Fix My Town!

Trabalho de Métodos de Desenvolvimento de Software 2018/19

Professor: Igor Ruivo

Turno Prático 7

Hugo Miguel Velez Simão 50266

Pedro Miguel Oliveira Valente 50759

Rafael Rodrigues Gameiro 50677

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Índice:

1.) Justificação das nossas opções ……...3
2.) Diagrama de casos de uso ..…….5
3.) Três especificações de casos de uso ...……...6

3.1.) Report Problem ……..……...6

3.2.) Validate Problem ..……....8

3.3.) Register public Fixer …..……10

4). Activity Diagrams ……..12

5). Sequence Diagrams ……14

6). Class Diagrams ……18

6.1). Analysis Diagram ……18

6.2). Design Diagram ……..18

7). OCL ……...19

8). Component Diagram ……21

9). Package Diagram ……22

9.1). Package Diagram Design ……...22

10). Conclusions ……24

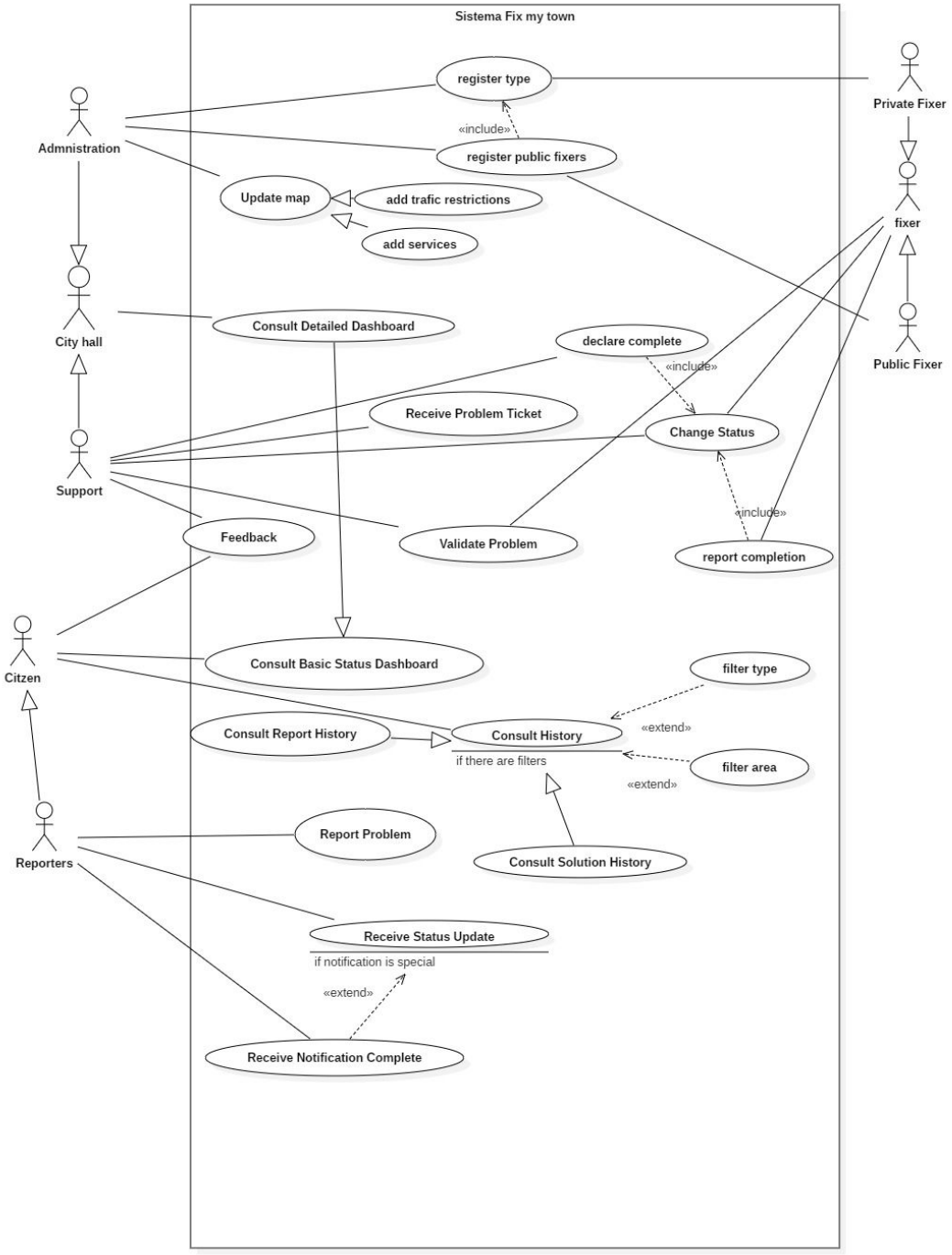
Justificação das nossas opções

1. Em relação aos **actores**:
    a. Assumimos que um cidadão poderá ter dois estatutos: O primeiro corresponde a um cidadão regular que consideramos que nunca reportou um problema . O segundo um reporter que já reportou um problema e por isso recebe notificações do estado da resolução de problemas
    b. Assumimos que os fixers poderão ser públicos ou privados: Os públicos são serviços do city hall ou de instituições públicas, os privados são instituições privadas.
    c. Assumimos que o city hall não é um fixer pelo que as funções de fixer que o city hall exerce são de um departamento à parte que é um fixer público. A city hall também está separada em dois departamentos distintos: o Back end e o front end. O back end trata de assuntos relacionados do registo de fixers (públicos ou privados) e updates ao mapa enquanto o front end está responsável pela parte da resolução de problemas, fazendo por exemplo actualizações do estado de resolução dos problemas, aprovando estes ou recebendo feedback.
    d. Assumimos também que os Fixers e a City Hall eram abstractos.
2. Em relação aos **casos de uso**:
    a. Assumimos que quando um reporter recebe uma notificação de conclusão esta é um caso especial de uma notificação normal que por isso tem um funcionalidades diferentes que lhe dão mais destaque justificando outro caso de uso

b. Assumimos que (como especificado no caso de uso abaixo) quando um problema é reportado nem sempre é emitido um report ticket por já ter sido reportado. Por isso o caso de uso report problem poderá ser concluído sem um report ticket ser recebido pelo city hall.
c. Assumimos que o consult history pode ser de dois tipos. Um de reports, outro de solutions justificando por isso dois casos de uso que generalizam este. Se o user quiser usar um filtro este extende ou para um filtro de localização o para um de tipo ou ambos
d. Assumimos que os cidadãos normais têm acesso a um dashboard com a informação básica dos estados das soluções que será depois generalizada para um dashboard mais detalhado a que o city hall terá acesso.
e. Assumimos que quer o city hall quer os fixers podem mudar o estado de uma resolução em todos os casos, excepto na sua conclusão. Aqui o fixer poderá considerar que um problema está completo e reportar o fim da obra, isto constitui um estado especial chamado percieved completion. Este não é um estado final mas tem características distintas o que justifica a existência de um novo caso de uso, o report completion. A obra só muda o estado para completo quando a city hall faz a análise da solução fornecida pelos fixers, aí é chamado o caso de uso declare complete, tanto este como o anterior atualizam o estado da obra, justificado o include do diagrama.
f. Assumimos que a city hall pode fazer update ao mapa da cidade sendo que este caso de uso pode-se generalizar em 2: Adicionar serviços, que adiciona uma nova escola, um novo evento e outros novos landmarks. E adicionar restrições de trânsito pintando as estradas com várias cores consoante o problema.

