# Design document


Figure 1: Politecnico di Milano

**Version 1.0**


- Claudio Cardinale (mat. 849760)
- Gilles Dejaegere (mat. 853950)
- Massimo Dragano (mat. 775392)

# Contents

# Introduction

## Purpose

The purpose of this document is give more technical details than RASD about MyTaxiService system.
This document is address to developers. In fact we want to identify:

- High level architecture
- Design patternes
- Main components with our interfaces with each other

**COMPLETE**

## Scope

We will project and implement myTaxiService, which is a service based on mobile application and web application, with two different targets of people:

- taxi drivers
- clients

The system allows clients to reserve a taxi via mobile or web app, using GPS position to identify client's zone (but the client can insert it manually) and find taxi in the same zone.
On the other side the mobile app allows taxi drivers to accept or reject a ride request and to communicate automatically their position (so the zone).
The clients are not registered since the company wants a quick system so if there is a registration a lot of clients won't use the app. So the clients must insert their name and phone number each time (this is faster than creating an account and logging each time).

The system includes extra services and functionalities such as taxi sharing.

The main purpose of the system is to be more efficient and reliable than the existing one in order to decrease costs of the taxi management and offer a better service to the clients. **KEEP OR REMOVE?**

We will design:

- Critical internal components
- Interface with external services like SMS gateway
- Interface with the old system

**COMPLETE**

## Definitions, acronyms, abbreviations

- RASD: Requirements analysis and specifications document
- DD: design document
- SMS: short message service; it is a notification sent to a mobile phone, we need an SMS gateway to use it.
- SMS gateway: it is a service which allows to send SMS via standard API.
- API: application programming interface; it is a common way to communicate with another system.
- Push notification: it is a notification sent to a smartphone using the mobile application, so it must be installed.
- Push service: it is a service that allows to send push notifications with own API
- Matching itineraries: two itineraries (A and B) are matching if one of the two following conditions are fulfilled:
    1. B is included in A: the starting point and the ending point of the itinerary B are both close to the itinerary A and the starting point of B is closer to the starting point of A than the ending point of B.
    2. The beginning of B is the end of A: the starting point of B is close to the itinerary A and the ending point of A is close to the itinerary B.
    3. A is included in B: see condition 1.
    4. The beginning of A is the end of B: see condition 2.

**COMPLETE INSERTING OTHER GLOSSARY FROM RASD**

## Reference documents

- RASD produced before **Write version number**
- Specification Document: Assignments 1 and 2 (RASD and DD).pdf
- Structure of the design document.pdf

## Document structure

- Introduction: in this section we introducing the design document, saying why we do it and which parts are covered from it that are not covered by RASD
- Architecture Design: this section is divided into two parts:
    1. High level design
    2. Architecture chosen presented via diagrams
- Algorithms Design: in this section we describe the most critical parts via some algorithms. We use code not completed since we want just to show the most important parts

- User Interface Design: we inserted mockups and user experience explained via UX and BCE diagrams
- Requirements Traceability: This section aims to explain how the decisions taken in the RASD are linked to design elements

**WRITE MORE**

# Architectural design

## Overview

We have a three tier architecture.

