the system (email) The parking lot system is unreachable
Reached Goals	[G5] [G3]

SuperUser Login

Actor:	Guest
Input Conditions	The guest is on the homepage, have already valid credentials
Events flow	Guest clicks on the Sign In section Guest insert username and password Guest send the information to the server The system evaluates the received credential The Guest is now upgraded to SuperUser
Output Conditions	The process ends successfully, and the user is redirected to the personal area.
Exception	The guest doesn’t have an account The guest filled wrong credentials
Reached Goals	[G5] [G4]

New Event Registration

Actor:	SuperUser
Input Conditions	The superuser is on the homepage, already logged in
Events flow	SuperUser clicks on the Sign In section SuperUser clicks on the “add new event” option SuperUser fills in the required information and sends them The system saves the information The system sends a confirmation email SuperUser follow the instruction on the email
Output Conditions	The process ends successfully and the SuperUser receives the confirmation email
Exception	The information entered are incomplete The system is not reachable
Reached Goals	[G5] [G4]

Emergency Report

Actor:	SuperUser
Input Conditions	The superuser is on the homepage, already logged in
Events flow	SuperUser clicks on the Sign In section SuperUser clicks on the “emergency report” option SuperUser fills in the required information and sends them The system saves the information The system automatically warns authorities The system sends a confirmation email
Output Conditions	The process ends successfully and the SuperUser receives the confirmation email
Exception	The information entered are incomplete The system is not reachable
Reached Goals	[G5] [G4]

Average Use

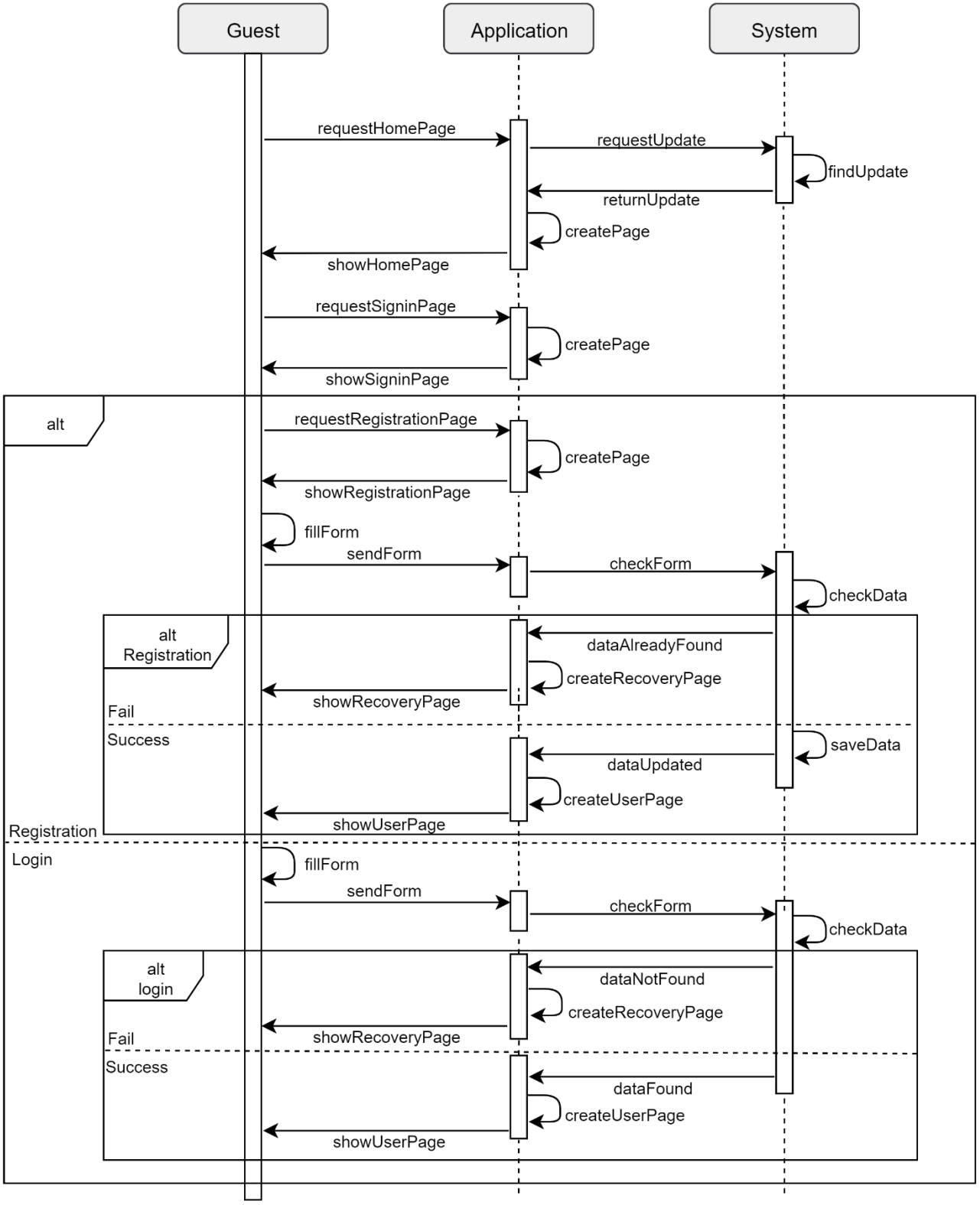
Actor:	Guest
Input Conditions	/
Events flow	Guest opens the homepage Guest checks the current traffic status Guest clicks on the “data” section to know more Guest inspects the latest data System provides the data from the DB Guest checks near parking spots System provides the data from the DB
Output Conditions	The User obtained the information he was looking for
Exception	System is not reachable
Reached Goals	[G5] [G3] [G2]