Diagrama e casos de uso

(**nota:** esta secção está em inglês)



Use cases

Name: Report Problem

Id: 1

Description: Reporter reports a problem in the city to be solved

Actors:

Main: Reporter

Secondary: none

Pre-conditions: Issue exists; Reporter exists

Main flow:

01- The case begins as the Reporter selects the report form of the app, opening it;

02- The Reporter selects the Location of the problem on a map;

03- If type of the issue is already included in the list of types;

03.1- The reporter selects the respective type;

04- Else The reporter selects other;

04.1- The reporter writes the type of problem;

05- The Reporter writes a brief title and description of the issue;

06- If the Issue has a deadline;

06.1- The Reporter selects it on a calendar;

07- Else the Reporter skips this section;

08- The Reporter rates the issue with one of 5 priority levels;

09- If the Reporter has photos of the issue;

09.1- The Reporter submits them to the system;

10- If the Reporter has witnesses;

1**0.1**- The Reporter lists them;

11- If the Reporter has other observations;

11.1- The Reporter writes the observations;

12- The Reporter gets a thank you message, confirming the submission;

13- The system issues problem ticket is issued to the database;

Alternative flows: Location doesn't exist; Report already made

Post-conditions: Problem report submitted

Name: Report Problem **:** Location doesn't exist

Id: 1

Description: Reporter reports a problem in the city to be solved

Actors:

Main: Reporter

Secondary: none

Pre-conditions: Issue exists, Reporter exists, no location exists

Alternative Flow:

01- The alternative flow begins when there is no location (after third step);

02- The Reporter is advised to put a valid location;

03- Because there is none, the Reporter clicks cancels this fase;

04- The reporter receives a warning;

05- The reporter resumes the previous flow;

Post-conditions: Returns to the original flow

Name: Report Problem **:**Report Already made

Id: 1

Description: Reporter reports a problem in the city to be solved

Actors:

Main: Reporter

Secondary: none

Pre-conditions: Issue exists, Reporter exists, no location exists

Alternative Flow:

01- The alternative flow begins when the report is submitted that reports a problem that was already issued (after 13th step)

02- The program fetches the problem ticket from the database

03- Its details are completed using extra information in the new report

04- The program updates the database

05- The Reporter is notified that the problem had been reported.

Post-Conditions: Issue solved

Name: Validate Problem

ID: 4

Description: The City Hall validates a problem and the fixers are notified

Actors:

Primary: Support

Secondary: *Fixers*

Pre-condition: The City Hall front-end has enough money to invest in the problem; The City Hall front-end has fixers available; The problem has to be valid and have a solid structure

Main flow:

01- The Use Case starts when a member of the city hall front end selects the option "Validate problem";

02- The System shows a list of problems to the user;

03- The User selects one of the problems;

04- If the problem is not valid:

04.1-The User selects the option "not valid";

04.2-The System activates the Alternative flow Problem not valid;

05- The user selects the option "Insert Cost";

06- The System prompts the user to writes the estimated cost;

07- If the available funds are less than the estimated cost:

07.1- The System activates the Alternative flow Insufficient funds;

08- If there aren't any fixers available.

08.1- The system activates the Alternative Flow Insufficient funds;

09- The System Shows a list of Fixers to the user;

10- The User selects the fixers he wants to use;

11- The Fixers are notified by the System;

Alternative Flows: Problem not valid; Not Enough Funds

Post-Conditions: The estimated funds are removed

Name: Validate Problem **:** Problem not valid

ID: 4

Actors:

Primary: Support

Pre-condition: The user selects the option "not valid"

Alternative Flow:

01- The alternative flow begins when the user selects the option ”not valid” (step 3.1);

02- The system prints a prompt asking the user if he wants delete the problem;

03- The user selects the option "yes" and the problem is removed from the list;

04- The system goes back the the select menu;

Post-Conditions: Issue solved; The estimated funds are removed;

Name: Validate Problem **:** Not Enough Funds

ID: 4

Actors:

Primary: Support

Pre-condition: The user tries to validate a problem without having enough funds

Alternative Flow:

01- The alternative flow starts when there aren't enough funds (step 04.1 and 05.1);

02- The problem changes his status from active to unfunded;

03- The system goes back to the select menu;

Post-condition: Issue solved; The estimated funds are removed;

Name: Register public Fixer

ID: 3

Description: A fixer registers to the app for the first time

Actors:

Primary: Public Fixer

Secondary: Administration

Pre-condition: Public Fixer wants to register
Main Flow:

01- The use case starts when the public fixer selects the option "register new Fixer";

02- The fixer insert its name;

03- The fixer insert its email Address;

04- The fixer types a password;

05- The fixer insert its location;

06- The fixer submits his request to create a new account, and a dialog message appears;

07- If the fixer chooses to submit;

07.1- The system creates a new fixer account in the system;

08- If the fixer chooses not to submit;

08.1- The fixer can change its options until he's satisfied;

Alternative Flows: AlreadyExistingName; FixerIsPrivate
Post-condition: A fixer was added to the system

Alternative Flow: Register public Fixer **:** AlreadyExistingName
ID: 3
Description: The system penalizes the fixers based on the new submission.
Actors:

Primary: Fixer

Pre-condition: The fixer as entered an already existing name.
Alternative Flow:

01- The alternative flow begins after step 2 of the main flow;

02- The system informs the fixer that there is already a fixer with the inserted name;

03- A cross appears next to the name's text box, indicating that the fixer can not choose that name;

Post-condition: Issue solved

Alternative Flow: Register public Fixer **:** FixerIsPrivate
ID: 3
Description: The system congratulates the fixers based on the new submission.
Actors:

Primary: Public Fixer

Pre-condition: The fixer is already private
Alternative Flow:

01- The alternative flow begins after step 7 of the main flow;

