



Final Presentation

MyTaxiService

Roberto Clapis, Erica Stella

Politecnico di Milano

12/02/2016






Table of Contents

RASD

Design Document

Test Document

Project Plan

Code Inspection





RASD

The RASD Document

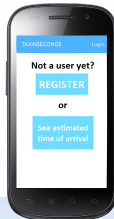
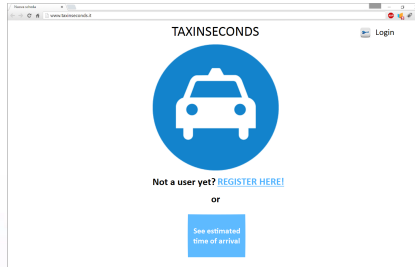
The approach

We tried to solve all the easy problems with a KISS approach, in order to focus on the actual difficulties.

- ▶ Login, logout, registration were kept as standard as possible
- ▶ Usability was kept in mind, customizability was not considered as a feature
- ▶ We tried to keep as little as possible the user inputs and let the application do the work

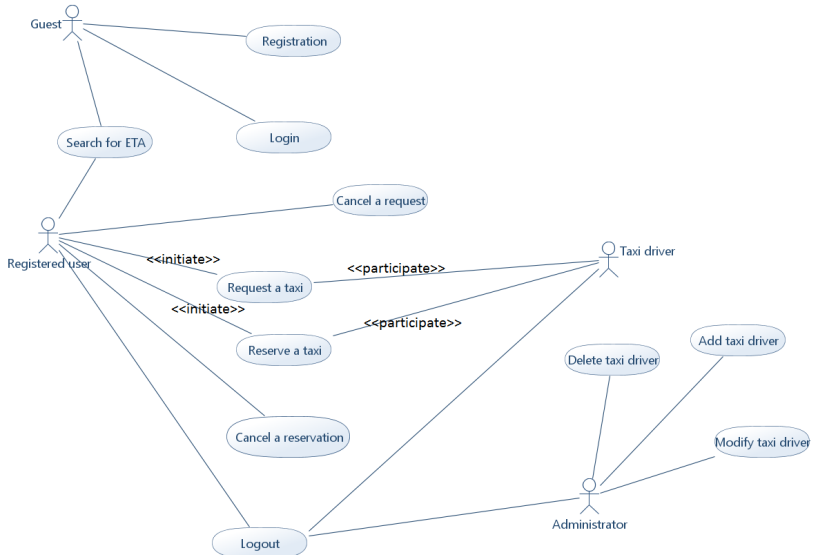
Two simple interfaces

Adaptive web-based UI to ensure maintainability for both the web UI and mobile UI