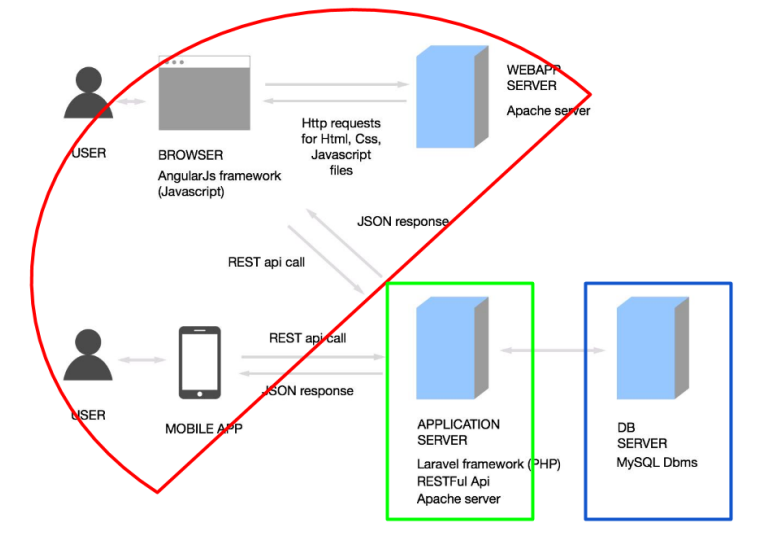


Figure 2: General architecture

On the client we don't have static GUI but a dynamic GUI that is generated on client side, in fact in the client there is a module that interacts with the application server via RESTful API.
With this architecture we can easily move this application to a cloud system, for example to amzon AWS where we have dedicated cloud servers with load balance for database and other for application logic on demand.

**Write more and make graph of interaction with the old system KEEP OR REMOVE?**

## High level components and their interaction

The high level components architecture is composed of four different elements types. The main element is the a singleton, the central. The central receives request or reservations from other elements, the clients. The client can initiate this communication from his mobile application or from the webpage of the application. This communication is made in a synchronous way since the client, who initiates the communication, has to wait the answer of the central that

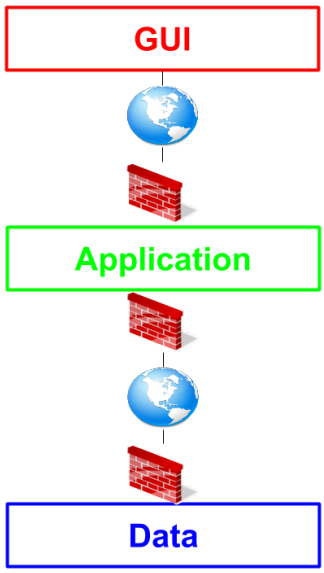


Figure 3: Tiers

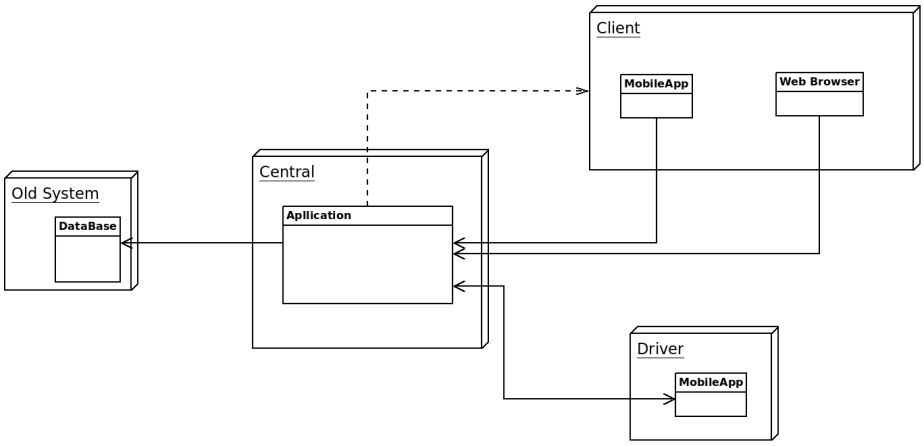


Figure 4: High level components

acknowledge him that his request has been taken into account. The Central will later send an asynchronous message to the client in the form of a sms to inform him about the code of the incoming taxi as well as the ETA.
The central communicates also with a third type of component, the taxi drivers. The central can send synchronous messages to the taxi drivers to propose them different request that the taxi driver can accept or reject. The taxi driver can send two type of messages to the central. First, he can change his availability. This must be done in a synchronous way since the central may have to respond with the position of the taxi driver in the waiting queue. The taxi driver can also send his position to the central. This can be done asynchronously. Taxi drivers also have to communicate with synchronous message with the central to log in.
A final type of components is also present, the old application. The old application still manages the registration of the new taxi drivers. Therefore, the central communicates synchronously with the old data base to extract the taxi drivers.