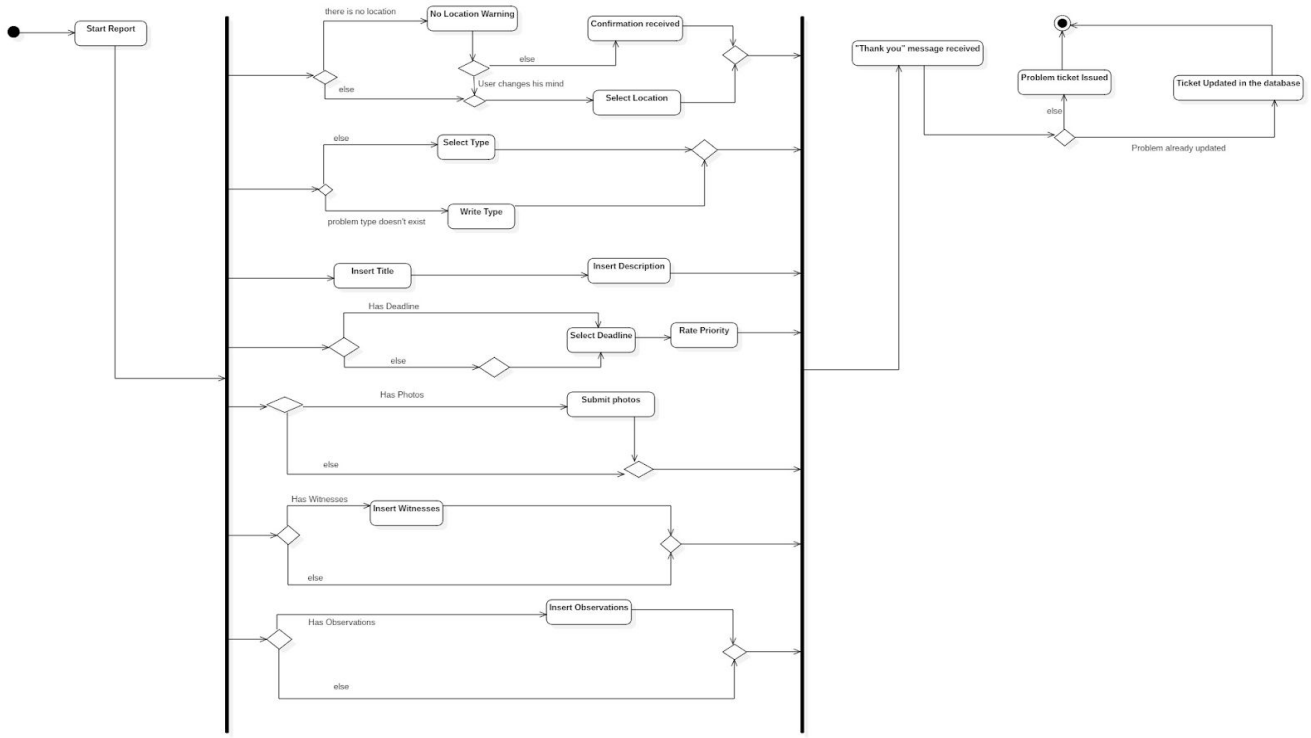
02- The system informs the fixer, through a dialog message, that the account cannot be created because the fixer is already a private fixer;

03- The systems returns the fixer to the initial page;

Post-condition: Issue Solved

Activity diagrams

Este activity diagram descreve o caso de uso 1: report problem. Como podemos ver usamos em vez de exceções usamos ifs para representar o início dos alternative flows já que consideramos que devido às suas verificações não precisamos de usar o mecanismo de exceções.

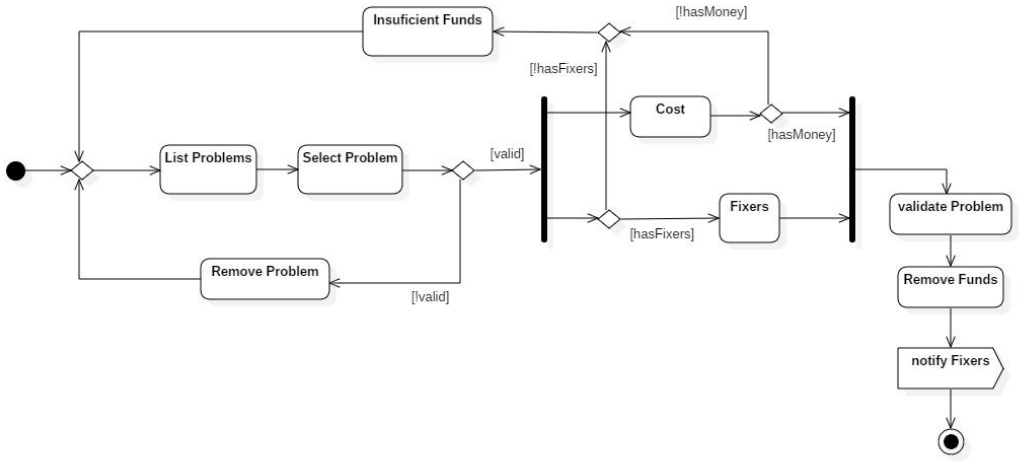


No desenho do diagrama de atividades baseado no caso de uso "Validate Problem" inicialmente tinha optado por usar duas exceções em vez dos nós de decisão na zona que se encontra em paralelo, optei por usar os nós de decisão uma vez que ambos usam a mesma sequência de resolução, ambos ativam a flow alternativa de "InsuficientFunds", uma interrupção não faria sentido tendo em conta que é preciso fazer uma verificação do fundos disponíveis de modo a saber se o uso de uso continua com a flow normal.

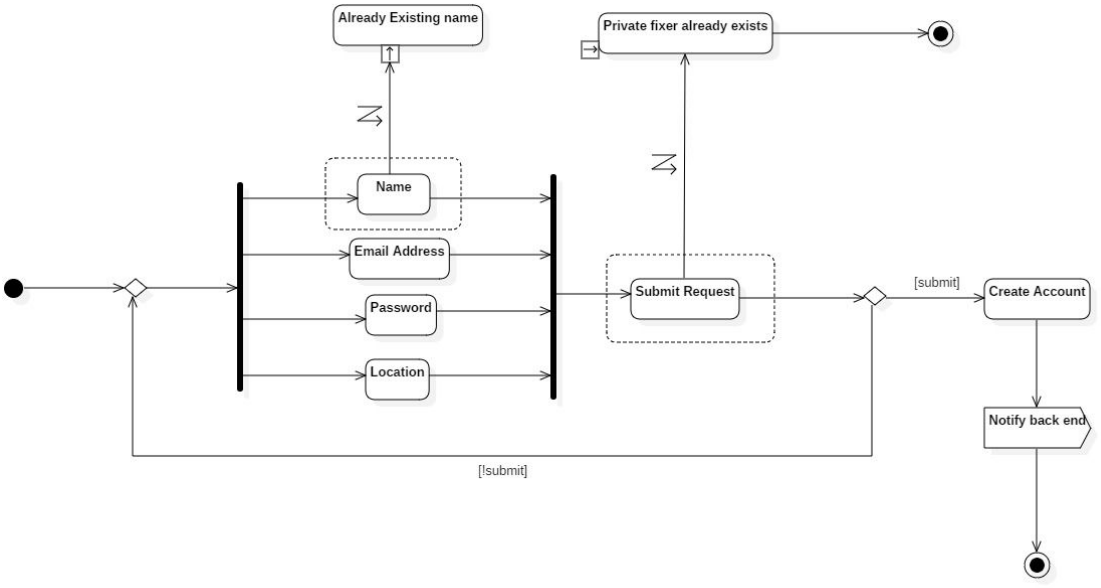
Neste diagrama a utilização de paralelismo justifica-se uma vez que ambas as atividades, de escolher o custo e escolher os fixers podem decorrer de forma independente uma da outra, no entanto

ambas precisam de acontecer de modo a que o problema se torne válido, e a sua ordem de execução não interessa.

