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.
- Path: it's a structure that contains at least 2 positions
- Sharing discount percentage: discount percentage applied only if the sharing option is enabled and there is more than one request in the merged request
- API: application programming interface; it is a common way to communicate with another system.
- ETA: estimated time available; it is the time the taxi needs to arrive to client starting position.
- Zone: it is a zone of approximately 2 km^2, the city is split into these zones. From taxi position the system gets his zone and inserts the taxi into the zone queue. So the system guarantees a fair management of taxi queues.
  - A zone is specified by a list of bounds.
  - It has more than 2 positions that compose its bounds.
  - Bounds must be composed by positions and not by none of its subclasses.
  - Bounds must be a set (it must not contain duplicates).

COMPLETE INSERTING OTHER GLOSSARY FROM RASD

Reference documents

- RASD produced before 1.1
- Specification Document: Assignments 1 and 2 (RASD and DD).pdf
- Structure of the design document.pdf

INSERT or not lecture slides

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.

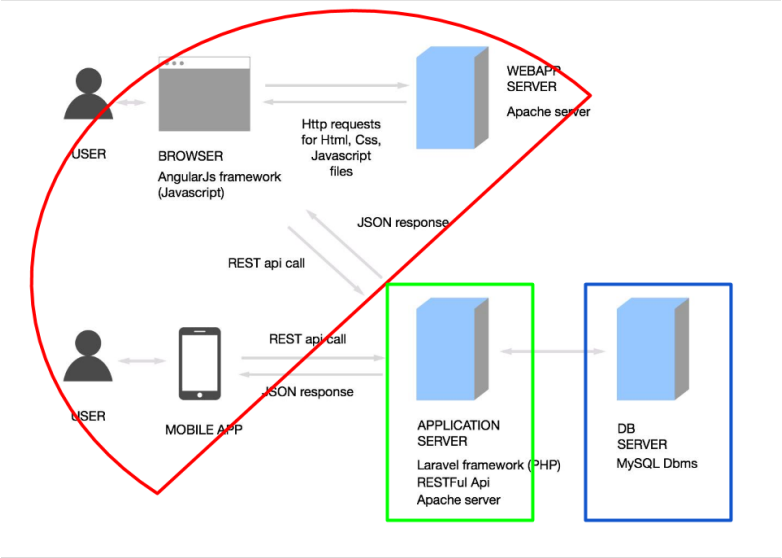


Figure 2: General architecture

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?

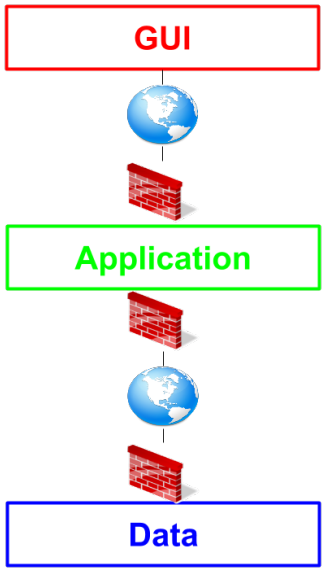


Figure 3: Tiers

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 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.

