Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 project: MyTaxiService

Design Document

Alessandro Pozzi (mat. 852358), Marco Romani (mat. 852361)

4 December 2015

Version 1.0

Summary

1. Introduction

1.1 Purpose

The aim of the Design Document (DD) is to define guidelines that describes the architecture of a software project. Here the requirements specified in the Requirements Analysis and Specification Document (RASD) will be translated into a representation of software components, interfaces and other data necessary for the implementation phase. It is important to underline that this document will be the main reference during the code development. For this reason it won't contain any detailed and extended code section (which would be an unnecessary constraint on programmers), but, instead, a set recommended design decision will be presented and theirs structures and advantages will be diffusely explained.

1.2 Scope

As specified in the previous paragraph, in this document we will focus on the overall structure and architecture of the system, without going deeply into the details of the implementation. Only a small section of this document, in fact, will be dedicated to some guidelines for the implementations of the application's main algorithms.

Components, connectors, interfaces are instead the main participants of this document, but their scopes and interactions will be described only at a high-level. We will also explain the architecture styles adopted and the reasons behind them, trying to give a motivation for every choice taken.

It is important to understand that all the content of the document is platform independent and the various architecture components will be mapped onto real hardware and software components only further in the implementation phase.

Since we have already provided a bunch of mockups for the graphical user interface in the RASD, we will only redirect you to them without showing them again.

1.3 Definitions, Acronyms and Abbrevations

1.3.1 Definitions

Taxi Driver – Employee of the taxi service with a driver account.

Customer –Registered user that may demand a taxi ride

Guest – Users that are accessing to MTS's homepage (or other free services) not yet registered or not logged in

1.3.2 Acronyms

MTS – MyTaxiService

GUI – Graphical User Interface

DB – Database

RASD – Requirements Analysis and Specification Document

1.4 Reference Documents

- MyTaxiService's RASD (by Alessandro Pozzi and Marco Romani)
- The IEEE standard 1016: Software Design Specification

1.5 Document Structure

- *Section 1 – Introduction:* Introduce this document in relation to the MTS's system.
- *Section 2 – Architectural Design*: explains in details the architecture and design of MTS's system, along with the chosen patterns and components identification.
- *Section 3 – Algorithm Design:* shows a possible high-level implementation of some relevant application algorithms.
- *Section 4 – User Interface Design*: shows indicatively how the user interface will look like.
- *Section 5 – Requirements Traceability:* Shows how the requirements specified in the RASD have been satisfied in the design phase.
- *Section 6 – References:* Hours of works, software and tools used and others external references.

2. Architectural Design

2.1 Overview

MyTaxiService involves different users communicating over the internet with a single system. Such users may use different platforms (mobile and web) and can send requests of different types. The system must not only accept those requests and elaborate an answer in a short time, but it is required that it notifies multiple users of the occurring of some events. Usually a single event provide notifications for two types of users: taxi drivers and customers. Event notifications and users requests might also necessitate to access stored data, like taxi identifier or users information.

This brief analysis clearly highlight the need of implementing MTS as a client-server-like architecture, eventually subdivided into multiple physical tiers and logical layers: this will allow to model properly the request-answer requirement. The notification and updates part, instead, requires in our opinion an event-based paradigm: in particular, the *publish-subscribe* pattern will be intensively used: it will allows users (the subscribers) to be notified by an entity (the publisher) on specific topics (a ride, for example).

These styles will be explained in detail in the following chapter.

2.2 Selected Architectural styles and patterns

2.2.1 Three-Tier Architecture

The image shows the tier architecture of the MTS system, composed by three physical tiers. We will now analyze every tier and explain its logical functions.

