



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 goals

- ▶ Provide an easy way to request a taxi.
- ▶ Provide an easy way to reserve a taxi.
- ▶ Guarantee a fair management of the taxi queues.
- ▶ Create an extensible system that allows expansion and interactions with other services.

Domain Assumptions

The main assumptions

- ▶ All taxi drivers will have a phone with active GPS functionality
- ▶ The GPS information about the position of the available taxi drivers sent to the system will always be accurate
- ▶ Taxi drivers' phones will always have an internet connection while myTaxiService is running
- ▶ Users will enter a valid email address during registration
- ▶ Users who have requested or reserved a taxi will always be present when the taxi arrives.

Requirements

The main functional requirements

For both the API and UI interfaces:

- ▶ The system will be able to calculate and show the ETA
- ▶ The system will provide a functionality to request a taxi at the given starting location
- ▶ The system will provide a functionality to reserve a taxi at the given starting location and meeting time
- ▶ The system will provide a functionality to see currently active reservations and requests
- ▶ The system will provide a functionality to notify users of the code of the incoming taxi

Requirements

Non functional requirements

▶ *Availability*

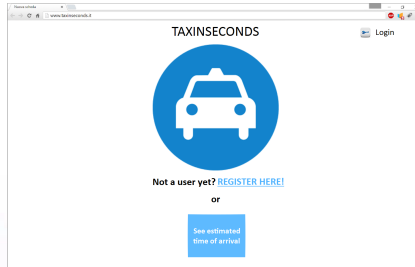
- ▶ The server of the application must always be available
- ▶ The app must never freeze
- ▶ The system must store all of its data in an always-reachable database
- ▶ Regular backups will be made in order to reduce or prevent data loss