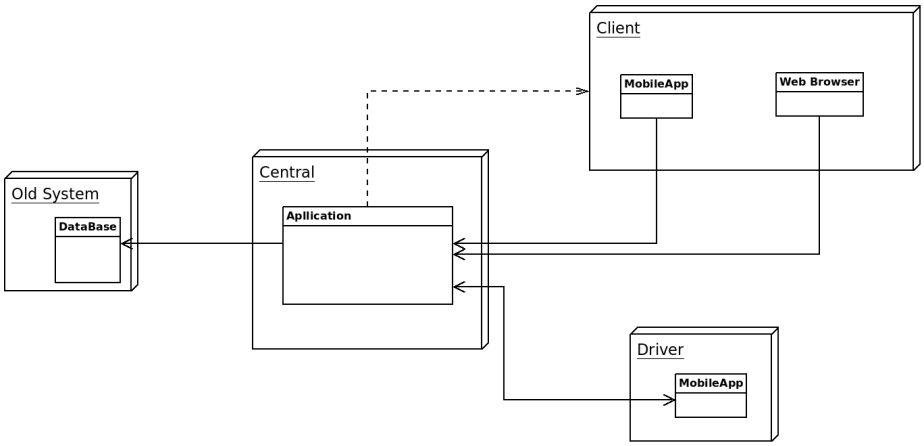


Figure 4: High level components

Component view

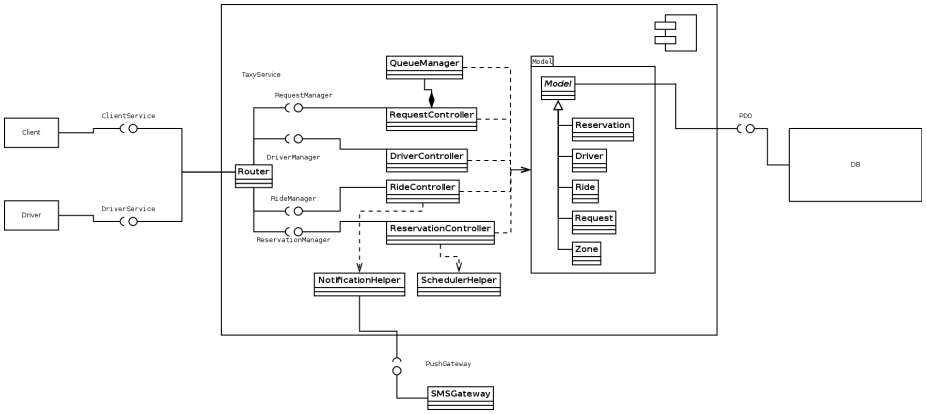


Figure 5: Component view

Deploying view

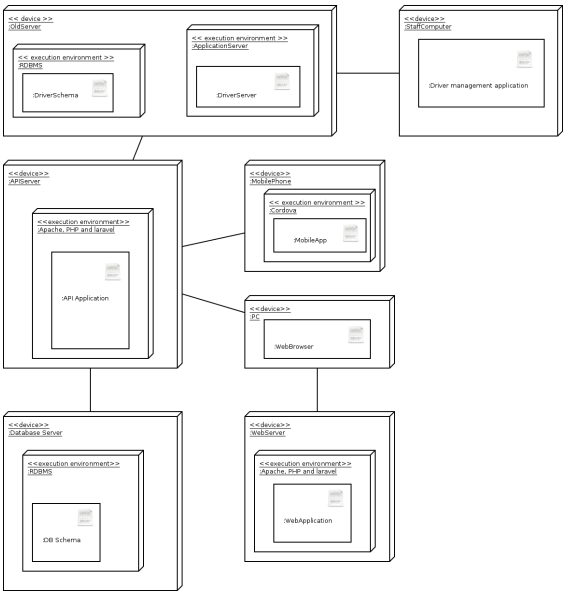
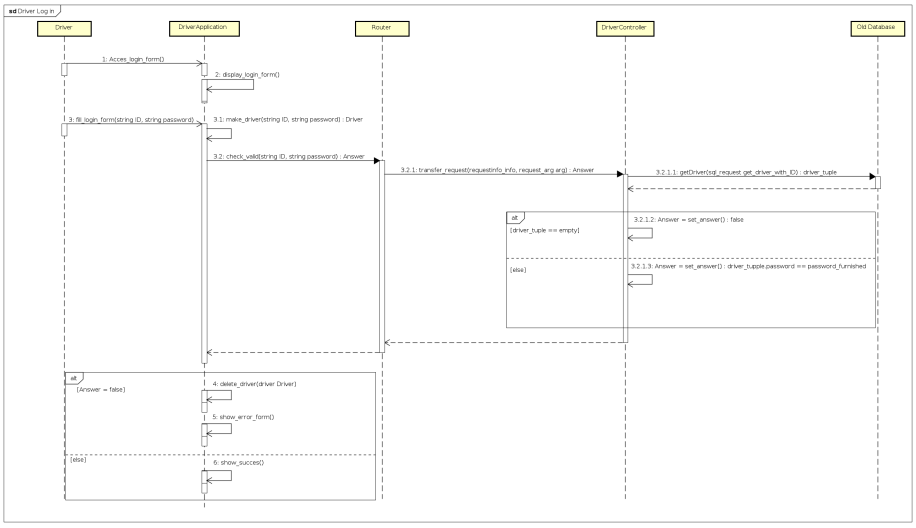


