Project presentation

myTaxiService

Authors:
Berbieru Mihai Adrian 854698
D’Onofrio Attilio 789614

Requirements Analysis and Specifications Document

Main goals

[G1] To allow the visitor to register;

[G2] To allow the unidentified user to log in;

[G3] To allow the registered user to make an instant booking;

[G4] To allow the registered user to make a reservation in a specified time;

[G5] To allow the registered user to delete the reservations he made;

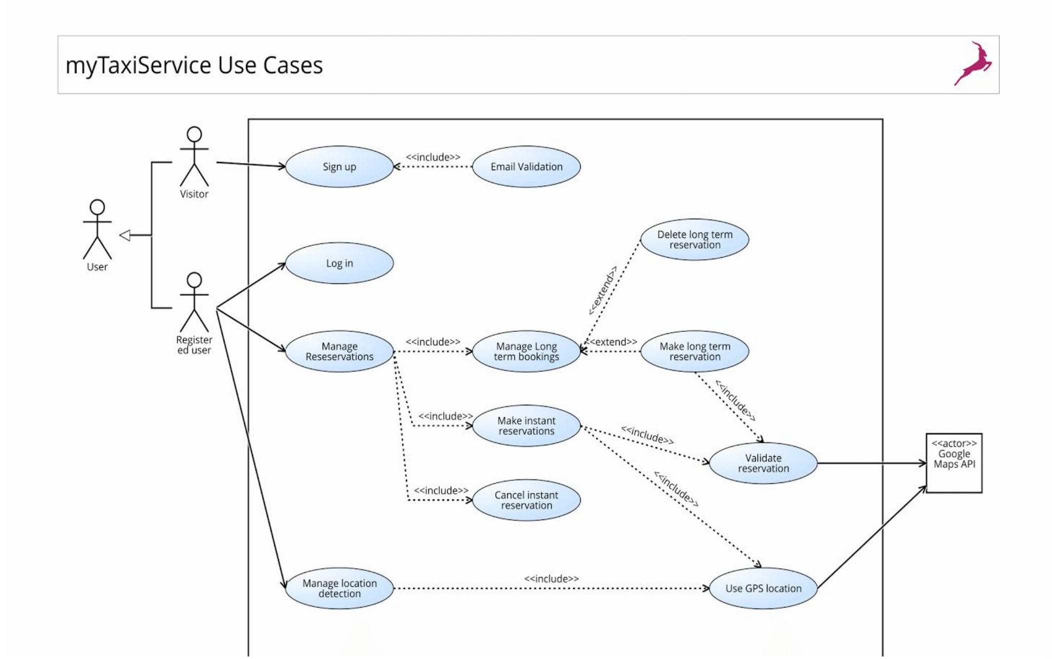
[G6] To allow the taxi driver to change its availability status;

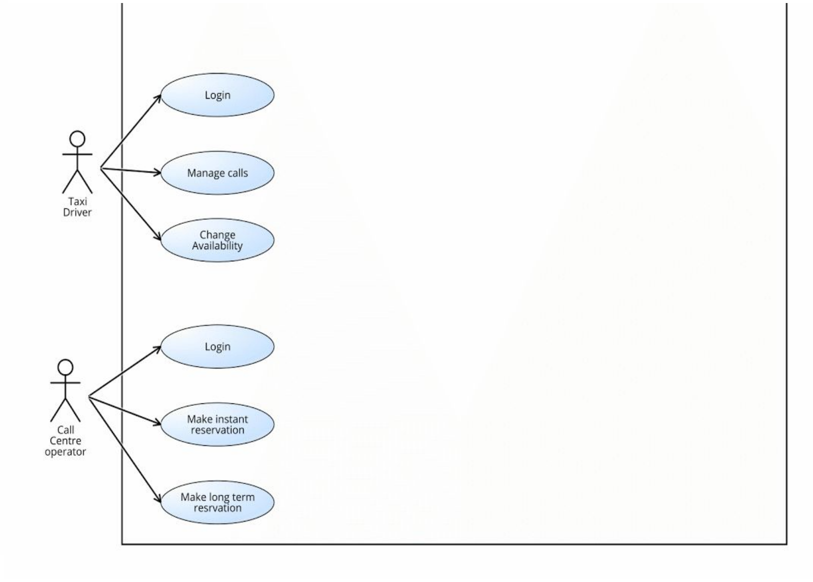
[G7] To allow the taxi driver to accept or decline a call;