▶ *Security*

- ▶ In no situation sensible data will pass through an insecure channel

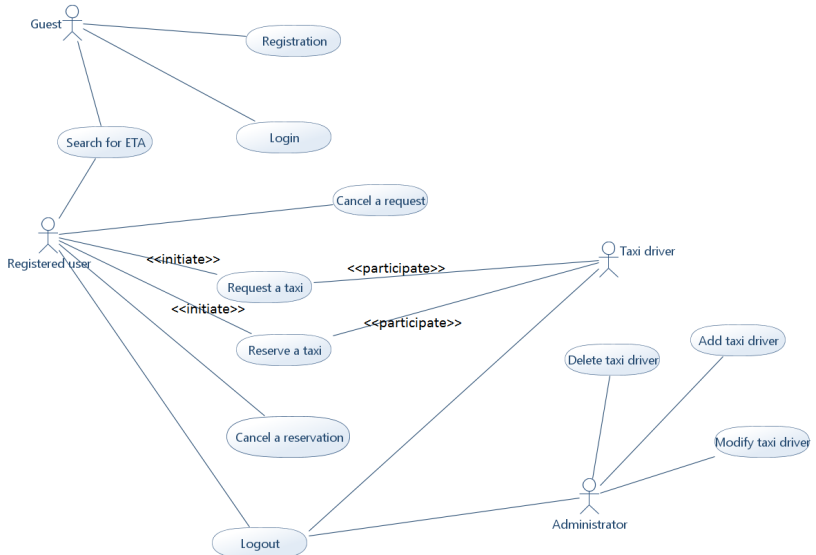
Two simple interfaces

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



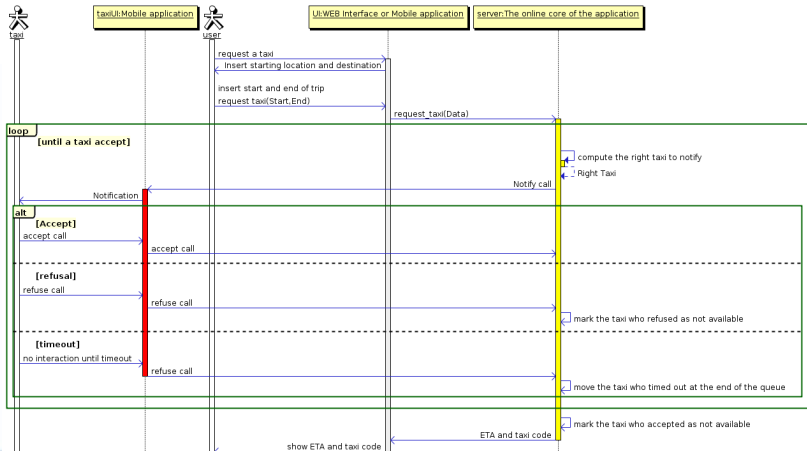
Use Cases

All the actions allowed were planned and analysed



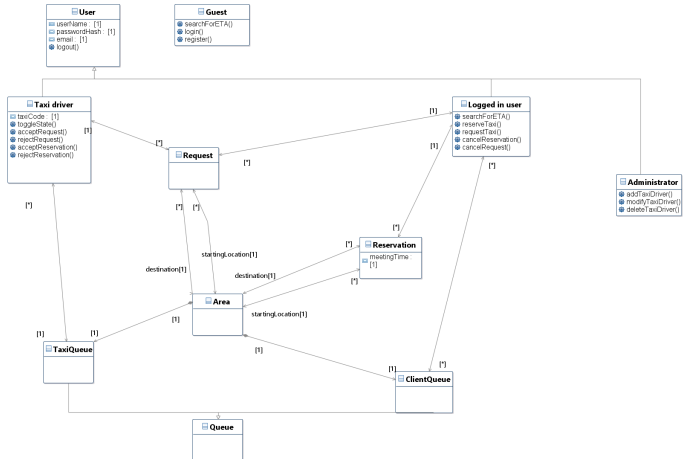
Use Cases

Each use case is analysed in depth with a sequence diagram

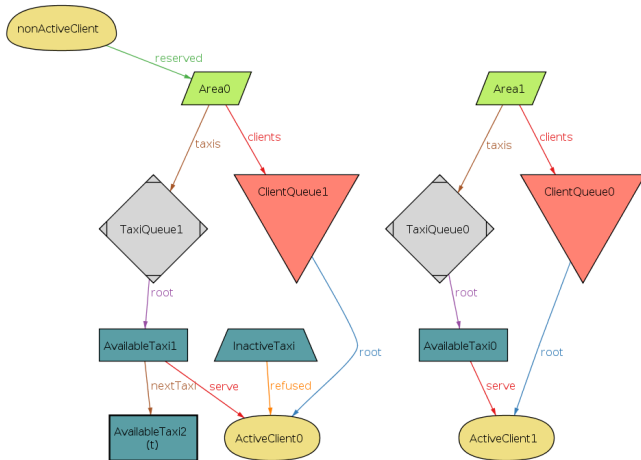


Class Diagram

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



Alloy





Design Document

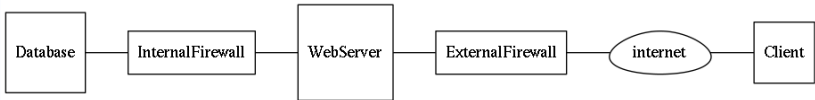
The Design

The architecture of the application is pretty standard

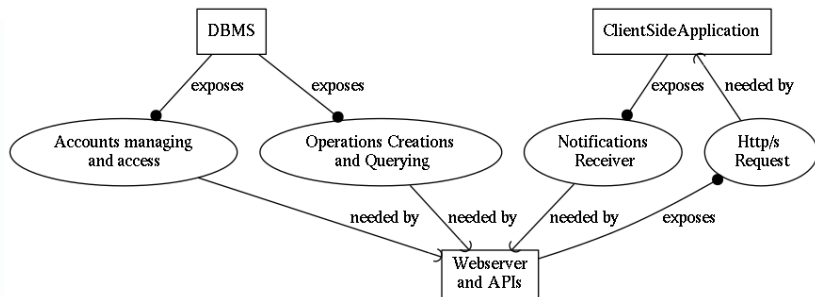
- ▶ Classical DMZ structure for the Network
- ▶ Database + Web server for the components