## Component view

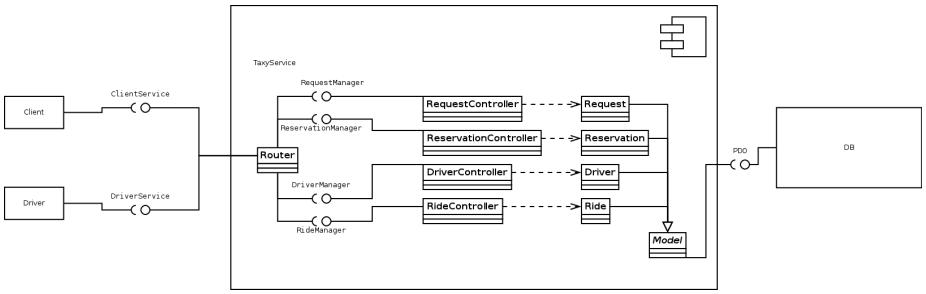


Figure 5: Component view

## Deploying view

### Talk about request merging

## Runtime view

## Component interfaces

## Selected architectural styles and patterns

### Overall Architecture

Our application will be divided into 3 tiers:

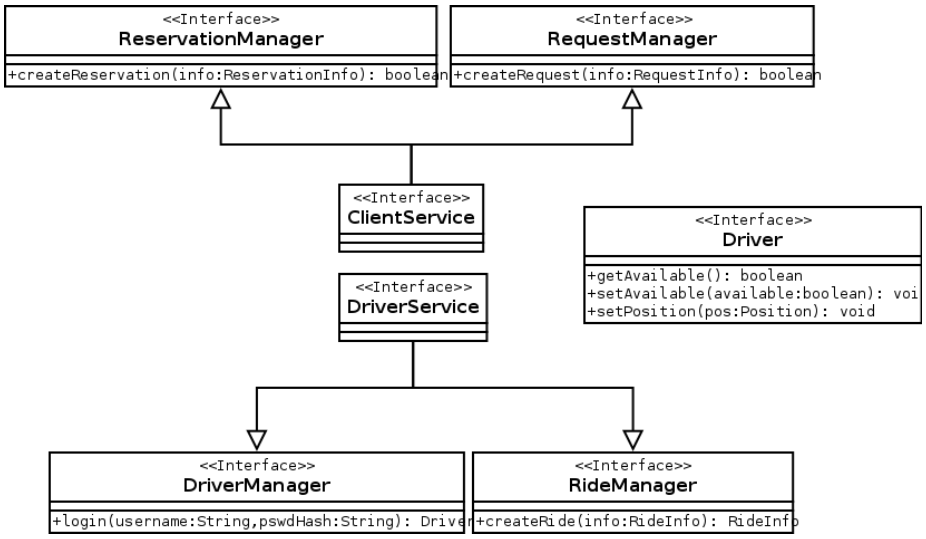


Figure 6: Component interfaces

1. Database ( DAL: Data Access Layer )
2. Application Logic ( BLL: Business Logic Layer )
3. Thin Client ( a simple and easy interface to BLL )

**Protocols**

Our tiers are connected through network and exchange data with the following protocols.

**PDO: PHP Data Objects** used by the BLL to communicate with the DAL.

currently supported databases:

- DBLIB: FreeTDS / Microsoft SQL Server / Sybase
- Firebird: Firebird/Interbase 6
- IBM (IBM DB2)
- INFORMIX - IBM Informix Dynamic Server
- MYSQL: MySQL 3.x/4.0
- OCI: Oracle Call Interface
- ODBC: ODBC v3 (IBM DB2 and unixODBC)
- PGSQL: PostgreSQL
- SQLITE: SQLite 3.x

**RESTful API with JSON** used by clients ( both mobile apps and web browsers ) to interact with the BLL. API calls that need authentication are required to authenticate via HTTP basic authentication for each request. exchanged data will be secured using SSL.

as now ( v1 ) our exposed methods are the following:

- api/v1/driver [auth]
  - GET: get driver info
  - PATCH/PUT: update driver data ( position and available status )
- api/v1/request
  - POST: create a new request
- api/v1/reservation
  - POST: create a new reservation
- api/v1/ride
  - POST: create a new ride

**INSERT API TO SAY THAT A RIDE IS TERMINATED EVEN IN INTERFACES DESIGN** and in mockup and ux

#### Design patterns

**MVC** Model-View-Controller pattern has been widely in our application.

Our Application server will use the Laravel PHP framework, which is an MVC framework. Our Web interface will use AngularJS, which is an MVC framework.