[G8] To allow the taxi driver to get GPS directions until he reaches client’s position;

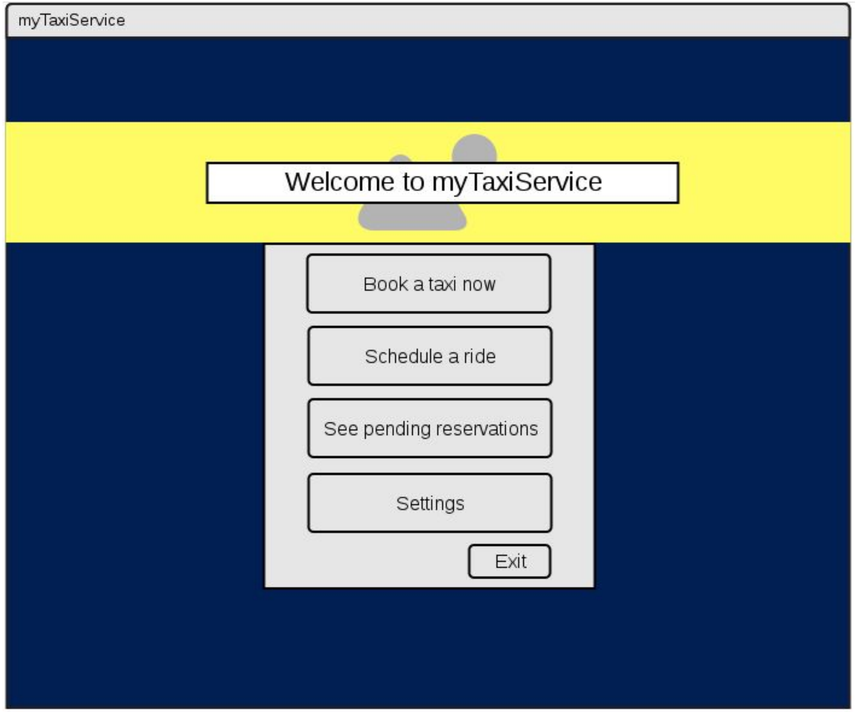
[G9] To allow the taxi driver to login.