The Deployment

A standard DMZ + Internal network approach was used



The Component View



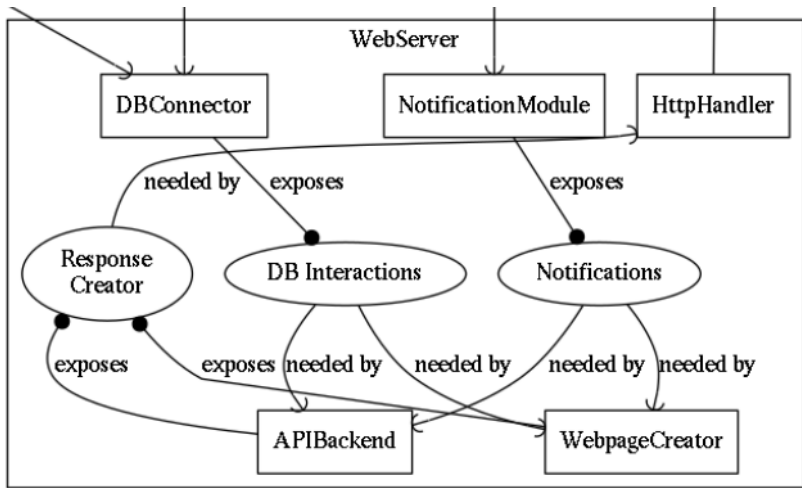
More on the Components

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

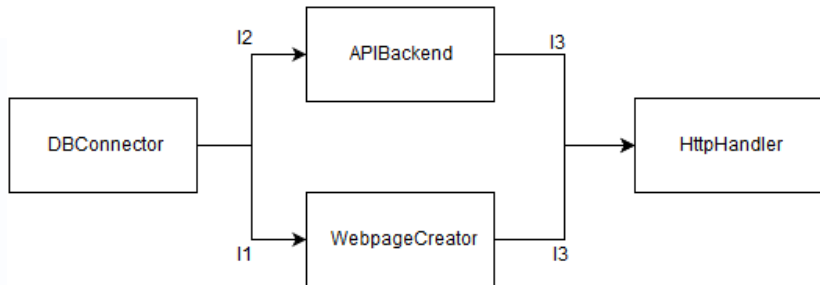
Focus on the WebServer



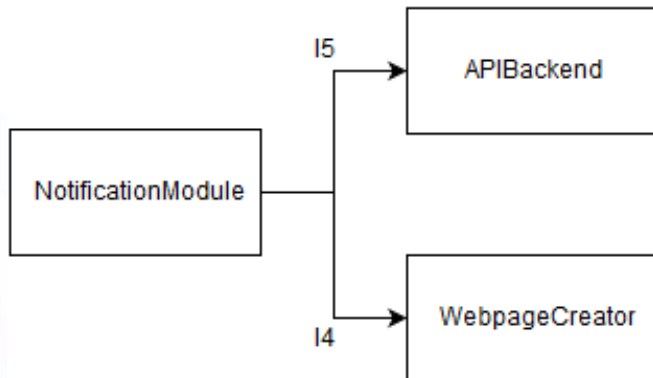


Test Document

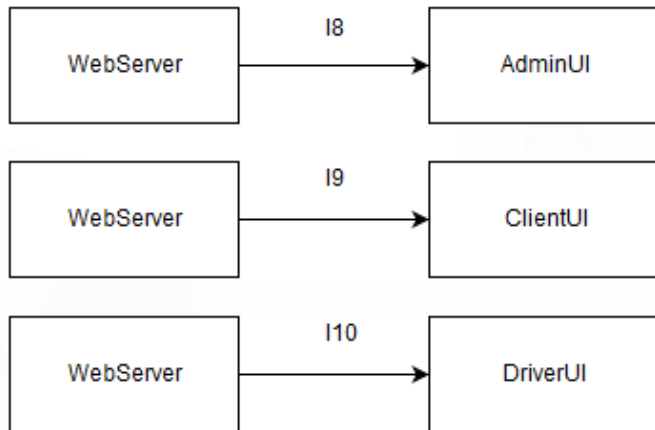
WebServer



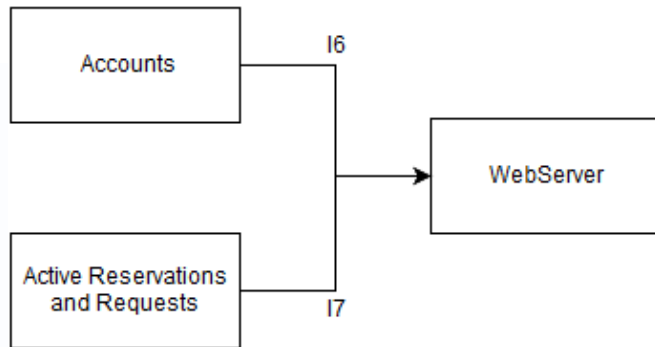
WebServer

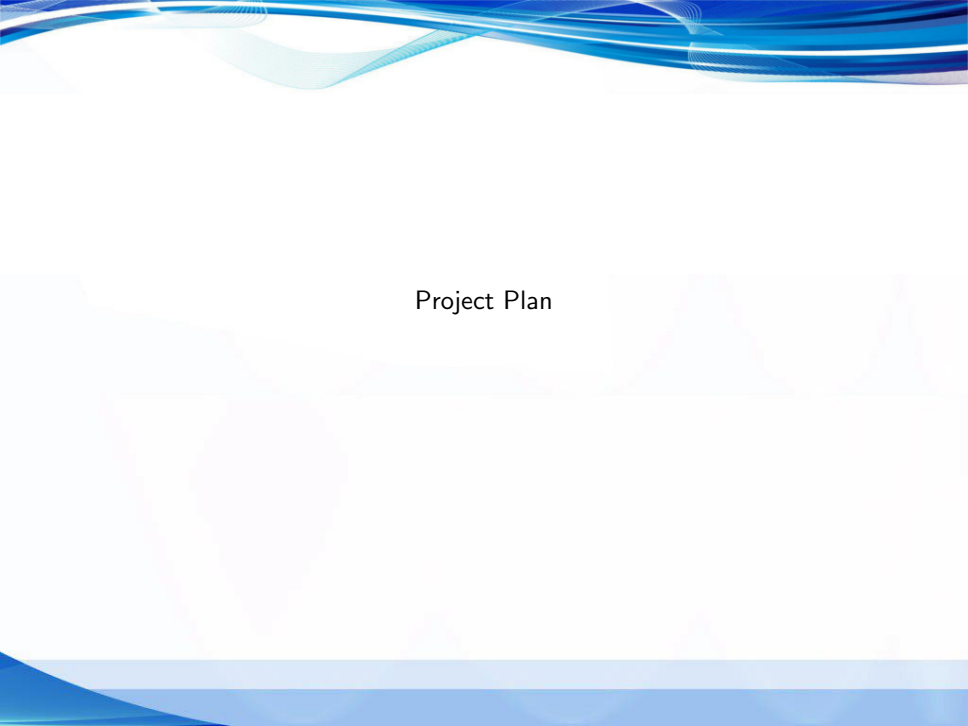


Webserver → UI



DBMS → Webserver





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

- ▶ $152 * 46 \cong 7000$ estimated **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 Class

The class was `ConnectorDeployer` from the package `com.sun.enterprise.connectors.module` The methods were

- ▶ `private void registerBeanValidator(String rarName, ReadableArchive archive, ClassLoader classLoader)`
- ▶ `private List<String>getValidationMappingDescriptors(ReadableArchive archive)`
- ▶ `public void event(Event event)`

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

```
if (/*env.isDas() && */  
    Deployment.UNDEPLOYMENT_VALIDATION.equals(event.type())) {
```

- ▶ 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.

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.



That's all!