Na atividade cost faz se a verificação dos fundos depois desta acontecer porque a atividade cost implica a inserção de um valor por parte de um utilizador e só após a inserção deste é que se pode verificar se a City Hall tem fundos ou não, enquanto que no caso da atividade fixers a verificação faz-se antes uma vez que não faz sentido sequer realizar a atividade se não houverem fixers disponíveis.

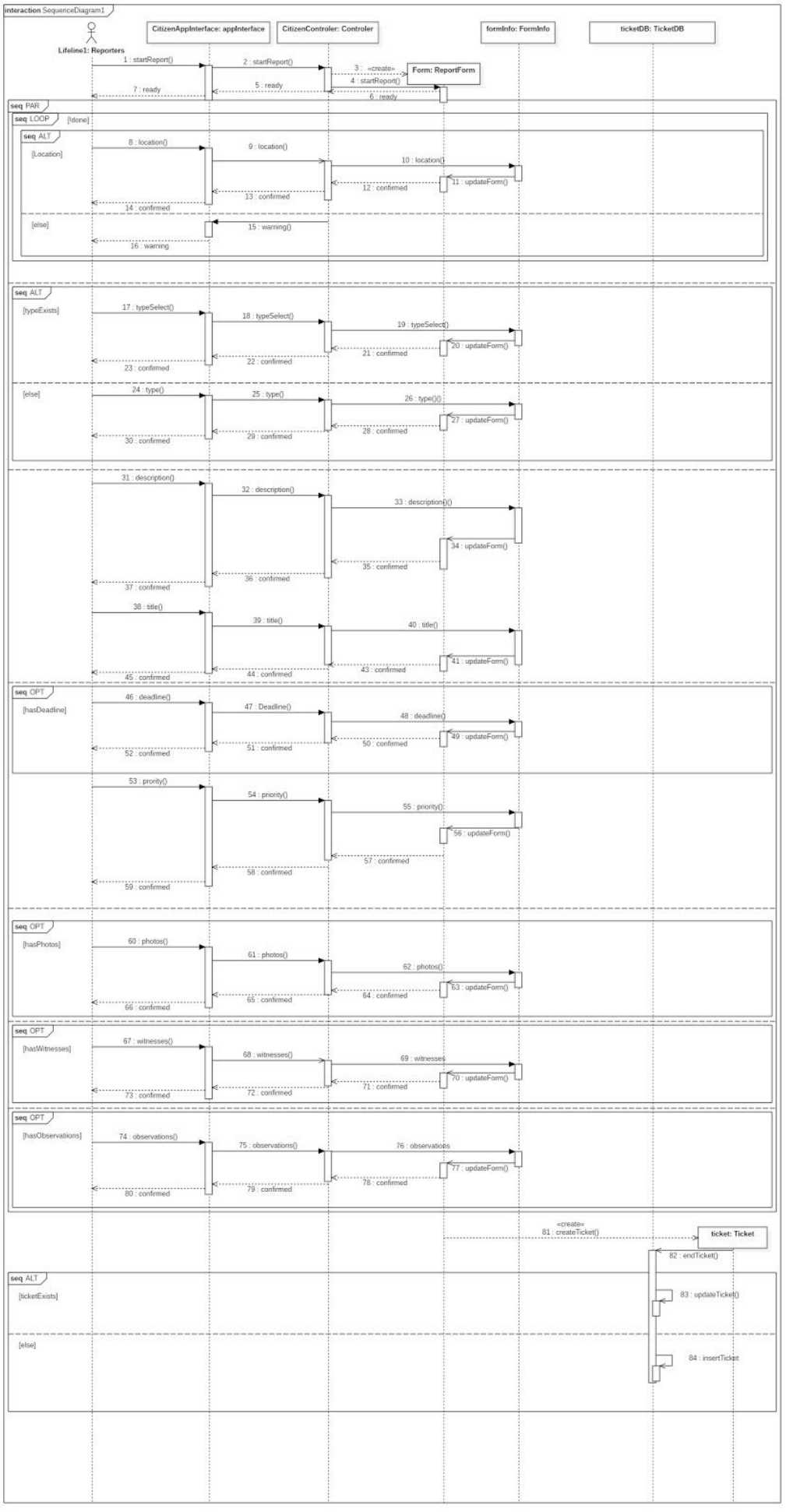


Este activity diagram descreve o caso de uso 3: register Public Fixer.
Neste caso de uso, recorremos ao uso de exceções pois nos pareceu a melhor forma de representar o caso de uso.
Cada caso de uso leva a action nodes diferentes, e consoante a exceção, pode ou não levar a terminar o caso de uso.