Use case diagram

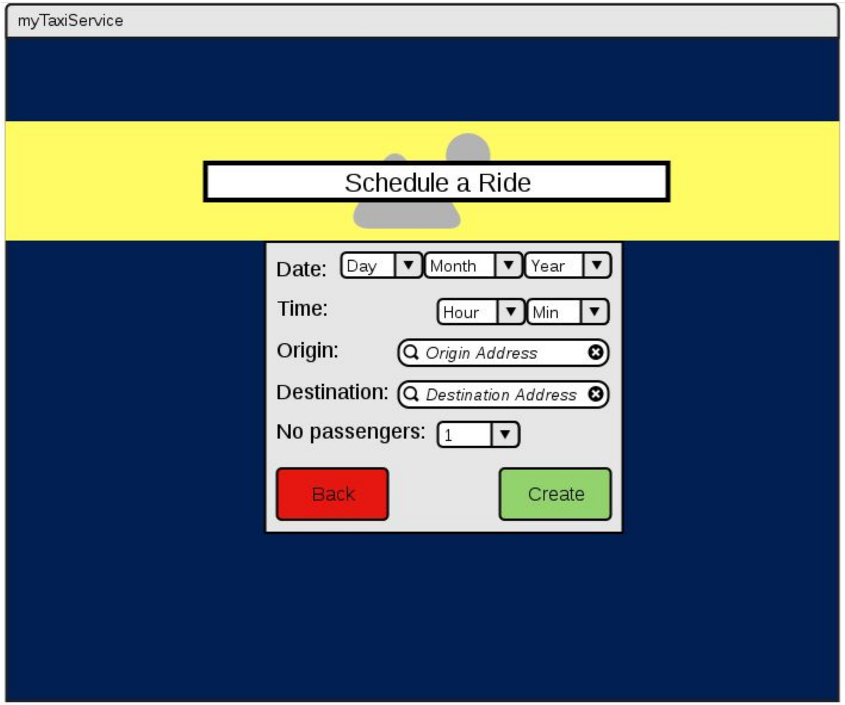




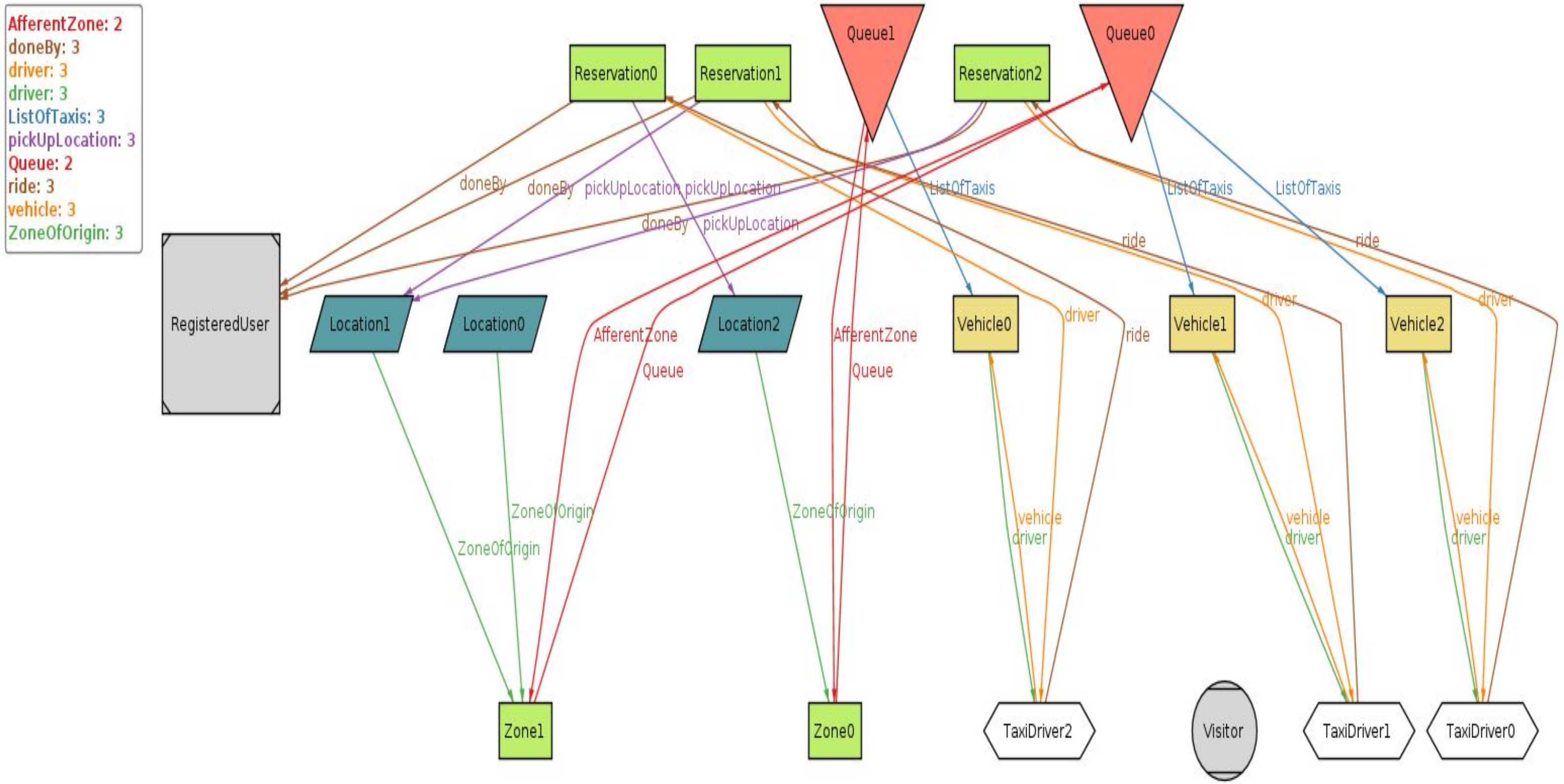
Mockup example 1: customer's home



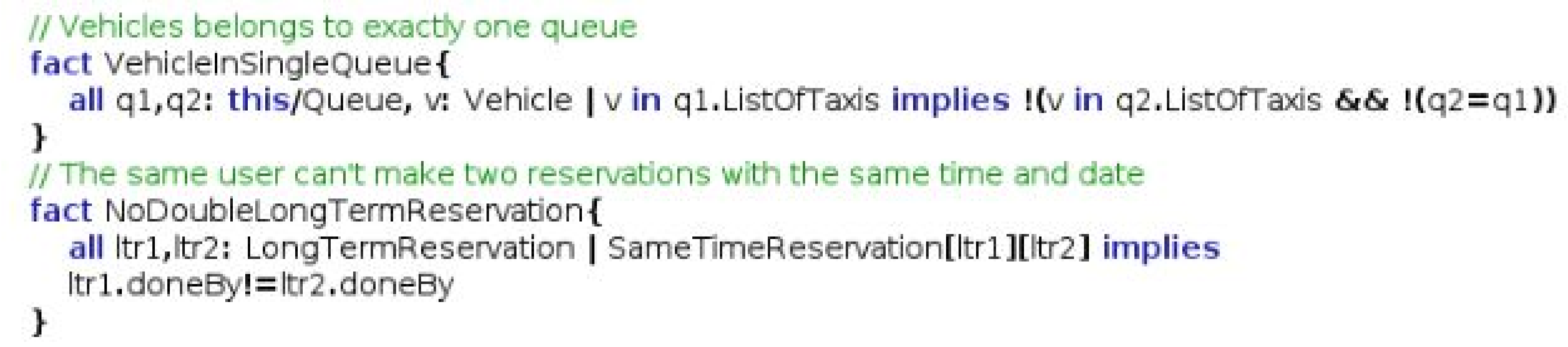
Mockup example 2: schedule a ride



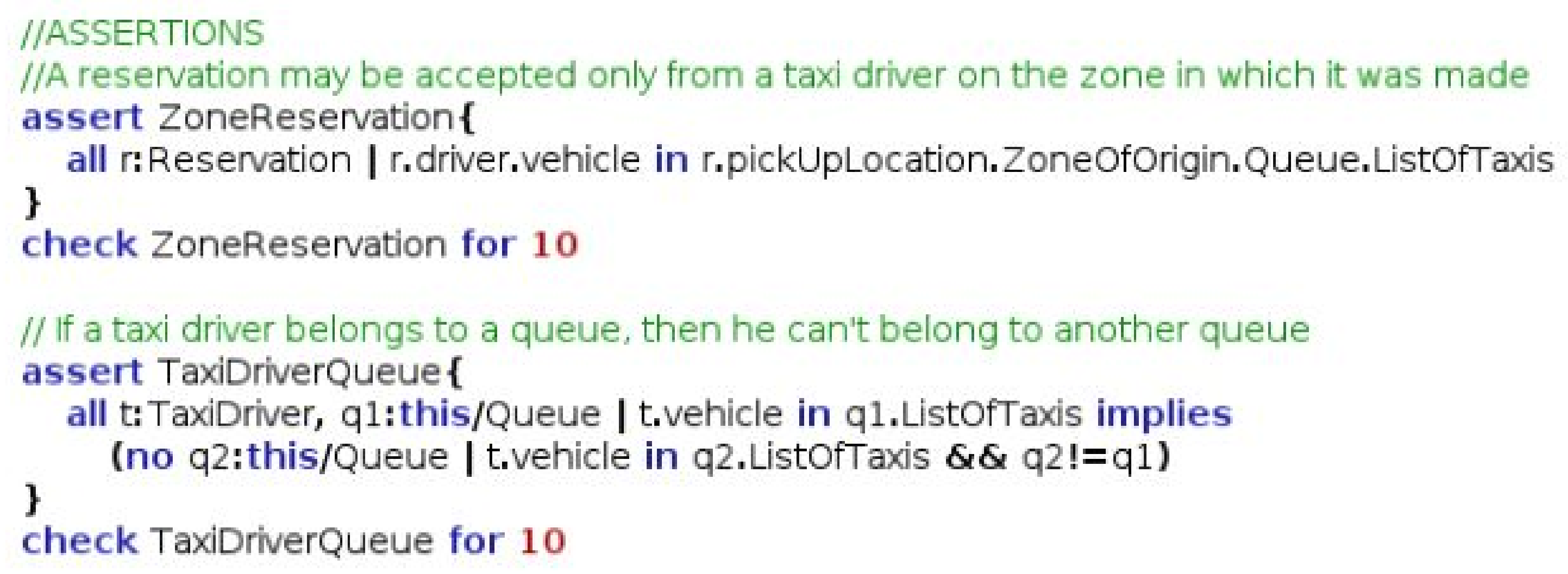
Alloy model: generated world



Alloy facts (just some of them)


```
// Vehicles belongs to exactly one queue
fact VehicleInSingleQueue{
   all q1,q2: this/Queue, v: Vehicle | v in q1.ListOfTaxis implies !(v in q2.ListOfTaxis && !(q2=q1))
}