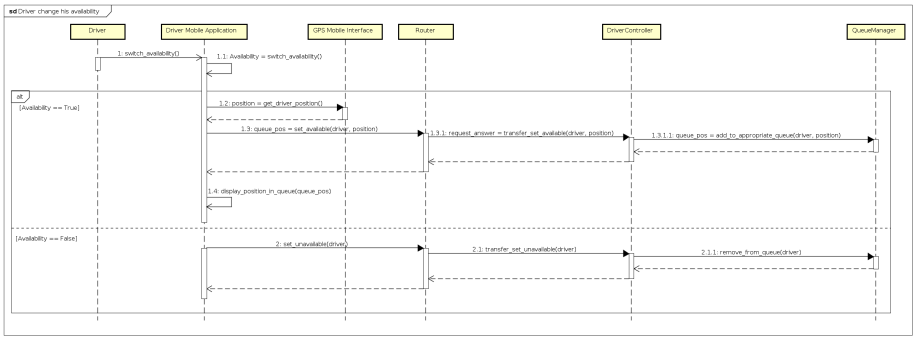
Figure 6: Deployment view

Runtime view

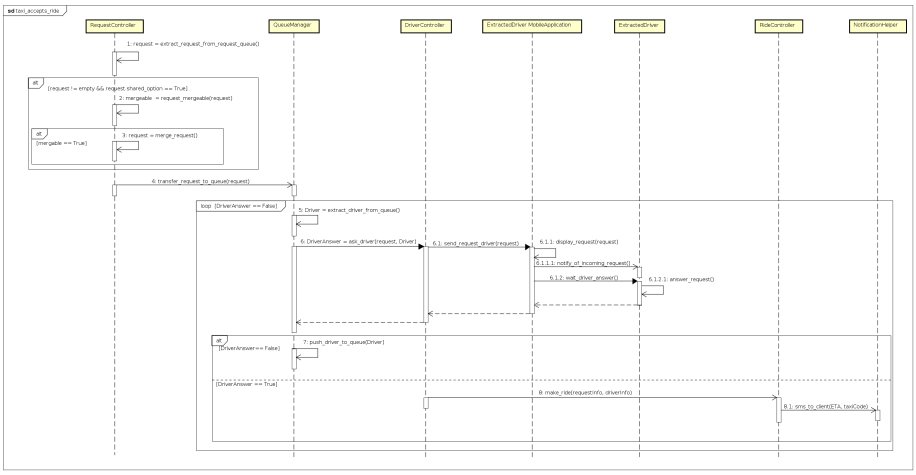


In this sequence diagram it can be seen that the user (in this case a non identified taxi driver) has to input his login informations on the taxi driver

mobile application. The login request is then send with these information as parameter to the systems. Once arrived to the system's router the request is transfered to the DriverController which first checks on the old database if the logins inserted by a user belongs to an existing driver, and if the answer is positive, if the if the password furnished is correct. The DriverController then retrns the results of these checks to the DriverApplication.