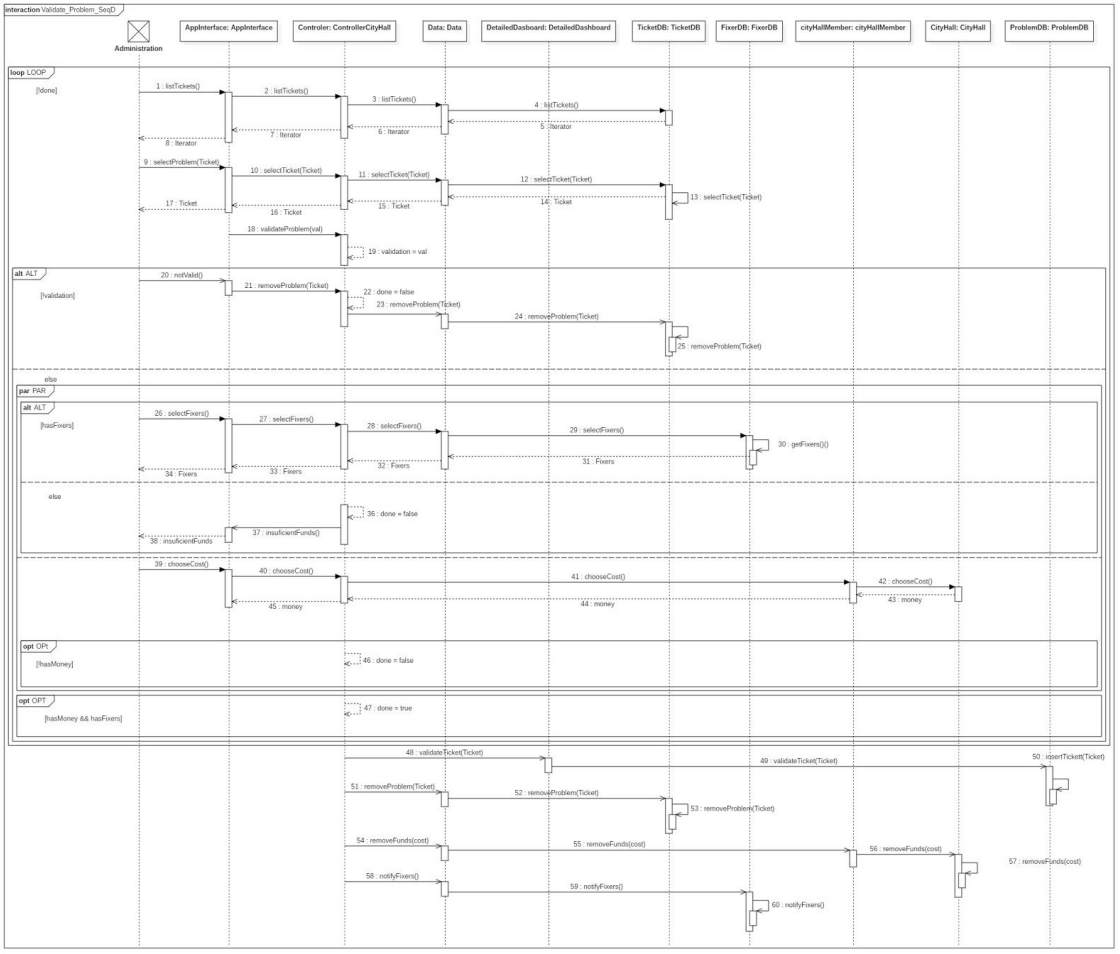
Sequence Diagrams

Este sequence diagram descreve o caso de uso 1: report problem. Como podemos ver o user interage com uma interface que por si interage com um controller que se encarregará de orientar as restantes operações. Como é possível ver o user não interage com a sua conta. Esta só está a ser usada para verificações dentro das funções do controller.

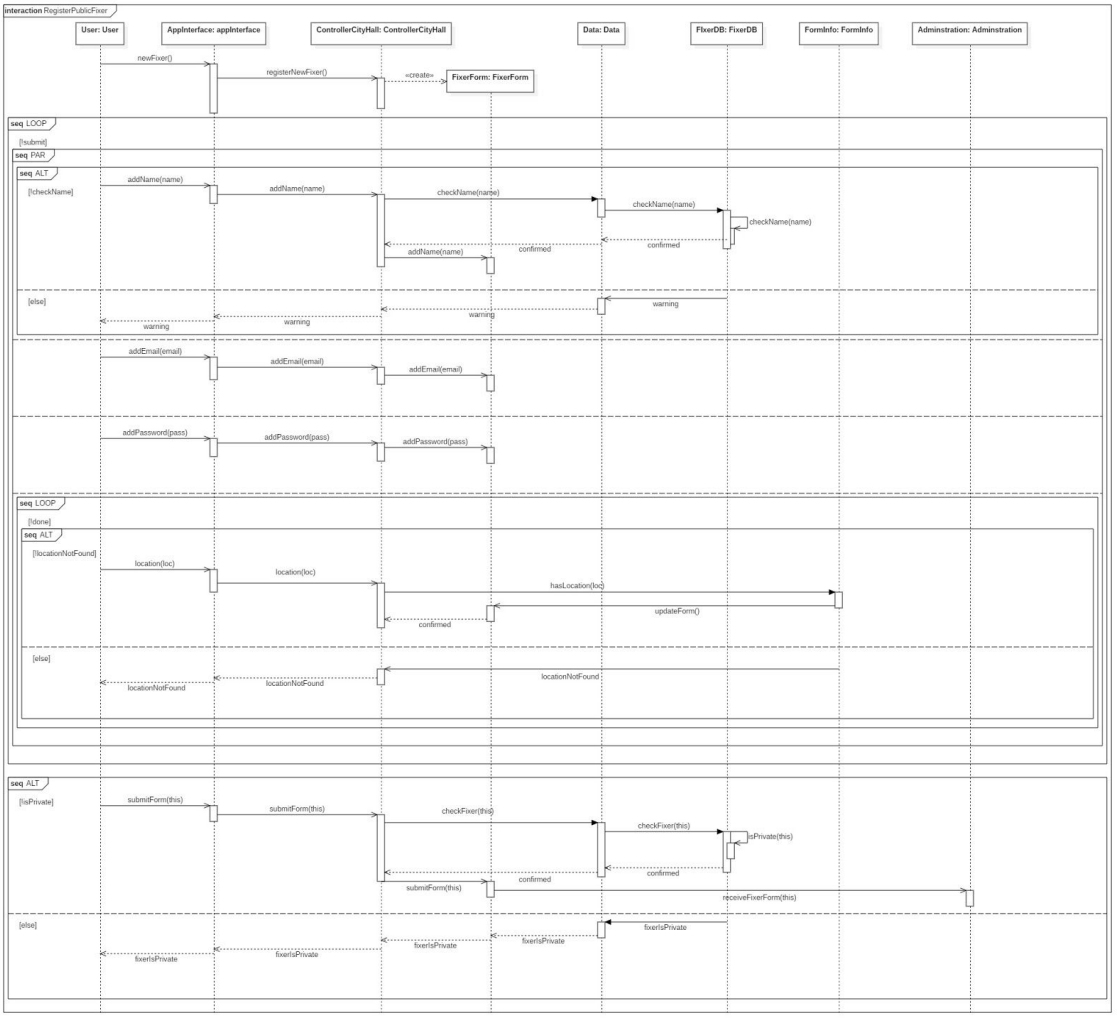

interaction SequenceDiagram1
Lifeline1: Reporters
CitizenAppInterface: appInterface
CitizenControler: Controler
Form: ReportForm
formInfo: FormInfo
ticketDB: TicketDB
1 : startReport()
2 : startReport()
3 : «create»
4 : startReport()
5 : ready
6 : ready
7 : ready
seq PAR
seq LOOP
[!done]
seq ALT
[Location]
8 : location()
9 : location()
10 : location()
11 : updateForm()
12 : confirmed
13 : confirmed
14 : confirmed
[else]
15 : warning()
16 : warning
seq ALT
[typeExists]
17 : typeSelect()
18 : typeSelect()
19 : typeSelect()
20 : updateForm()
21 : confirmed
22 : confirmed
23 : confirmed
[else]
24 : type()
25 : type()
26 : type()()
27 : updateForm()
28 : confirmed
29 : confirmed
30 : confirmed
31 : description()
32 : description()
33 : description()()
34 : updateForm()
35 : confirmed
36 : confirmed
37 : confirmed
38 : title()
39 : title()
40 : title()
41 : updateForm()
43 : confirmed
44 : confirmed
45 : confirmed
seq OPT
[hasDeadline]
46 : deadline()
47 : Deadline()
48 : deadline()
49 : updateForm()
50 : confirmed
51 : confirmed
52 : confirmed
53 : prority()
54 : priority()
55 : priority()
56 : updateForm()
57 : confirmed
58 : confirmed
59 : confirmed
seq OPT
[hasPhotos]
60 : photos()
61 : photos()
62 : photos()
63 : updateForm()
64 : confirmed
65 : confirmed
66 : confirmed
seq OPT
[hasWitnesses]
67 : witnesses()
68 : witnesses()
69 : witnesses
70 : updateForm()
71 : confirmed
72 : confirmed
73 : confirmed
seq OPT
[hasObservations]
74 : observations()
75 : observations()
76 : observations
77 : updateForm()
78 : confirmed
79 : confirmed
80 : confirmed
«create»
81 : createTicket()
ticket: Ticket
82 : endTicket()
seq ALT
[ticketExists]
83 : updateTicket()
[else]
84 : insertTicket

No diagrama de sequência baseado no caso de uso "Validate Problem" de modo a ciclo de atividades que constitui este decidimos optamos por usar um loop com uma variável que indica quando todas as condições necessárias para a validação de um problema se encontram representadas, ou seja quando os fundos do City Hall são superiores ao custo de resolução do problema e quando há fixers disponíveis, assim durante a execução deste loop sempre que se verifica que uma destas condições não é satisfazível a variável done fica a falso e no fim volta-se a ver se estas condições se verificaram, se sim então mete-se done a true e o loop termina.