// The same user can't make two reservations with the same time and date
fact NoDoubleLongTermReservation{
   all ltr1,ltr2: LongTermReservation | SameTimeReservation[ltr1][ltr2] implies
   ltr1.doneBy!=ltr2.doneBy
}
```

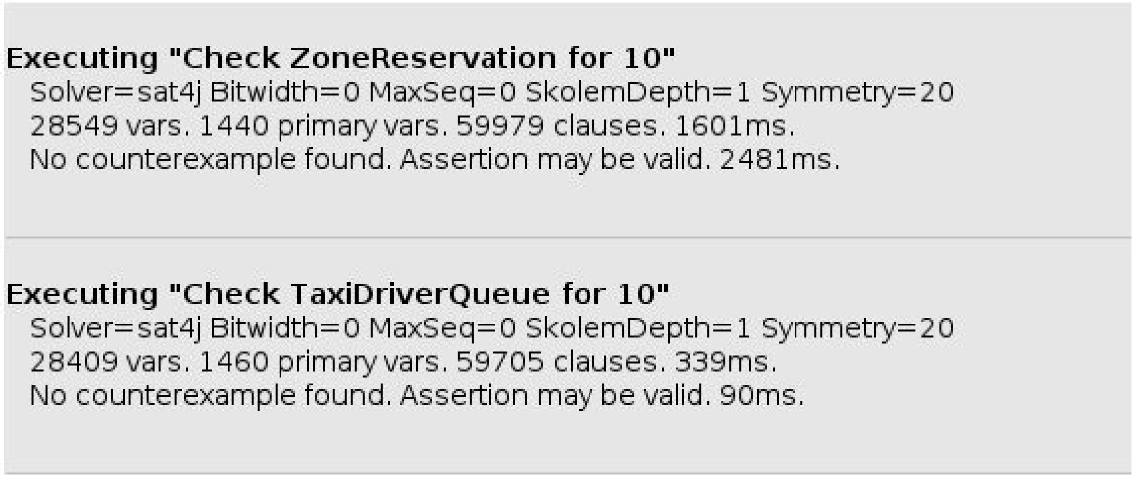
Alloy assertions



```
//ASSERTIONS
//A reservation may be accepted only from a taxi driver on the zone in which it was made
assert ZoneReservation{
    all r:Reservation | r.driver.vehicle in r.pickUpLocation.ZoneOfOrigin.Queue.ListOfTaxis
}
check ZoneReservation for 10

// If a taxi driver belongs to a queue, then he can't belong to another queue
assert TaxiDriverQueue{
    all t:TaxiDriver, q1:this/Queue | t.vehicle in q1.ListOfTaxis implies
        (no q2:this/Queue | t.vehicle in q2.ListOfTaxis && q2!=q1)
}
check TaxiDriverQueue for 10
```

Alloy assertions check



```
Executing "Check ZoneReservation for 10"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   28549 vars. 1440 primary vars. 59979 clauses. 1601ms.
   No counterexample found. Assertion may be valid. 2481ms.
```