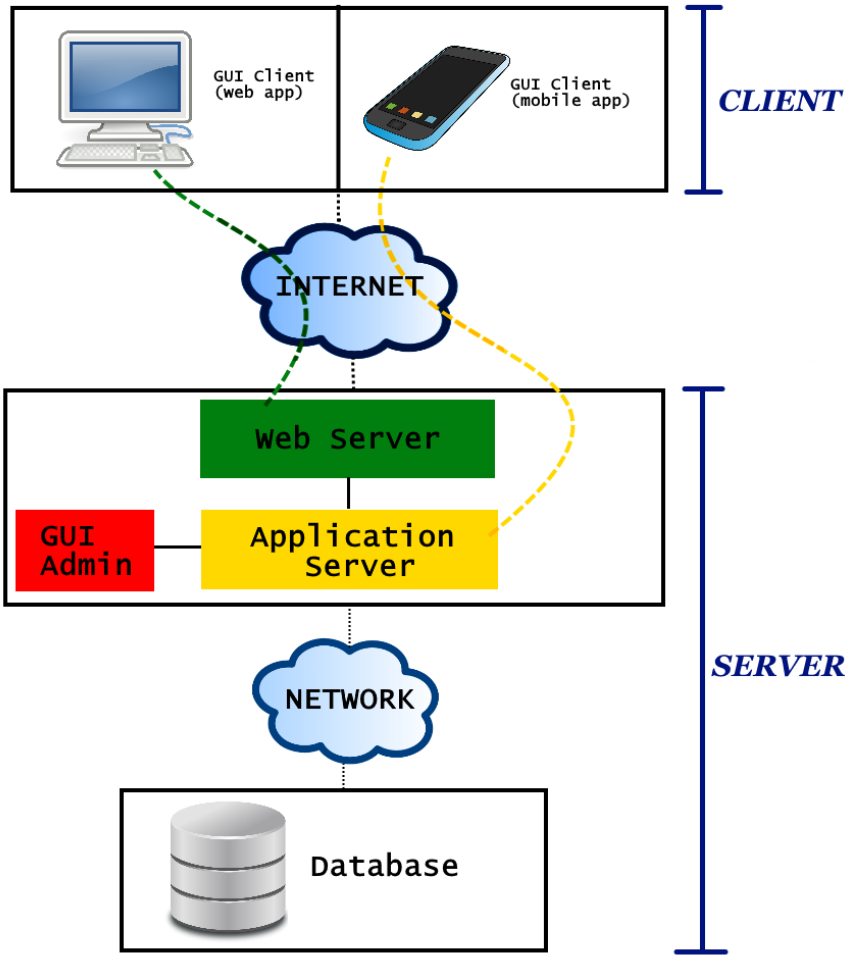
- *Top tier (Client)*

  The users' machines, that in our domain are mobile phones and computers, will have the only purpose to load the Graphical User Interface (GUI), which shows the services that can be requested from the MTS's system. No application logic is involved at this level: Clients will only be able to send requests to the web server and application server.
  Notice that users identified as clients are limited to the followings: Taxi Drivers, Customers, Guest.

- *Middle tier*

  This tier encapsulate:
  - The Web Server, which is the component of the system that manages the web requests sent by clients using the web application.
    This component can handle such requests in two ways:
    - if the request can be resolved with a static content page, the web server will generate and send the response itself

- if the request comport a dynamic content, the web server will delegate the dynamic response generation to the application server



- The Application Server, which provides access to the business logic, to be used by the client application programs. This component is the central part of MTS's system, and will contain all the logic that provides MTS's services. To accomplish this, it will be able to execute complex algorithms and access the Database tier.
  The Application Server will also provide lightweight APIs to be used directly by mobile application clients. It will answer mobile's requests by sending only the strictly necessary information, reducing the amount of data transiting over the mobile network and thus increasing the performance of the application.
  Web application clients, instead, will be able to access this component only indirectly, through the Web Server.
- The Admin's GUI, the specific interface for Administrators, is actually included and provided by the Application Server. It allows Admins to access to their exclusive functions dialoguing directly with the business logic of the system. It's completely disjointed from the other users' GUI and functions.

- *Bottom tier (Database)*

  This tier, which will be separated from the previous one with a (possibly local) network, contains all the data that MTS needs to store, ranging from the users' information to the city map.

2.2.2 Even-based architecture

As anticipated, the core of MTS's application logic is event-based. Every significant change of state (for example users modifying theirs account information, drivers changing status, etc.) generates an event that the system can handle accordingly. In particular, the managing of the customer's requests and taxi rides will be modeled with the publisher-subscribe pattern. The usage of such pattern is described below.

- *Publish-subscribe*

  Customers and taxi drivers have the role of subscriber: the system will automatically register them to specific topics, and they will receive update messages related to such topics. A topic is created every time customers request or reserve a ride. When a taxi driver is associated to that ride, he will be subscribed to the same topic too, and receive the relative notifications.

  In this pattern we intend to use a broker, an intermediary component which performs the queue management and the filtering of the messages. The broker will allow to filter messages based on their content, so that taxi drivers and customers related to the same topic won't receive necessary the same notification or messages.

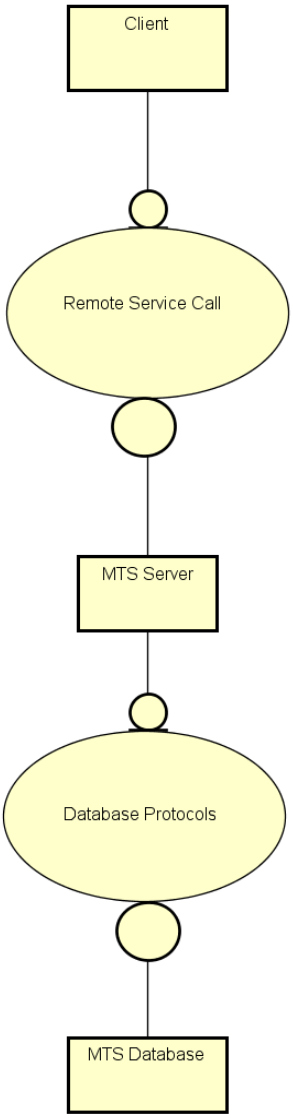
  The role of the publisher, instead, is associated to the logic components of the system that manages the rides, the research of available taxis, etc. Basically, there are more components of the system that may generate an update for a certain topic.