**Adapter** Adapters are used in our mobile application to adapt the Driver interface to the RESTful API one.

**CLIENT/SERVER ?**

**EXPLAIN WHY**

### Other design decisions

# Algorithm design

**ideas:**

- Describe queue and availability algorithm

Here we give just an idea of most critical parts, we don't write complete code

## Merge requests

As we said in RASD we use merged request to manage sharing option

```
function mergeRequests(Request[] $requests)
{
    $newRequests = array();
    foreach($requests as $request){
        if($request instanceof SharedRequest)
            if($match = findRequestMacthing($newRequests, $request))
                $newRequests[] = createMerge($math, $request);
            else
                $newRequests[] = new MergedRequest($request);
        else
            $newRequests[] = $request;
    }

    return $newRequests;
}

function findRequestMacthing(Requests[] &$newRequests, Request $request)
{
    foreach($newRequests as $key=>$newRequest)
        if(($newReuqets instanceof MergedRequest || $newReuqets instanceof SharedRequest)
                && matching($newRequest, $request)){
            unset($newRequests[$key]);
            return $newRequest;
        }
}

function matching(Request $request1, Request $request2)
{
    //...
    //this is explianed in the glossary
}
```

```
function createMerge(Request $request1, Request $request2)
{
    if($request1 instanceof MergedRequest)
        return $request1->add($request2);
    else
        return new MergedRequest($request1, $request2);
}
```

### Make ride and calculate fees

```
function makeRide(Request $request, Driver[] $drivers)
{
    $ride = New Ride();
    //$ride->set... //set data like drivers
    $paths = $request->path();
    $clients = array();

    //order paths
    foreach($paths as $path)
        if($path instanceof LoadPosition)
            $clients[$path->client->id]['LoadPosition'] = $path;
        else
            $clients[$path->client->id]['DropPosition'] = $path;

    //calcualte fees
    if($request instanceof MergedRequest && count($clients)>2)
        foreach($clients as $client)
            $ride->setPrices($client->client->id, calcualteFee($client, true));
    else
        $ride->setPrices($client->clients[0]->id, calcualteFee($clients[0], false));
}

function calculateFee(Array $client, boolean $discount)
{
    return calculateDistance($client['LoadPosition'], $client['DropPosition']) *
    $request->passengers * ($discount?(1-SHARING_DISCOUT):1);
}
```

# User interface design

## Mockups

We have already done mockups in RASD in section 3.2.1 and 3.2.2

## UX diagrams

We insert UX (user experience) diagrams to show how our user performs main actions