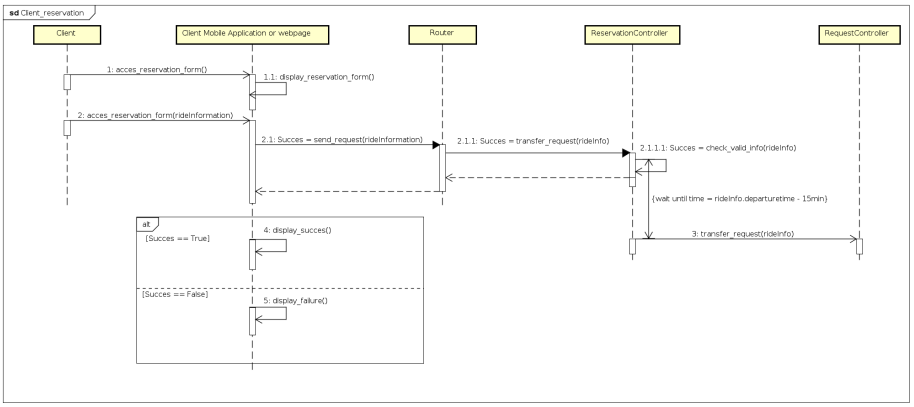


In this sequence diagram it can be seen that the when a driver changes his availability, the request is transferred to the DriverController via the Router. If the driver asks to be available, the DriverController will have to ask to the QueueManager to add the driver to the appropriate queue accorring to the driver's position. The QueueManager then returns to the position of the driver in his queue. This information goes all the way back to the driver's mobile application. In the other case, when the driver does not want to be available anymore, the DriverController has to ask to the QueueManager to remove the driver from the queue he is in.



In this sequence diagram it can be seen that when a request has to be handeled, first of all the RequestController checks if it is a request for a shared ride. If it is the case, the RequestController will check with other shared request if they can be merged together in one. After that, the request is transferred to

the QueueController which will have to asks to the appropriate driver if he wants to take care of the Ride. The Queue controller extract the driver from the appropriate queue, then asks to the DriverController to transfer the demand to the driver. If the driver has rejected the ride he is put back at the end of his queue and the new first driver is extracted and is asked if he wants to take care of the ride and so on until a driver accepts the ride. When a driver accepts a ride, he is not put back in the queue. The DriverController will then ask the RideController to make a new ride and to notify the appropriate clients via the SMSGateway.



In this sequence diagram it can be seen that a client needs to indicate the request information. Once that is done, the request can be transfered via a the Router to the ReservationController which is in fact a sort of scheduler. Once the time indicated for the reservation by the client has nearly been reached, the reservation is handeled by the RequestController like a normal request. If some information indicated by the user is not valid (like a wrong departure time for the reserved ride) the RequestController detects it and send an error back to the client mobile application or webpage.

Link seq inserted on RASD OR NOT?

Make or not slides example diagram

Check diagrams if they are correct in our previously assumptions

Component interfaces

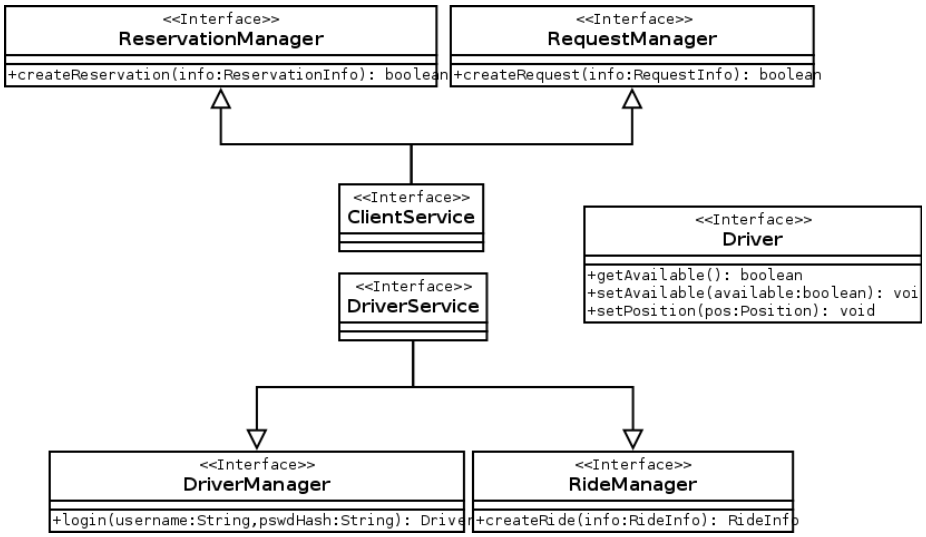


Figure 7: Component interfaces

Selected architectural styles and patterns

Overall Architecture

Our application will be divided into 3 tiers:

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

N.B. To end the ride we use the position of the driver, when the position it's nearby the final location the ride will be canceled. **Good here or not?**

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

WRITE SOMETHING?