Em seguida o ticket é guardado na base de dados dos problemas, é de notar que este não é criado uma vez que a gente o recebe da base de dados dos tickets, o mesmo é removido da base de dados dos tickets e os fixers são notificados.



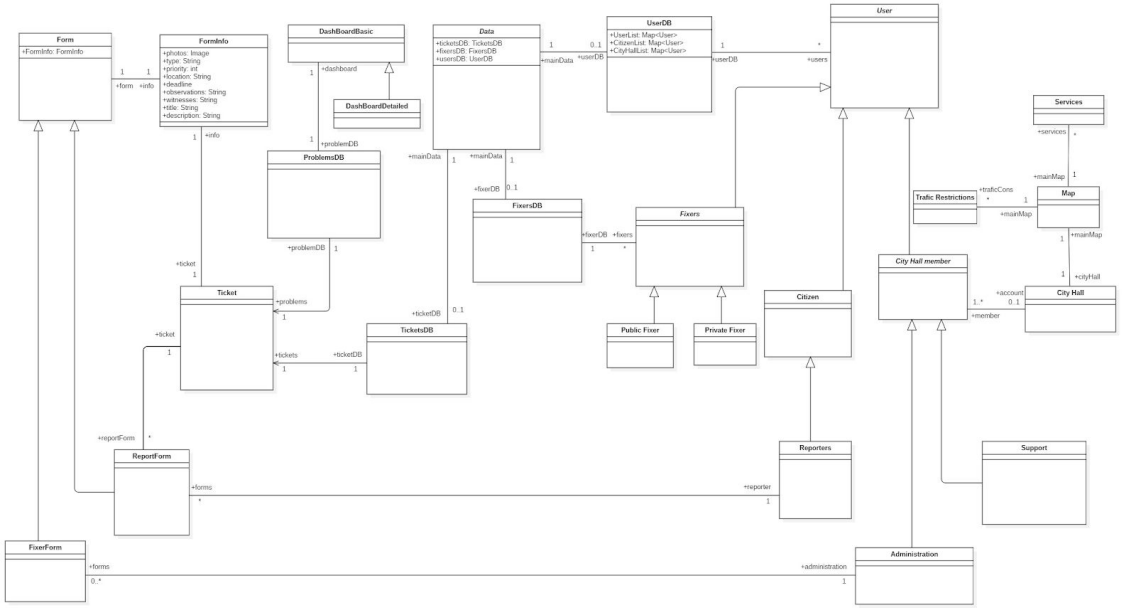
Este Sequence diagram descreve o caso de uso 3: register Public Fixer. O próprio ator interage com um interface de um controlador, que irá efetuar o registo do novo fixer.
Todas as informaçõe fornecidas pelo ator são passadas ao novo objeto Form, onde são inseridas. Algumas informações são primeiro verificadas de modo a impedir a repetição de informação comparativamente a outros fixers já existentes.



Class diagram

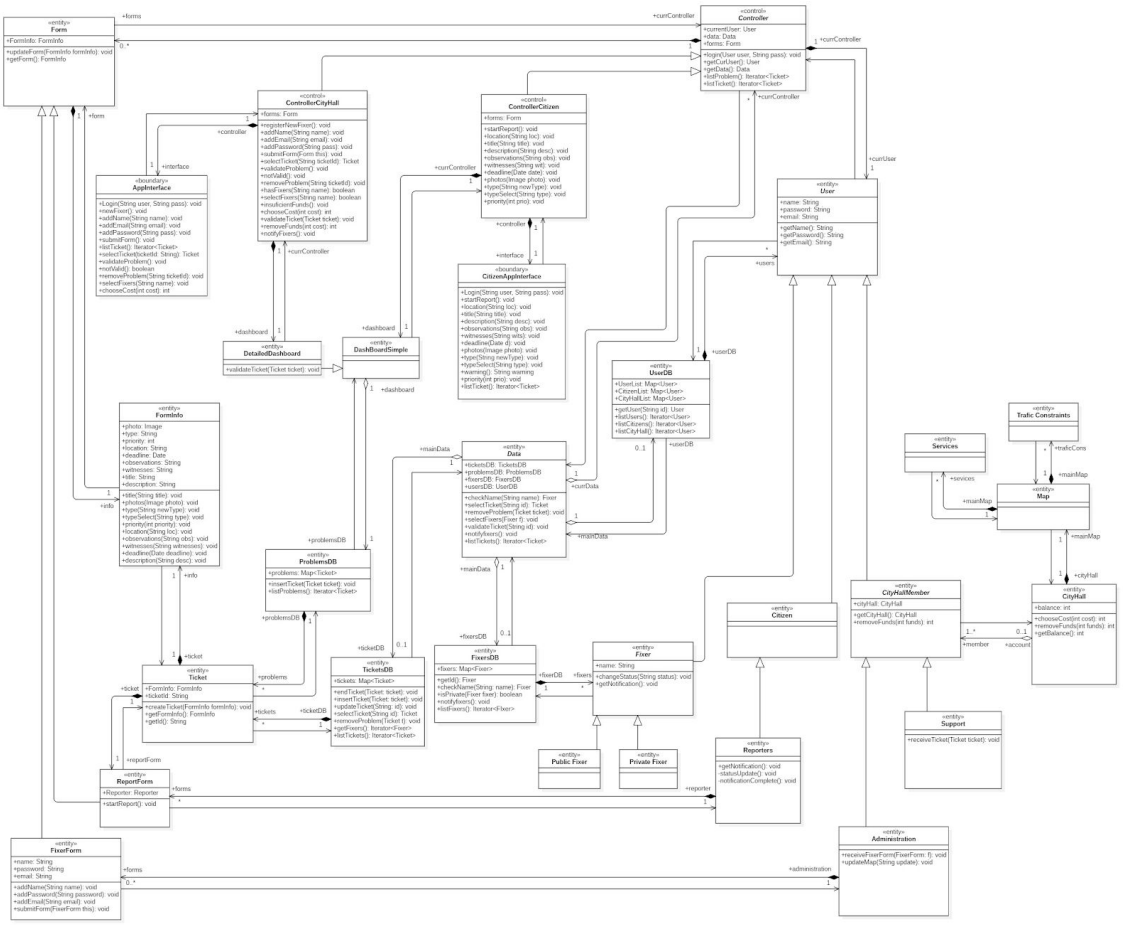
Analysis Class Diagram:

Uma classe essencial de se compreender para compreender este diagrama é a classe FormInfo. Esta classe é de vital importância pois é nela que guardamos a informação correspondente a um certo form que será depois “oferecida” a outras classes que necessitem de guardar esta informação (Tickets por exemplo).