Remove Event

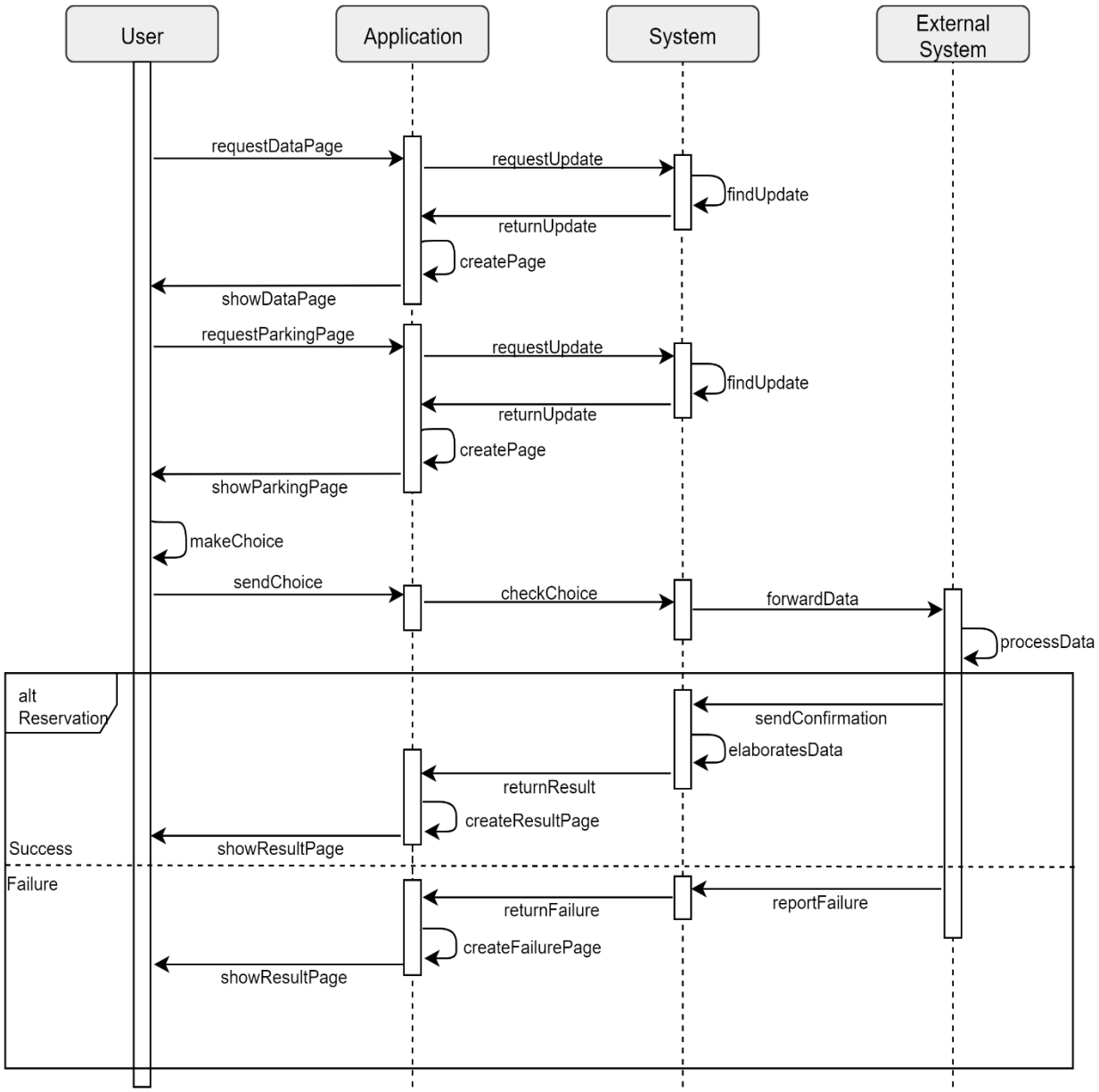
Actor:	SuperUser
Input Conditions	The superuser is on the homepage, already logged in
Events flow	SuperUser clicks on the Sign In section SuperUser clicks on the “remove event” option System provides a list of upcoming events SuperUser chooses which events he wants to remove System ask for an additional form to fill SuperUser fills the form about the motivations of the choices and sends System sends a confirmation email with a recap
Output Conditions	The process ends successfully and the SuperUser receives the confirmation email
Exception	The information entered are incomplete The system is not reachable
Reached Goals	[G5] [G4]