Algorithm design

ideas:

- Describe queue and availability algorithm **MAKE OR NOT?**

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 explained in Definitions, acronyms, abbreviations
}
```

```
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

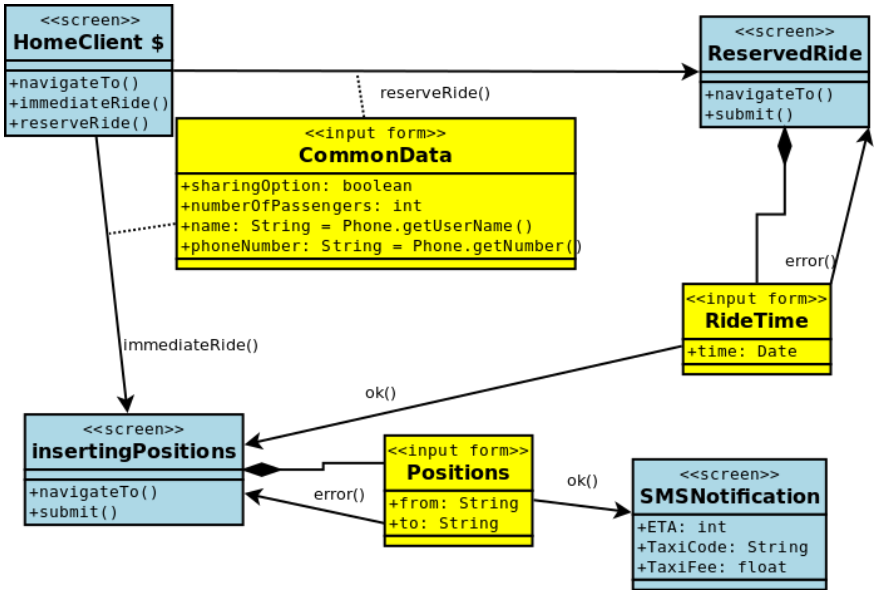


Figure 8: UX user mobile

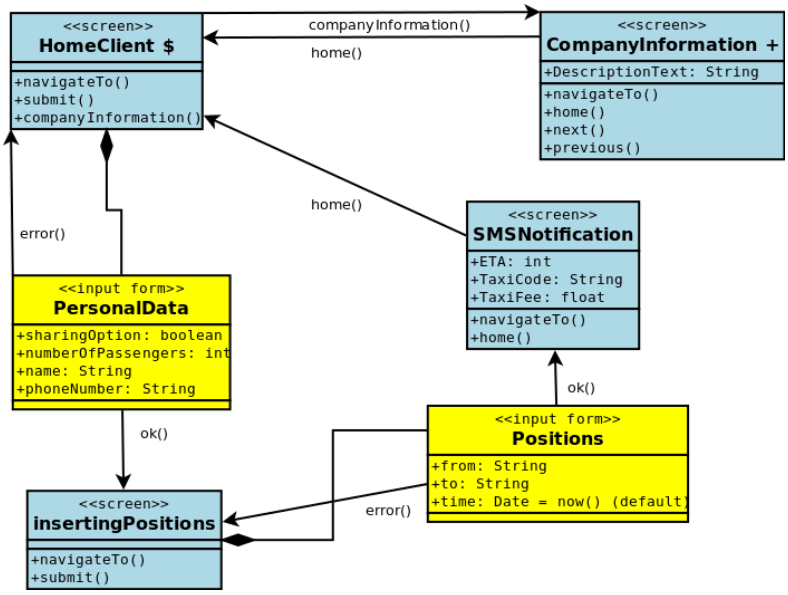


Figure 9: UX user desktop

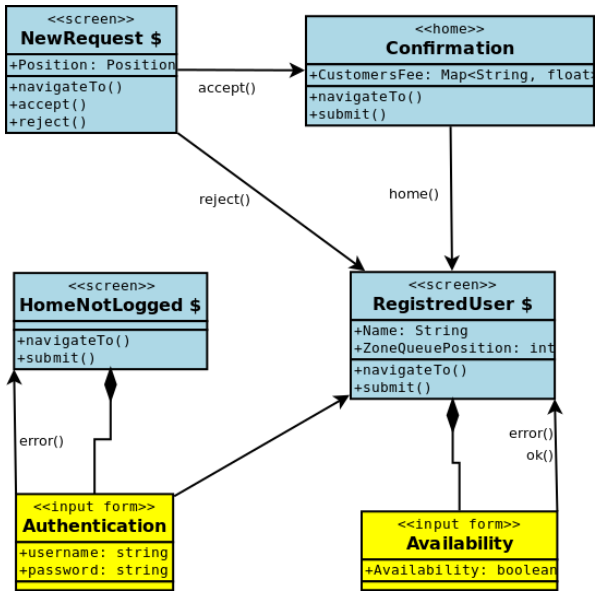


Figure 10: UX taxi driver mobile

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.