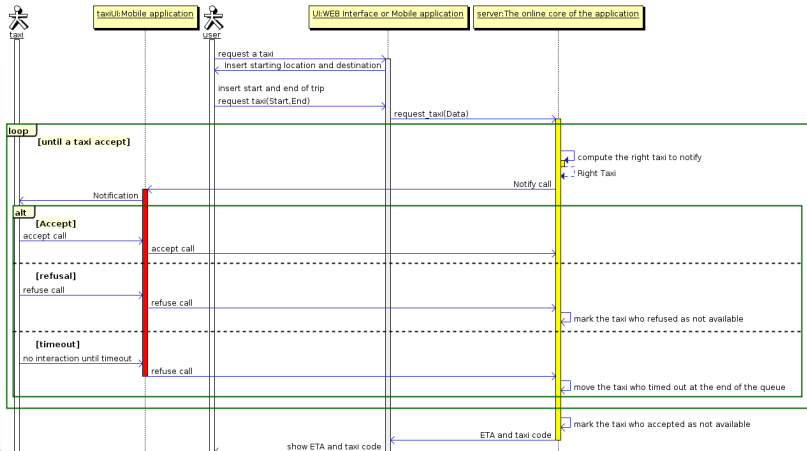
Use Cases

All the actions allowed were planned and analysed



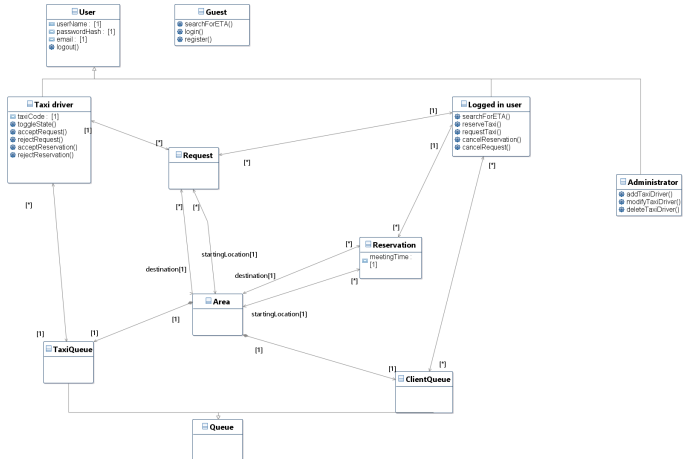
Use Cases

Each use case is analysed in depth with a sequence diagram

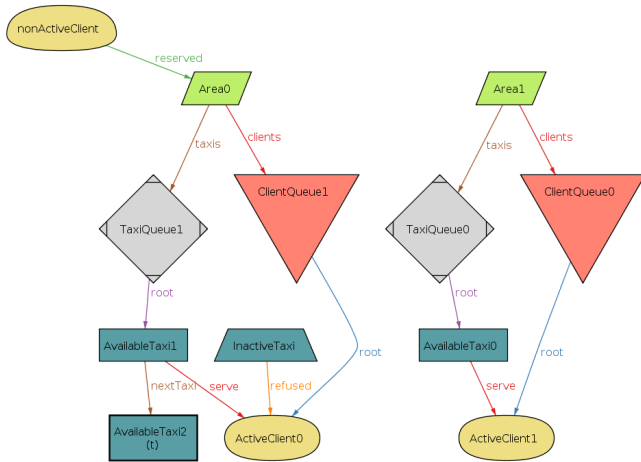


Class Diagram

The class diagram was kept as readable as possible, underlying the main decisions taken



Alloy





Design Document

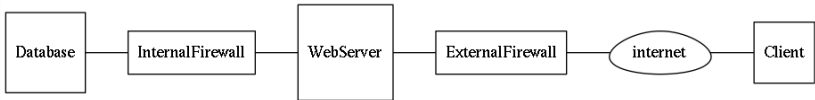
The Design

The architecture of the application is pretty standard

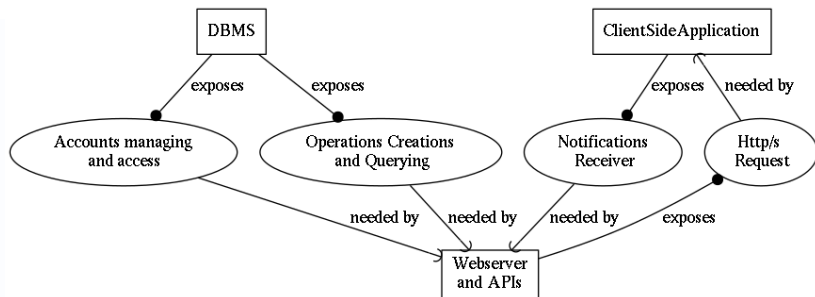
- ▶ Classical DMZ structure for the Network
- ▶ Database + Web server for the components

The Deployment

A standard DMZ + Internal network approach was used



The Component View



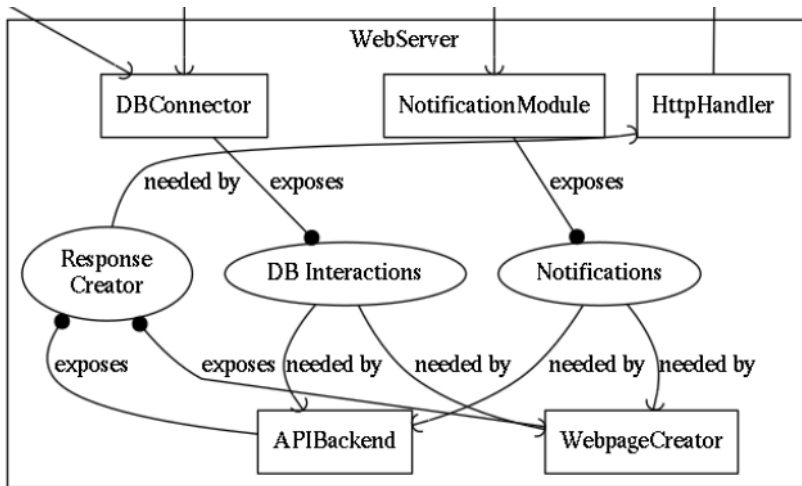
More on the Components

Extensibility and maintainability as targets

- ▶ Both mobile and web version are based on web technologies.
- ▶ The queues, reservations and requests are computed by the DBMS.
- ▶ The web server ignores the fact that the client is using a mobile app or a browser.
- ▶ The APIs query the DBMS in the exact same way the Web Server does.