3.2.4 Sequence Diagrams

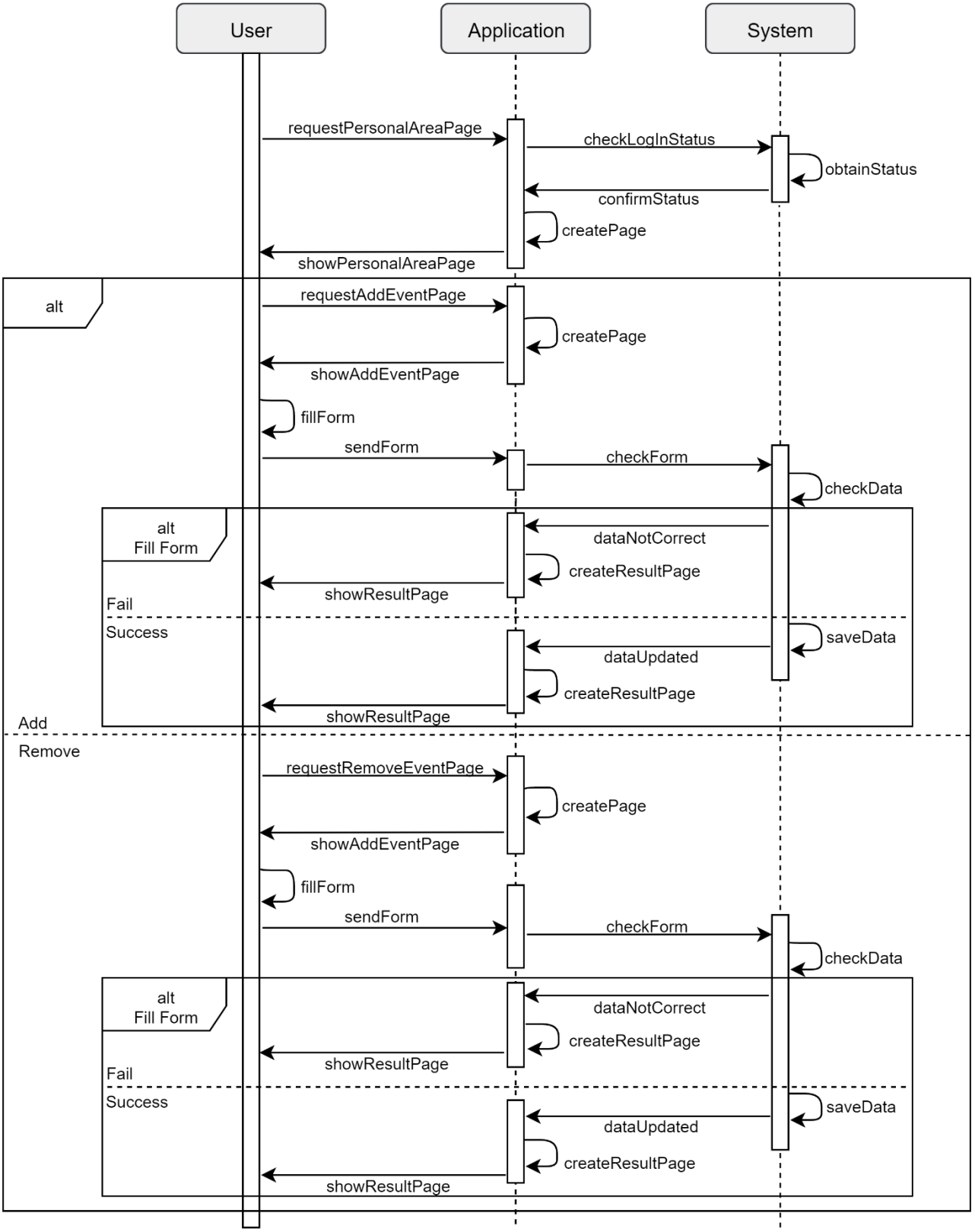
“Guest Registration and Login” Diagram:



“Parking Spot Reservation by User” Diagram:



“Add/Remove Event by SuperUser” Diagram:



3.3 Performance Requirements

- The system must be able to process a great number of requests at the same time and has to process most of them in 5 or less seconds.
- The system has to handle a lot of daily requests, it's esteemed that 10.000 people will use the app daily
- The system must keep an uptime of at least 99,8%

3.4 Design Constraints

3.4.1 Standards Compliance

- The software must be written in compliance with all the with the web standards and common practices such as the ones of W3C (World Wide Web Consortium).

3.4.2 Hardware Limitations

- The device used should have an internet connection
- The device should run an OS that's not considered too old
- The device should have enough memory to run the app or load the website
- GPS is an optional to run the app

3.5 Software System Attributes

3.5.1 Reliability

- On average the system should have a failure every at least 30 days
- In case of failure the system will be restored in few hours
- In case of problem the user information and credential should be easily restored and with a chance higher than 99,9%

3.5.2 Availability

- Users should be able to rely on this app at any moment, but since failures may happen in the night when restoring the system takes longer then 99,8% is a high enough ratio to maintain. In the future this value should be improved.

3.5.3 Security

- User personal information should always be secret and encrypted, the website will use https to offer a better security to users.

3.5.4 Maintainability

- The app offers a limited number of functionalities and the code complexity can then be easily kept very low.

3.5.5 Portability

- The app should work with any Android or iOS devices that satisfy the hardware requirements. The website can be accessed on any device that can run a last generation browser.