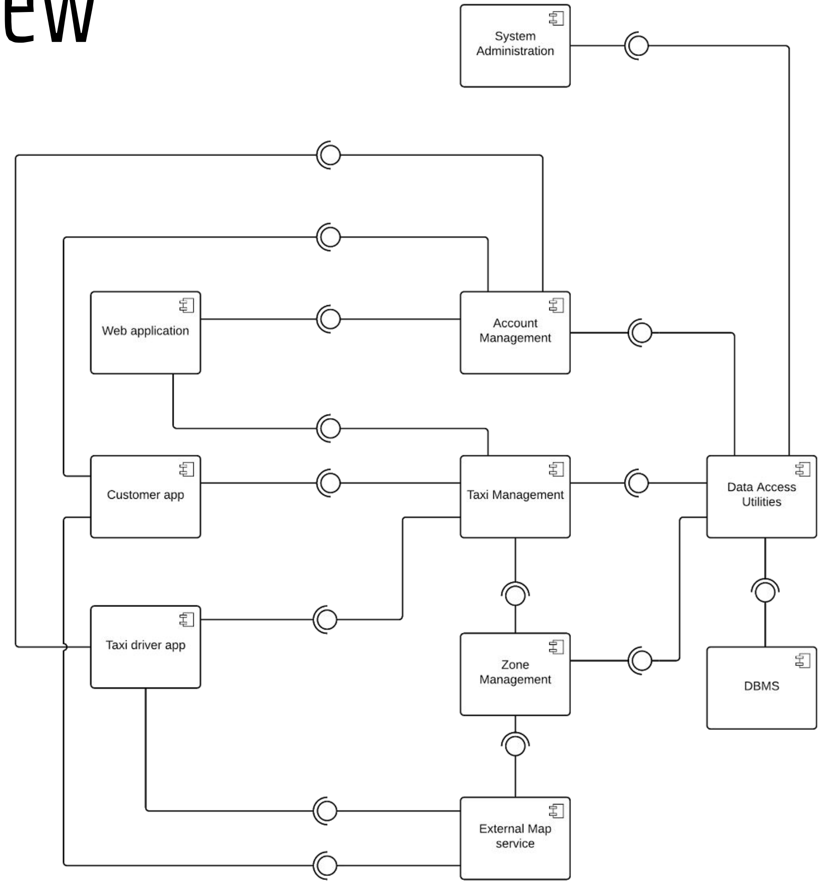
```
Executing "Check TaxiDriverQueue for 10"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   28409 vars. 1460 primary vars. 59705 clauses. 339ms.
   No counterexample found. Assertion may be valid. 90ms.
```

Design Document

Architectural style

- ***Client-Server***
- ***N-Tier physical architecture***.
- ***Thin Client***
- ***Plugins style***.

Component view



Algorithm design (I)

What does it happen “behind the curtains” when a customer makes an instant reservation?

1) A customer makes an instant reservation using the web application or the app
2) The system receives a request containing (among other data), the address of the customer
3) Address is extracted and associated to a city zone (binding is done by looking at a static table containing for each zone the associated addresses)
4) If the address does not belong to city perimeter, customer is warned to send a correct address
5) The system sends a message to each active driver and waits for a reply
6) The reply contains the driver ID and the current position
7) The system gathers all the pings and puts each driver in the queue according to its city zone
8) The driver at the top of the queue corresponding to customer’s city zone is able to answer to the ride request
9) If he accepts, the call is served. Otherwise, request is sent to the next driver in queue, and so on

Algorithm design (II)

The process described above works like that only when server starts, i.e queues are empty!

Most of the times indeed queue will be updated and not created from scratch

Queue is updated

- When a taxi driver switches the availability status
- When a taxi driver connects to the system
- When a taxi is disconnected (no ping within 120s)

Algorithm design (III)

- Ride requests made by customers follow the FIFO policy, so queues do
- Long term reservations work in the same way, but the reservation is stored and sent by the system itself ten minutes before the scheduled time

Code Inspection Document

Assigned class and methods

The portion of source code assigned to our group is inside the WebContainer class, whose path in the GlassFish project is *appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebContainer.java*
The methods contained in the WebContainer class we have inspected are the following ones:

1. *createJKConnector(NetworkListener listener , HttpService httpService)*
   Starts the AJP connector that will listen to call from Apache using mod_jk, mod_jk2 or mod_ajp.
2. *configureHttpServiceProperties(HttpService httpService , PECoyoteConnector connector)*
   Configure http-service properties.
3. *checkHostnameUniqueness(String listenerId , HttpService httpService)*
   Ensures that the host names of all virtual servers associated with the HTTP listener with the given listener id are unique.