More on the Components

Focus on the WebServer





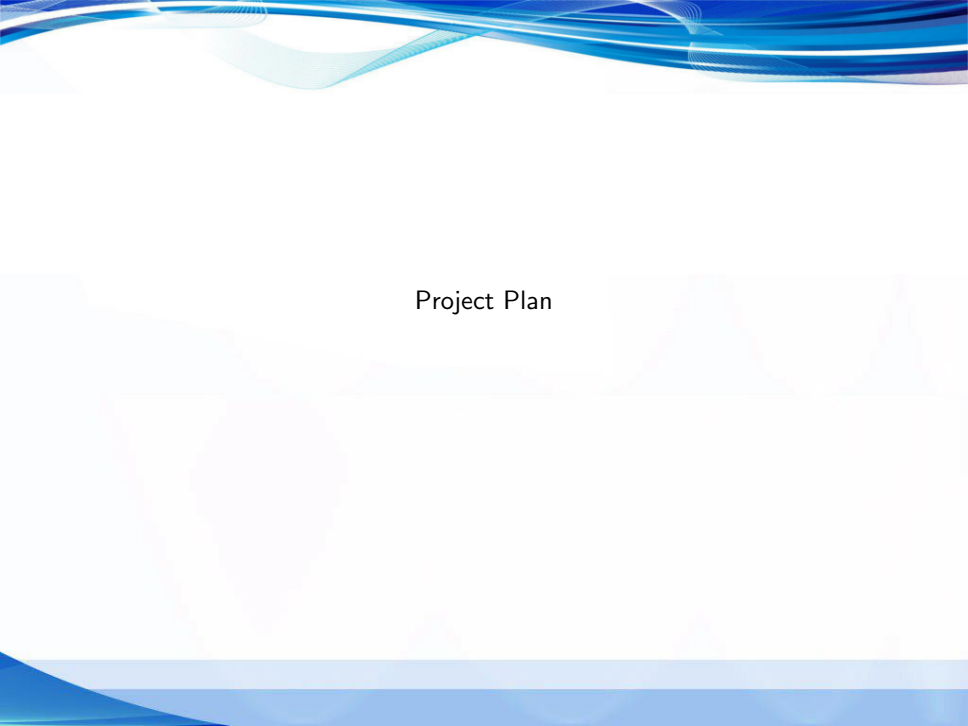
Test Document

The sandwich approach

The sandwich approach allows to first test the outer parts of the applications, which are easier to test with mock data and mock requests.

We chose to first test the UIs and the DBMS tables, in order to have a reliable surrounding when testing the WebServer part, which is the core of the application.

Once reached the Integration of the WebServer we decided to keep the same approach even for the internal components.



Project Plan

Function Points

152 FP in Total:

- ▶ Internal Logical File 45 FP
- ▶ External Input 70 FP
- ▶ External Output 19 FP
- ▶ External Inquiry 18 FP

COCOMO

- ▶ ~7000 estimated thousands of **Source Lines Of Code**
- ▶ ~25 Person-Month
- ▶ 3 People for a total of 10 Months

Risks

- ▶ Personnel shortfall due to any cause (e.g. illness)
- ▶ Capacity shortfalls of the entire system
- ▶ Problems with the mobile application development framework
- ▶ Taxi drivers don't cooperate in the use of the application
- ▶ The application is not well-accepted by the public



Code Inspection

The Checklist

The Style is ok

- ▶ Naming conventions were mainly respected
- ▶ Braces and indentation respected the standard except for a couple of lines
- ▶ File organization and white spaces were correctly used. With the exception of a comment line that was too long and a couple of white lines to be removed.
- ▶ High level breaks were used in code lines

The Checklist

But the style is not everything

Documentation is mostly absent or incomplete

There is a condition of an if that was commented out with no reasons to explain why

One of the functions was definitely too complex, it represented the logical function of three different methods, one after the other, separated by a comment.

An other couple should have been splitted in at least two simpler ones.

The Checklist

A couple of exceptions

A couple of exceptions were caught but not handled, and both of them have a comment stating

```
try {  
    reader.close();  
} catch (Exception e) {  
    //ignore?  
}
```

To close up

Closing is important

There is a method that opens a list of inputstreams from an archive file and if an exception is thrown the handler just closes the last inputstream opened.

According to the Java Documentation closing every inputstream that refers to a file closes the file itself.

Even if a failure in that point of the application probably implies a failure of the whole process, that will soon close and free the file descriptors, this is a terrible practice and it should be avoided.



That's all!