4 – Formal Analysis Using Alloy

```
/* Signatures */

/*Every person is possibly assigned to a position,
whether they are Guests, Users or Superusers*/
abstract sig Person{
  currentPos: lone Position
}

sig Position{}
sig Text{}

/*Current traffic status, R G Y*/
sig Status{}
one sig Red extends Status{}
one sig Yellow extends Status{}
one sig Green extends Status{}

/*ID is one and assigned randomly,
mostly to keep the world "easier" to read*/
sig randomID{}
{#randomID=1}

sig Time{
   hour: one Int,
   day: one Int,
   month: one Int,
   year: one Int
}
{
  hour>0
  day>0
  month>0
  year>0
}

sig TrafficLight {}
```

```
/*Basic components managed by the system*/

sig Actuator{
  hasID: one randomID,
  refersTo: one TrafficLight
}

sig Sensor{
  hasID: one randomID,
  hasValue: one Int
}
{
  hasValue>0
}

sig Display{
  hasID: one randomID
}

/*Users Hierarchy: SuperUser>User>Guest*/

sig Guest extends Person{
  tempID: one randomID
}

sig User extends Guest{
   username: one Text,
  reservations: set Reservation,
   receives: set ParkingSuggestions
}

sig SuperUser extends User{
}

/*Registered Users Reservations, multiples allowed*/
sig Reservation{
  position: one Position,
  lot: one ParkingLot,
  time: one Time
}
```

```
/*Represents the internal system*/
sig System {
  currentStatus: one Status,
  managing: set Actuator,
  showing: set Display,
  sensors: set Sensor,
  news: lone News
}

/*Possible reprentation of events, news are multiple events*/
/*Events Hierarchy*/

abstract sig Event{
   start: one Time,
   end: one Time,
  addedBy: one SuperUser
}
{
  start != end
  start.year=end.year
  start.month=end.month
  end.day=start.day
  end.hour>start.day
}

sig SpecialEvent extends Event{
   title: one Text,
   desc: one Text,
   position: one Position
}

sig Emergency extends Event{
   type: one Text,
   position: one Position
}

sig News {
   events: set Event
}
```

```
/*Every suggestion is the system answering with
multiple possible parking lots near you, user may
ask many times for a suggestion*/
sig ParkingLot {
   parkingPosition: one Position,
   distance: one Int,
   spots: one Int
}
{
  spots>0
  distance>0
}

sig ParkingSuggestions {
   suggestions: some ParkingLot
}


/* Facts */

/*Unique Usernam is required */
fact UniqueUsername{
  no disjoint u1,u2: User | u1.username = u2.username
}

/*If user made a reservation he certantly asked
for a suggestion*/
fact reservationImplySuggestion{
  all u: User |
  #u.reservations < #u.receives
}

/*Can't have two reservations on the same parking lot
for a single user*/
fact noDoubleReservations{
  all u: User |
  no disj x,y: u.reservations | x.position=y.position
}
```

```
/*Number of actuators and traffic lights should be the same,
at least the active working ones*/
fact actuatorHasTrafficLight{
  #Actuator = #TrafficLight
}

/*Two actuators can't refer to the same traffic light*/
fact noDoubleActuator {
  no disj a1,a2: Actuator | a1.refersTo = a2.refersTo
}

/*System shouldn't suggest the same parking lot twice
when asked for suggestions*/
fact noDoubleSuggestion {
  all u: User |
  no disj x,y: u.receives.suggestions | x.parkingPosition=y.parkingPosition
}

/*Can't have different parking lots in the same position*/
fact differentPosParking{
  no disj p1,p2: ParkingLot | p1.parkingPosition = p2.parkingPosition
}

/*Events can't overlap*/
fact disjointEvents{
  all disj e1,e2: Event |
  e2.start.year=e1.start.year &&
  e2.start.month=e1.start.month &&
  e2.start.day=e1.start.day &&
  (((e1.start).hour)>((e2.end).hour) ||
  ((e2.start).hour) > ((e1.end).hour))
}

/*Some possible bindings*/
fact Bindings {
  all u: User, r: Reservation | r in u.reservations
  all s: System, se: Sensor | se in s.sensors
  all s: System, n: News | n in s.news
  all s: System, a: Actuator | a in s.managing
  all s: System, d: Display | d in s.showing
}
```

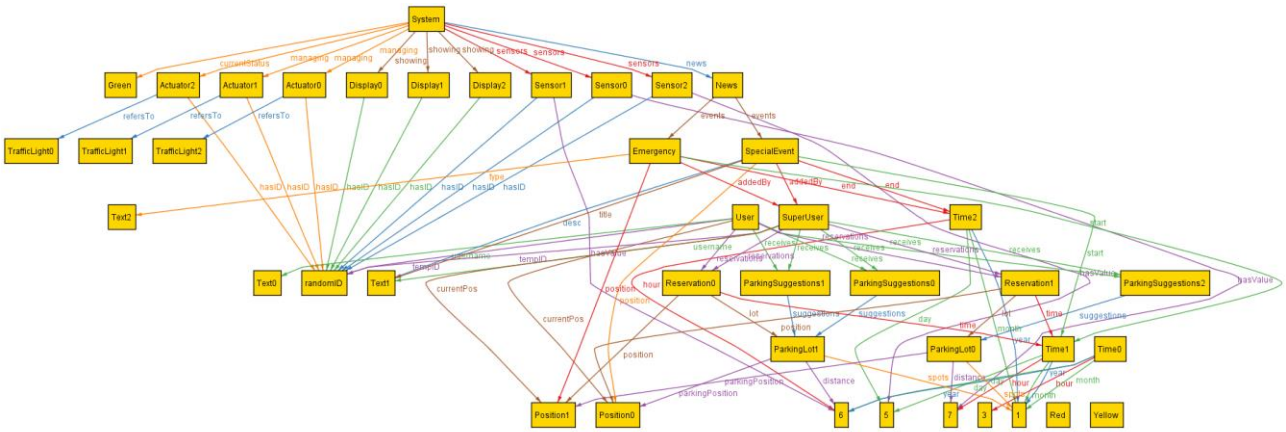
```
/* Predicate */

/*A possible world may have the following signatures*/
pred generateWorld{
  #System = 1
  #User = 2
  #Reservation = 2
  #Emergency = 1
  #SpecialEvent = 1
}

check DisjointEvents for 3
run generateWorld for 3
```