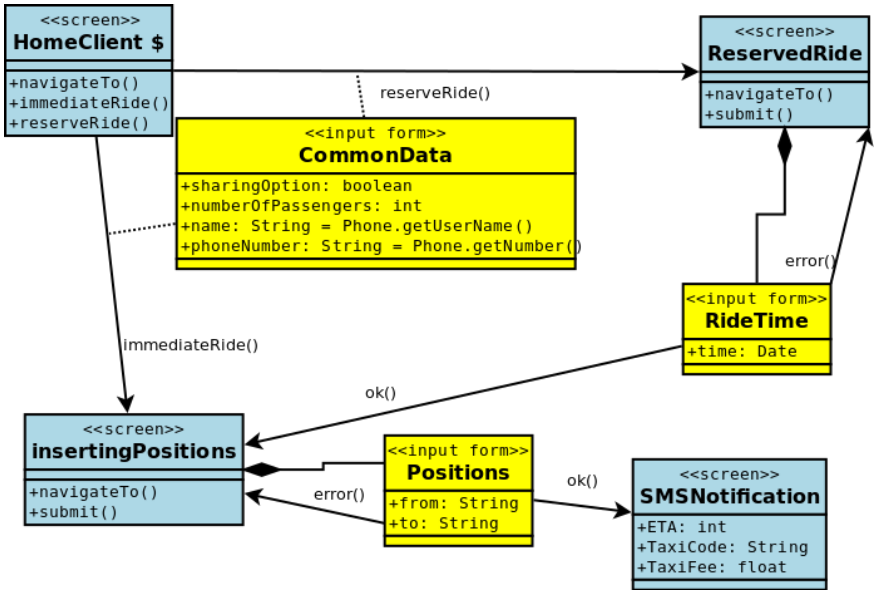


Figure 7: UX user mobile

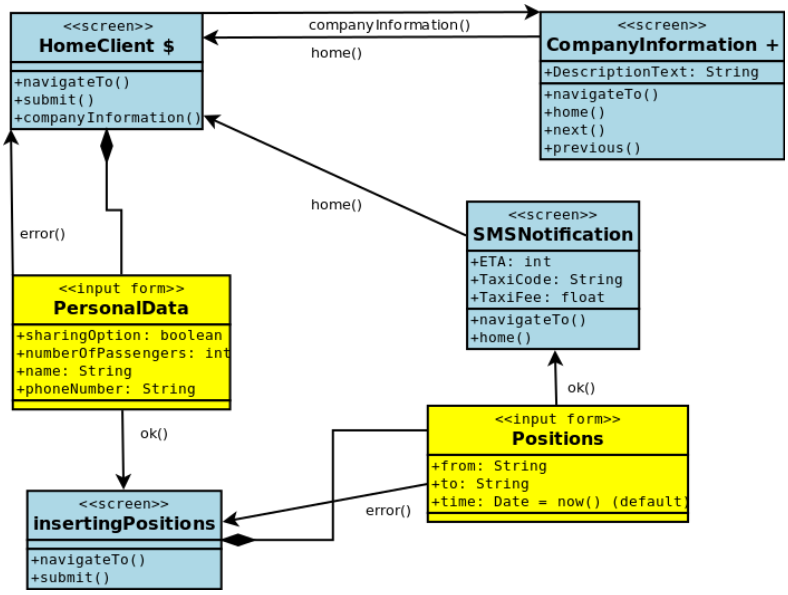


Figure 8: UX user desktop

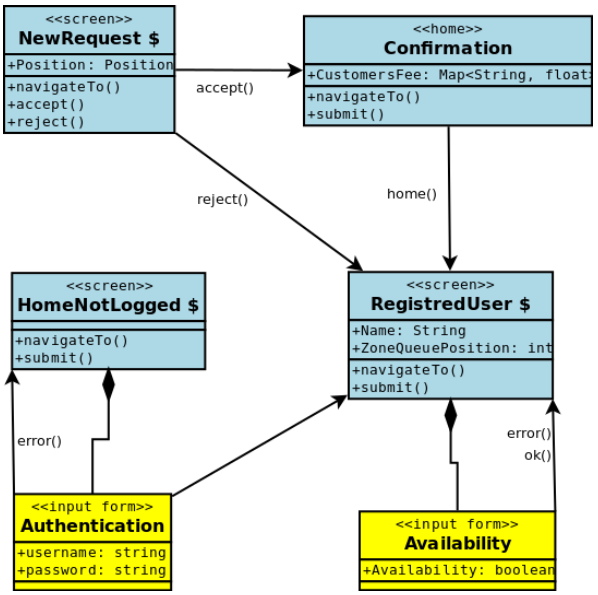


Figure 9: UX taxi driver mobile

## BCE diagrams

We insert BCE (business controller entity) diagrams to show how each user action is managed internally and how it's linked with our model. This diagram is very useful since we use MVC.