Design Class Diagram:

Relativamente ao primeiro class diagram fizemos uma análise mais detalhada, acrescentando controllers e boundaries e especificamos novos métodos (alguns deles não relacionados com os nosso casos de uso) especificando também os seus tipos.



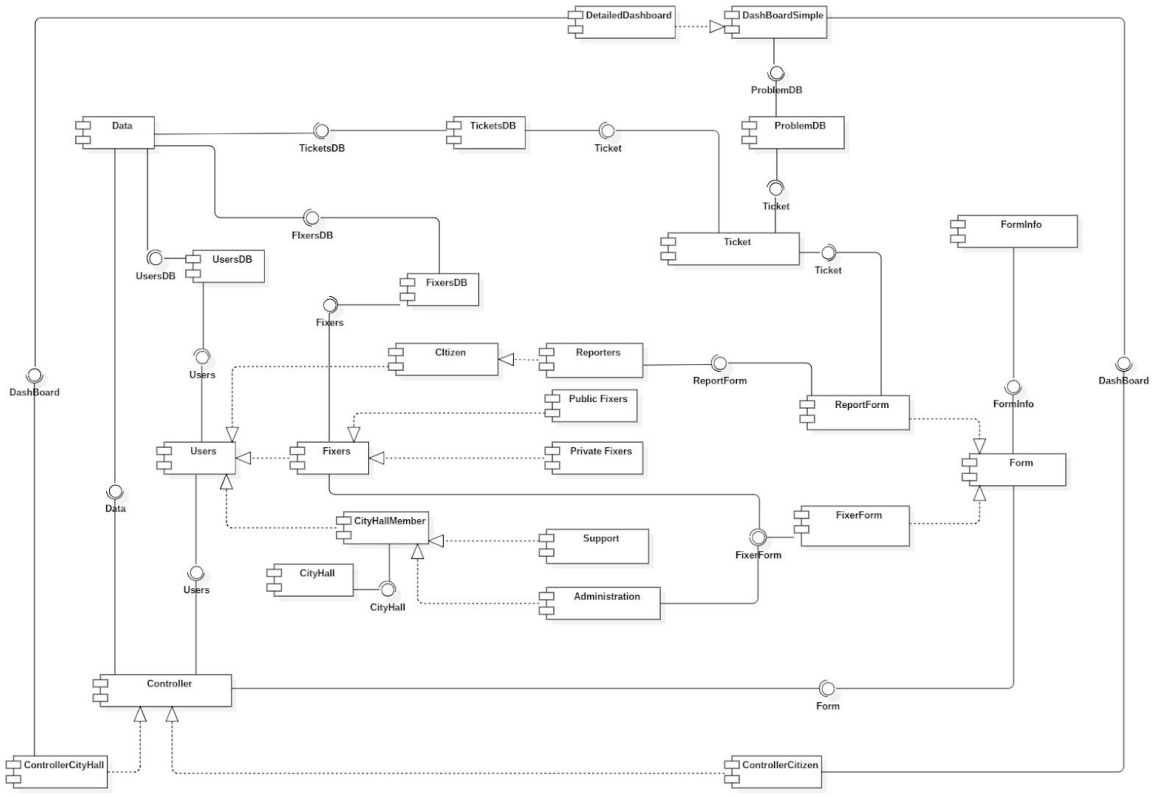
OCL

Na secção do ocl consideramos implementar apenas as classes necessárias para a implementação das nossas invariantes, pré e pós condições.

```
context CityHall::removeFunds(cost: Integer) : Integer
pre insufficientBalance:  cost <= self.balance

context ProblemsDB::listProblems()
pre hasProblems: self.problems->notEmpty()

context ProblemsDB::insertTicket(ticket : Ticket)
pre notbelongsToProblems: self.problems->excludes(ticket)
post belongsToProblems: self.problems->includes(ticket)

context FixersDB::isPrivate(fixer : Fixer) : Boolean
post notPrivate: self.fixers->excludes(fixer)

context TicketsDB::removeProblem(ticketId : String)
post notBelongsToTicketsDB: self.tickets->forAll(t: Ticket | t.ticketId <> ticketId)

context FixersDB
inv UniqueID:
   self.fixers->forAll( f1, f2: Fixer | f1 <> f2 implies
        f1.fixerId <> f2.fixerId )

context ControllerCityHall
inv needsToBeMember:
        self.data.usersDB.members->includes(self.currUser)

context TicketsDB
inv duplicateProblems:
        self.tickets->forAll( t1, t2: Ticket | t1 <> t2 implies
        t1.info.location <> t2.info.location
        and
        t1.info.type <> t2.info.type)
```

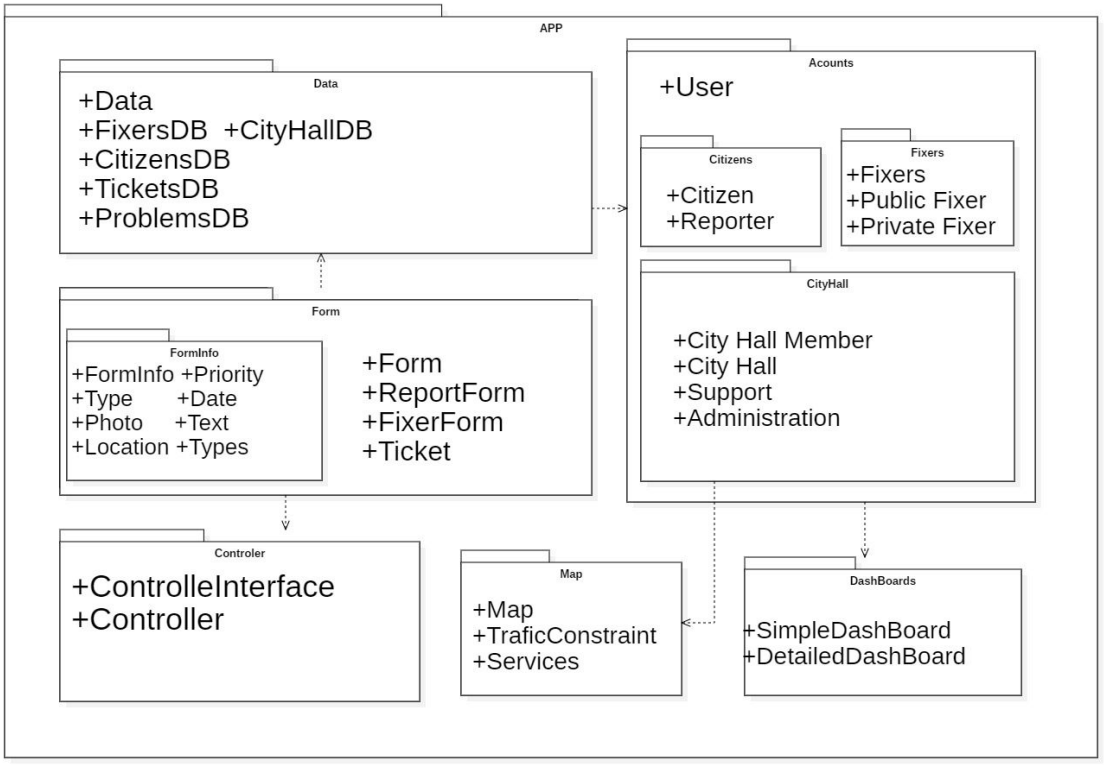
Component diagram

Neste diagrama uma coisa a observar é o facto de considerarmos que as classes boundary não são interfaces mas sim componentes em si. Fizemos isto por considerar que estas boundary são mais que simples interfaces.