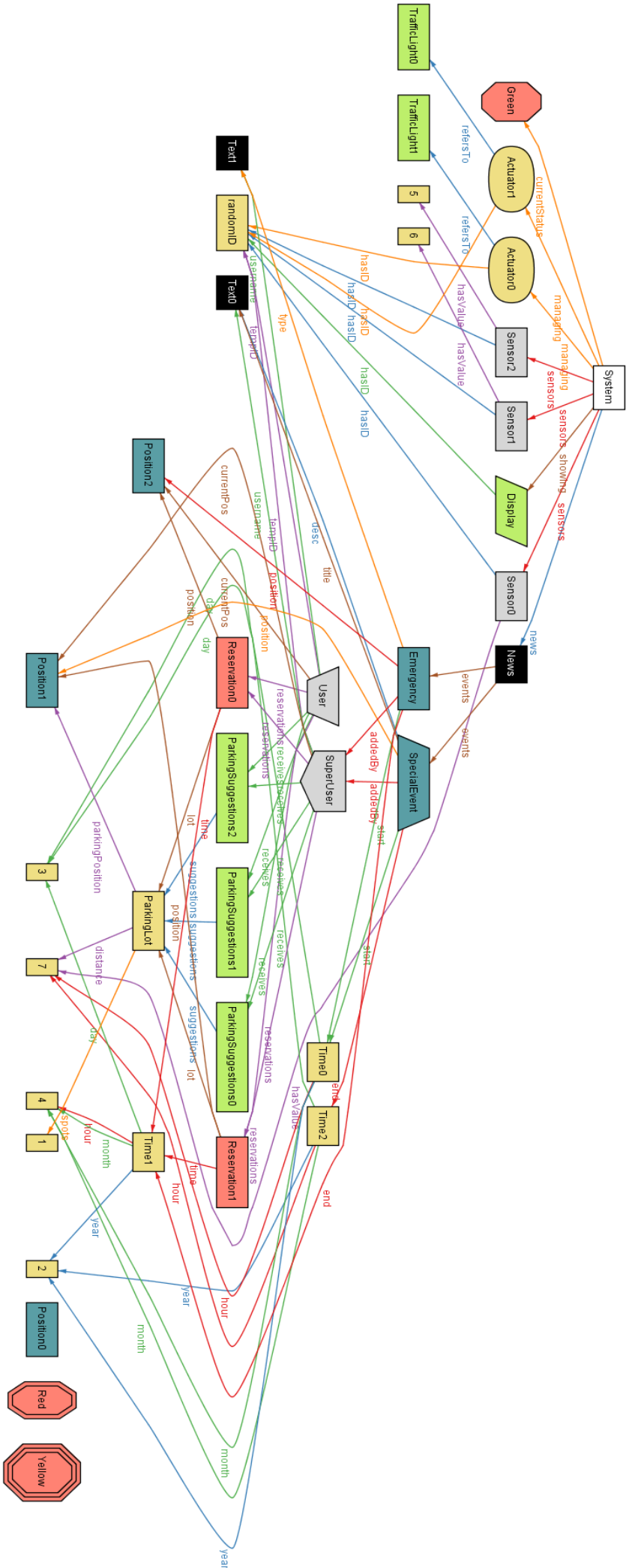
```
2 commands were executed. The results are:
  #1: No counterexample found. DisjointEvents may be valid.
  #2: Instance found. generateWorld is consistent.
```

Generated World Template:



Generated World

Magic Layout:



5 – Effort Spent and References

This project has been developed by a single member because of a lack of other teammates, probably because not many other students are taking the second delivery of this year project.

The amount of time needed to complete this document is about 60 hours, many of them spent thinking which was the best way to display the information and making/modifying the needed diagrams.

Tools Used:

- The diagrams have been done using the draw.io free tool.
- The wireframes have been realized using Balsamiq.
- Alloy analyzer 4.2 for alloy analysis.
- Microsoft Word to write this document

The commits on GitHub are symbolical of the major steps of the work, as taking the project alone didn't really require updating the other members and share the work.

A small personal thought is that I feel this kind of projects are much more useful than plain teaching, they are quite funny to develop (this is very personal) and offer many possibilities to look for interesting information online, while learning to use useful tools.