The WebContainer class

- The class under exam is a container of utilities, in other words methods, to manage operations related to connections
- Creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks
- There is a wide array of constants, declared as static final attributes - usually belonging to the string type
- Singleton pattern applied: a single instance of WebContainer will be running on the server

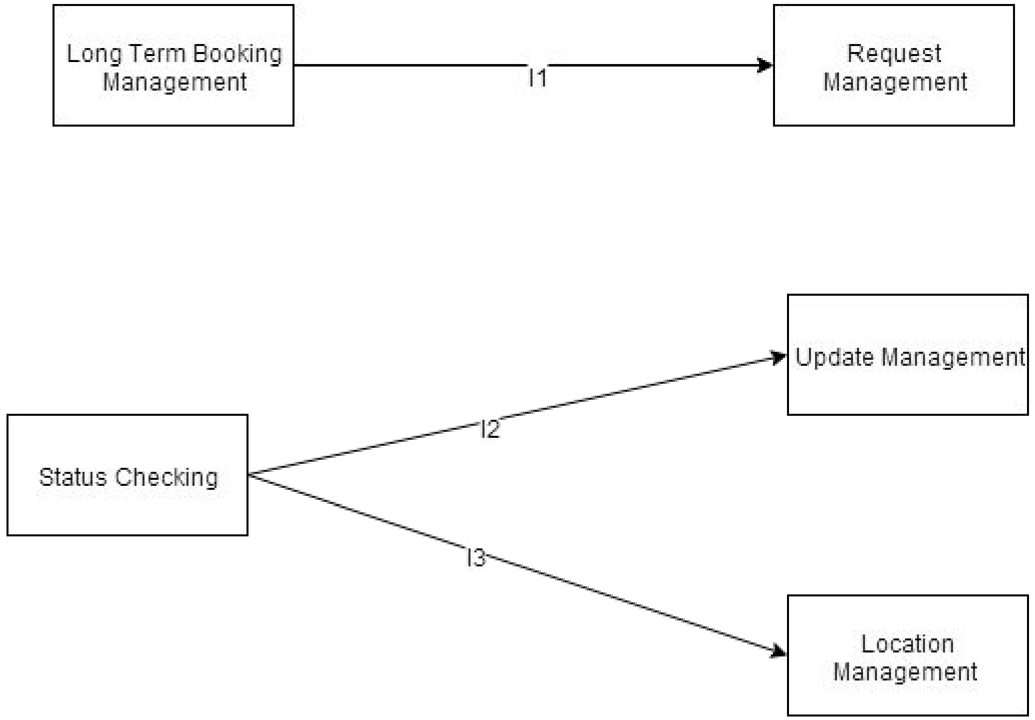
Issues found

The code inspection leads us to discover low relevancy issues and we think they do not affect the quality of code in a significant way