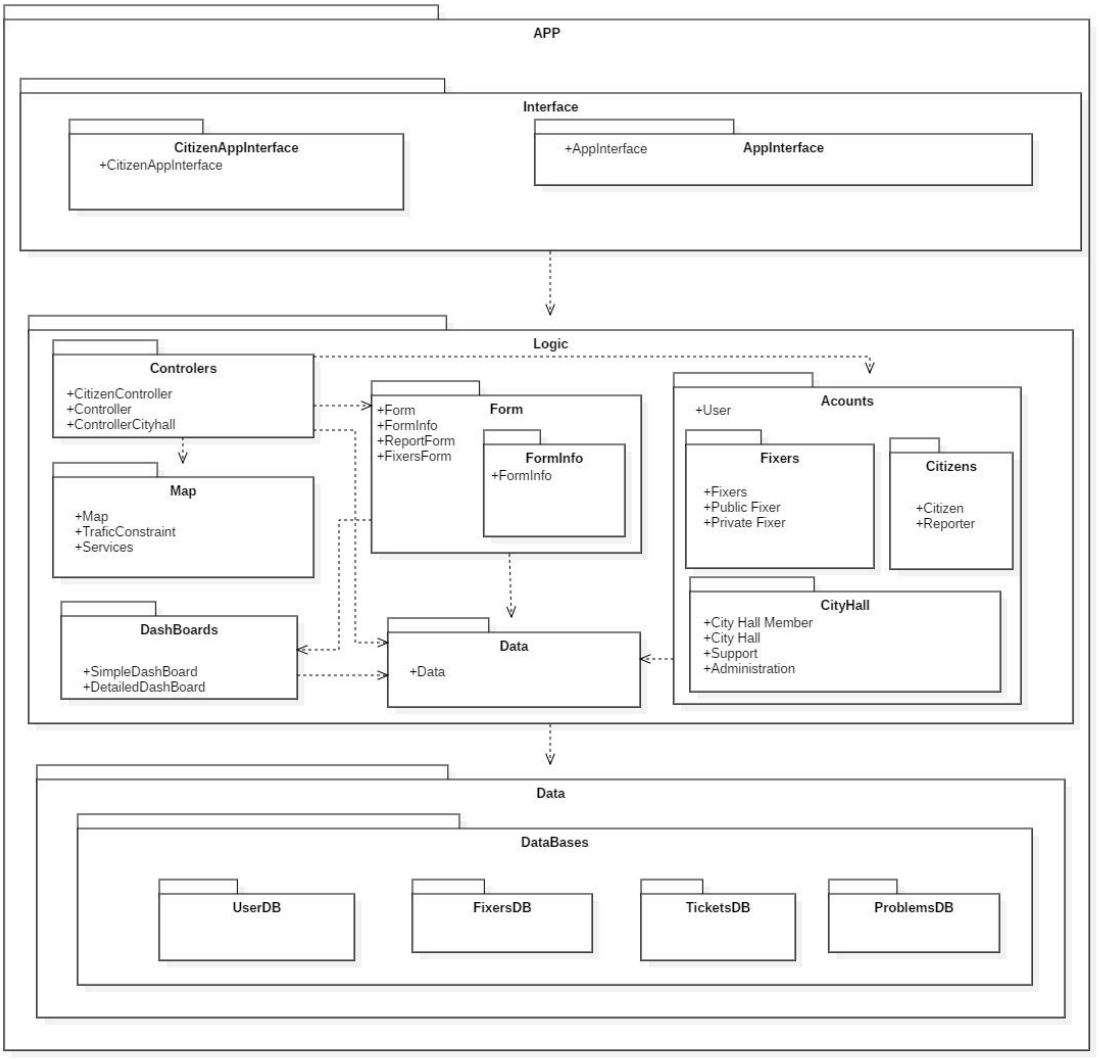
Package diagram

Devido às relações lógicas das várias classes considerámos que o agrupamento deverá ser feito por proximidade entre o conceito das classes. Por exemplo consideramos que faz sentido guardar todos os utilizadores que interagem com city hall numa única package.



Package Diagram Design:

Além de ser relevante lembrar o que escrevemos no primeiro package diagram achamos importante referenciar a novas três camadas. Separamos estas packages em 3 novas packages. A primeira a interface irá guardar todas as classes que servem de boundary, a segunda a lógic irá guardar as packages que contém os controllers e todas as outras entidades que manipulam através de lógica dados e a última a de databases contém todas as packages que contém bases de dados de algum tipo.



Conclusão

Ao concluir o projecto, notamos que neste estado ainda haveria muito trabalho pela frente por causa do nível de abstração dos diagramas que nos foram encarregados. Muitas das decisões que tomamos muito possivelmente seriam revistas à medida que o projecto fosse sendo implementado. Um exemplo é o primeiro caso de uso que descrevemos tem um fluxo alternativo que implica ver se a reclamação (ticket) já foi feita. A implementação desta verificação poderia ser feita de várias maneiras (desde algoritmos de neural networks até algoritmos na área da big data) que não conhecendo as limitações dos recursos disponíveis é nos impossível garantir com toda a certeza que a nossa implementação seria possível exatamente como queríamos. Outras dificuldades que tivemos foi na separação das várias funcionalidades da aplicação por várias classes. Esta área, admitidamente, poderia ser refeita de várias maneiras igualmente viáveis sendo algo subjectiva. Além do diagrama de classes o diagrama de pacotes (especialmente o primeiro) também poderia possivelmente ter várias outras alternativas viáveis, que em ambos dos casos poderiam ser consideradas uma vez que começada a implementação. Claro que a estrutura em si não sofreria muitas alterações mas alguns aspectos mais particulares dela poderiam (e possivelmente iriam) sofrer algumas pequenas alterações (atribuição de funções, tipos de funções e atributos, etc). O software que usámos para o trabalho, a versão dada do StarUml, também poderia ser consideravelmente melhor já que alguns aspectos do IDE fazem no por vezes extremamente irritante de se usar. Alguns destes aspectos é uma interface atrás do que seria desejável com funcionalidades óbvias e comuns noutras ferramentas (por exemplo páginas com zooms diferentes, o StarUml mantém o mesmo zoom para todas) e outras implementadas de forma amadora. Como por exemplo o zoom que em vez de fazer zoom no centro da imagem faz-o no canto. Outras implementações (por exemplo meter agentes no sequence diagram) são ridiculamente nada intuitivas.