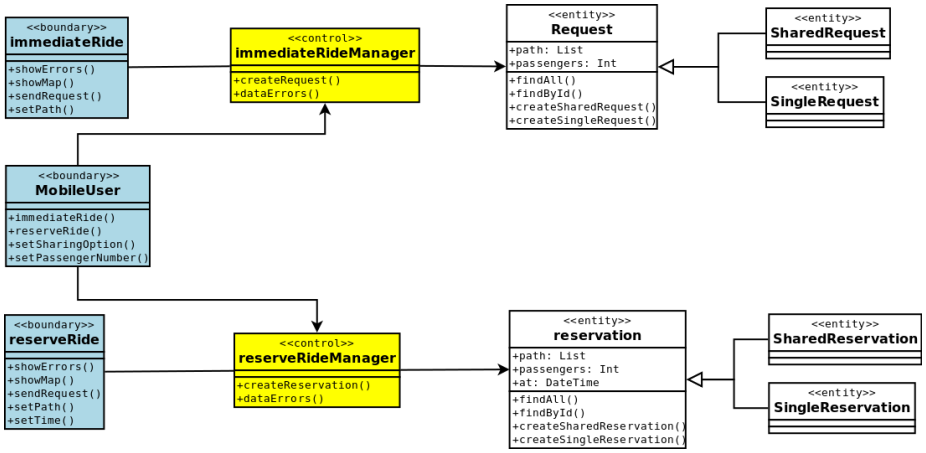


Figure 11: BCE user mobile

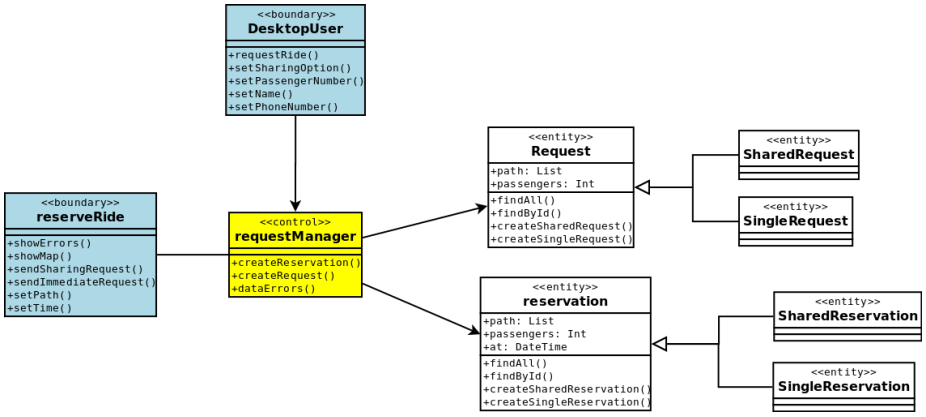


Figure 12: BCE user desktop

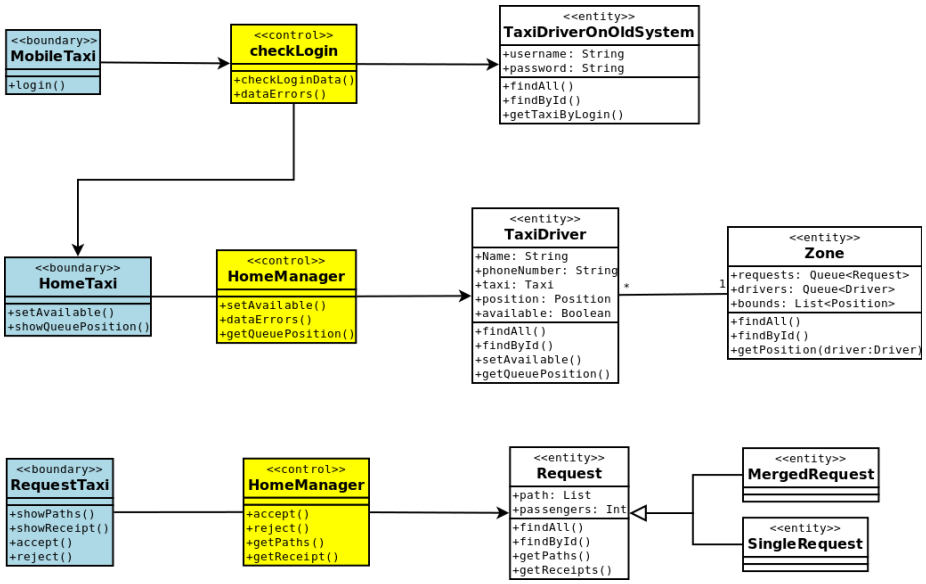


Figure 13: BCE taxi driver mobile

Requirements traceability

The design of this project was made aiming to fullfill optimally the requirements and goals specified in the RASD. The lector can find here under a the list of these requirements and goals and the designed composent of the application which will assure its fullfilment.

- [G1] Allows taxi drivers to log in the system.
  - The DriverController and its interface to the old database.
- [G2] Allows taxi drivers to precise to the system if they are available or not.
  - The DriverController
  - The Router
  - The DriverMobileApp composant
  - The QueueManager
- [G3] Taxi drivers should receive a push notification for incoming request.
  - The DriverController
  - The Router
  - The DriverMobileApp composant
- [G5] Allows taxi drivers to accept or decline incoming requests for an immediate ride.
  - The DriverMobileApp composant
  - The DriverController
  - The QueueManager
  - The RideController
  - The RequestController
- [G6] Allows taxi drivers to accept or decline incoming request for a later reservation.
  - The DriverMobileApp composant
  - The DriverController
  - The QueueManager
  - The RideController
  - The ReservationController
  - The RequestController
- [G7] Allows taxi to know the fee for each ride before it starts via the request notification (but after he has accepted).
  - The RideController
  - The DriverController
  - The Router

- [G8] Allows clients to request for an immediate taxi ride.
  - The Client composant
  - The Router
  - The RequestController
- [G9] Allows clients to request for the reservation of a taxi at least two hours in advance.
  - The Client composant
  - The Router
  - The RequestController
  - The ReservationController
- [G10] Clients should receive an SMS notification with the ETA and code of the taxi that takes care of the client's request.
  - The RideController
  - The SMSGateway
- [G11] Allows clients to require to share the taxi.
  - The Client composant
  - The RequestController
- [G12] Allows clients to identify themselves via phone number (and name) not via login, they are not registered into the system.
  - The Client composant
  - The RequestController
- [G13] Allows clients to specify the number of passengers.
  - The Client composant
  - The Router
  - The RequestController
- [G14] Allows clients to know the fee for the ride via SMS notification of taxi assigned see [G10]
  - The Client composant
  - The RideController **In G14 G10 insert interface with SMS gateway and the same for push notification?**

References

Maybe software used

Hours of work

Claudio Cardinale

Gilles Dejaegere

Massimo Dragano