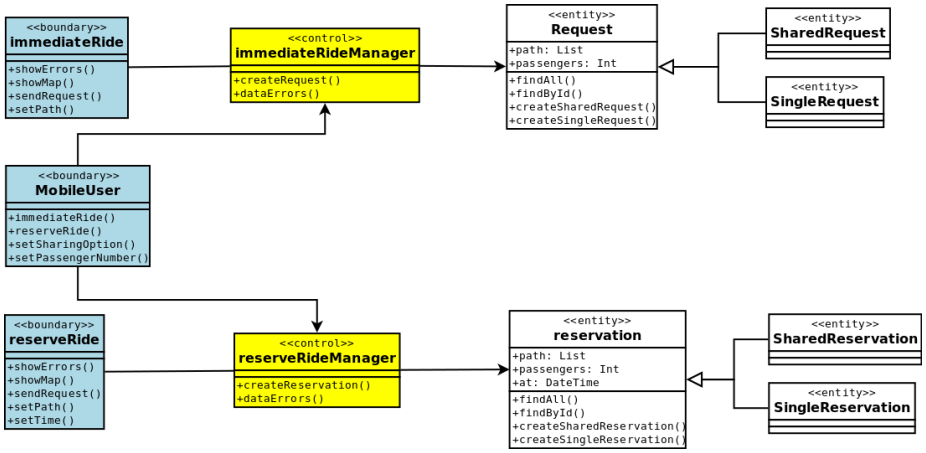


Figure 10: BCE user mobile

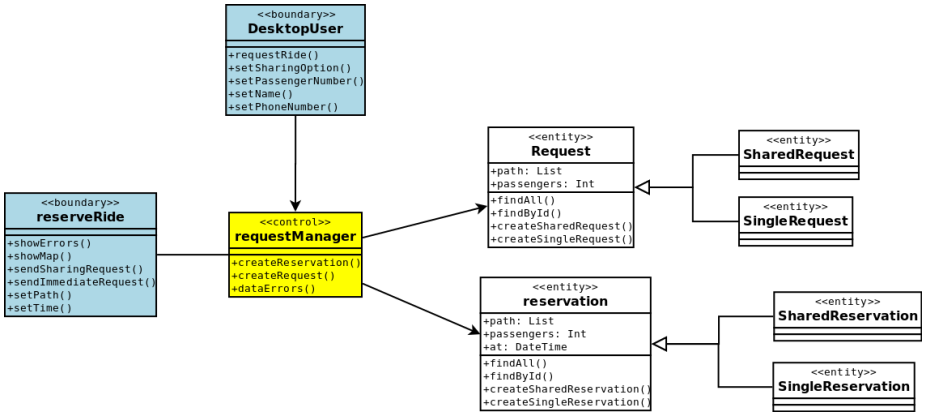


Figure 11: BCE user desktop

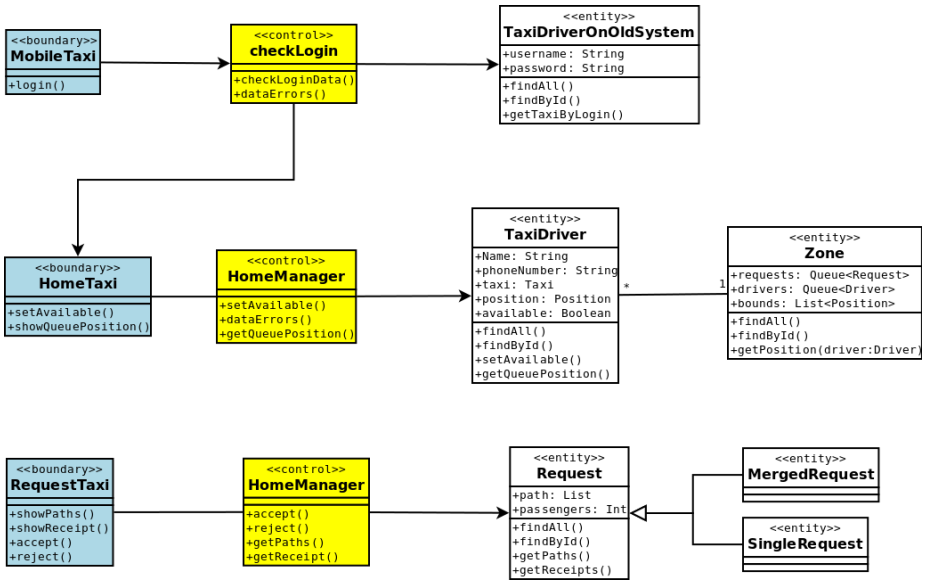


Figure 12: BCE taxi driver mobile

# Requirements traceability

# References

Maybe software used

## Hours of work

**Claudio Cardinale**

**Gilles Dejaegere**

**Massimo Dragano**