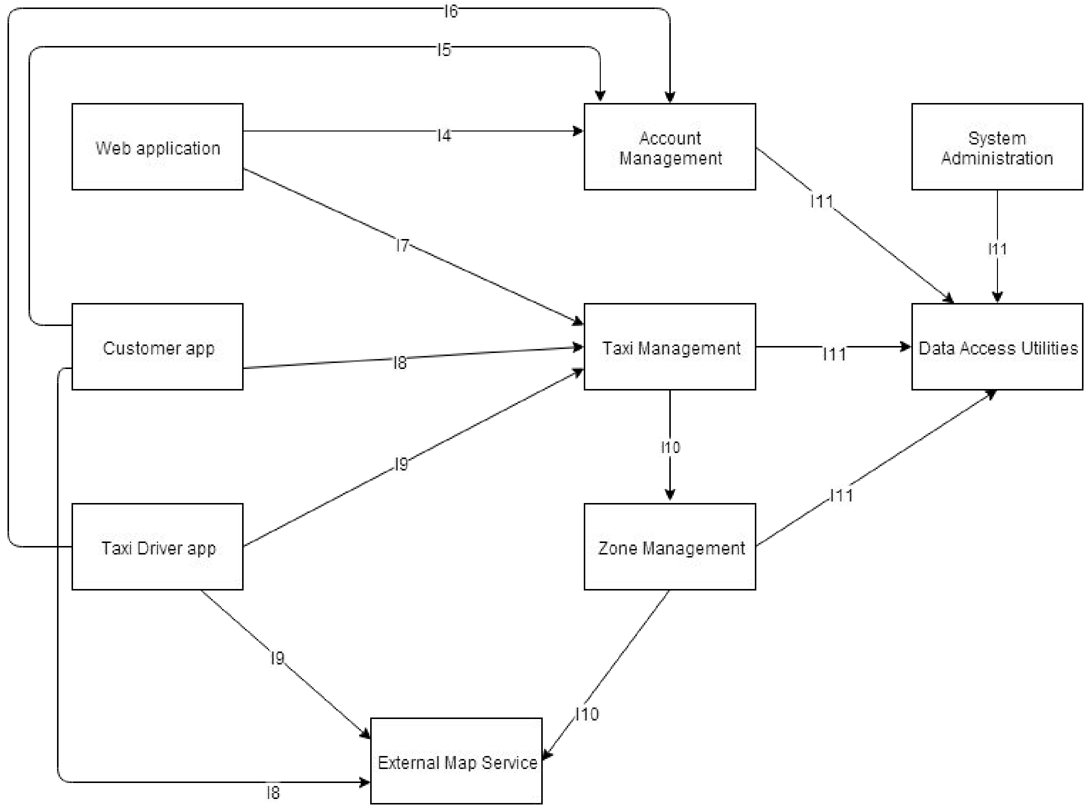
1. Many lines exceed 80 characters (but are below 120 characters)
2. Static and instance variables are declared in a random way
3. A couple of methods have a higher than average length
4. Seldom objects are compared used “==” and not “.equals”

Integration Test Plan Document

Integration testing strategy (I)



Integration testing strategy (II)



Example of test integration

Instant Booking with unavailable taxi and booking deletion

Purpose	To ensure a registered user can make an instant booking
External Dependencies	The user's smartphone is connected to the Web and GPS tracking is active
Test Description	1)Open and log in myTaxiService app 2)Select to book a taxi now 3)Fill out the form with required data 4)Send the request 5)Receive notification of taxi unavailability 6)Delete taxi request
Expected Results	As long as the provided address is valid, the system looks for a queue in that city zone and there are no taxis available. During the wait, user decides to delete the booking.

Project Plan Document

Function Points (I)

External Inputs

Customer/driver login	Low	6
Customer signup	Medium	4
Password recovery	Low	3
Instant booking	High	6
Long term booking	High	6
Driver status switch	Medium	4
Driver’s call management	High	6

Function Points (II)

External Inquiries

List long term bookings	Medium	5

External Interface Files

Position by map service	Medium	7
Step by step navigation	High	10

Function Points (III)

Internal Logical Files

Customer	Low	7
Taxi driver	Low	7
Mapping address/city zone	Low	7
Queue	High	15
Long term bookings	Average	10

Total FP: 103

COCOMO II

Scale drivers=16.84 and Cost drivers=0.84

*Effort=2.94*0.84*(5)^1.0784=14 [P/M]*

Duration=3.67(14)^0.31368=8.4 [M]*

N=Effort/Duration=14/8.4=1.72 [Person]

	Estimated Value	Real Value
Effort	14 Person/Months	2.2 Person/Months
Duration	8.4 Months	3 Months
Number of people	2	2

Risks (I)

Two kinds of potential risks:

- **General:** may arise throughout the whole development process
- **Task specific:** are related to a single task

Risks (II)

General risk

- Inexperienced team members
- Dependencies between parts of the project assigned to different team members (may lead to inconsistencies)
- Tight correlation with external factors:
  - Google APIs
  - Cloud hosting
  - Taxi laws and policies

Risks (III)

Task specific

- Algorithm description: requires a crystal-clear and in depth knowledge of system’s inner working, so it may be necessary to check and edit the work previously done in order to be consistent
- Code inspection: getting a clear picture of what the whole class does may not be straightforward; javaDoc may come in handy