Despite the event-based pattern may not be strictly required to model the actual MTS system, it provides much space for future extendibility. For example, it allows to handle notification for multiple customer connected to the same ride (e.g. in a taxi sharing service), or permits to easily add new types of notifications.

2.3 High level components and their interaction

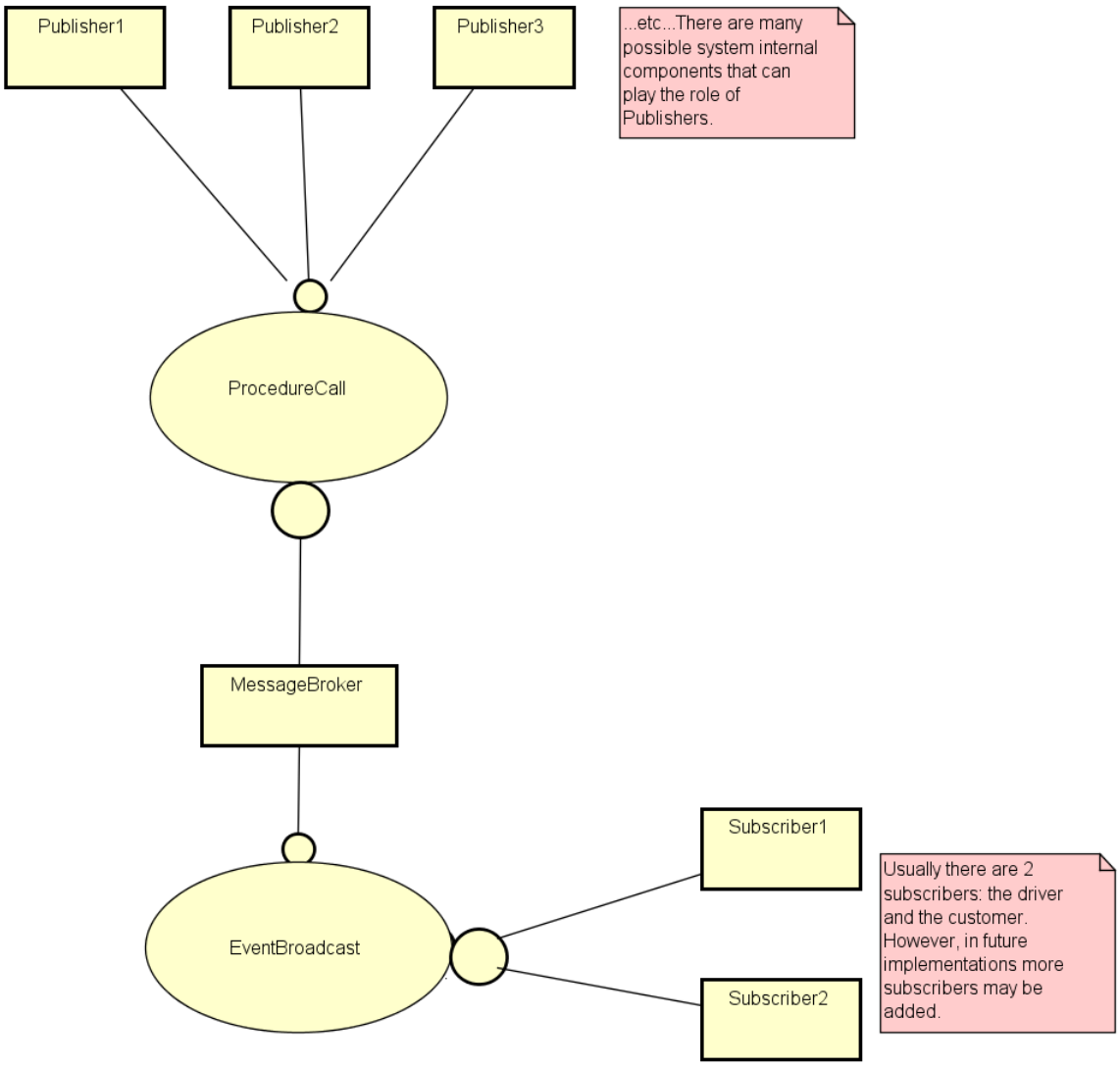
2.3.1 Client-Server-Database

Shows a high-level view of the three main components of the system.



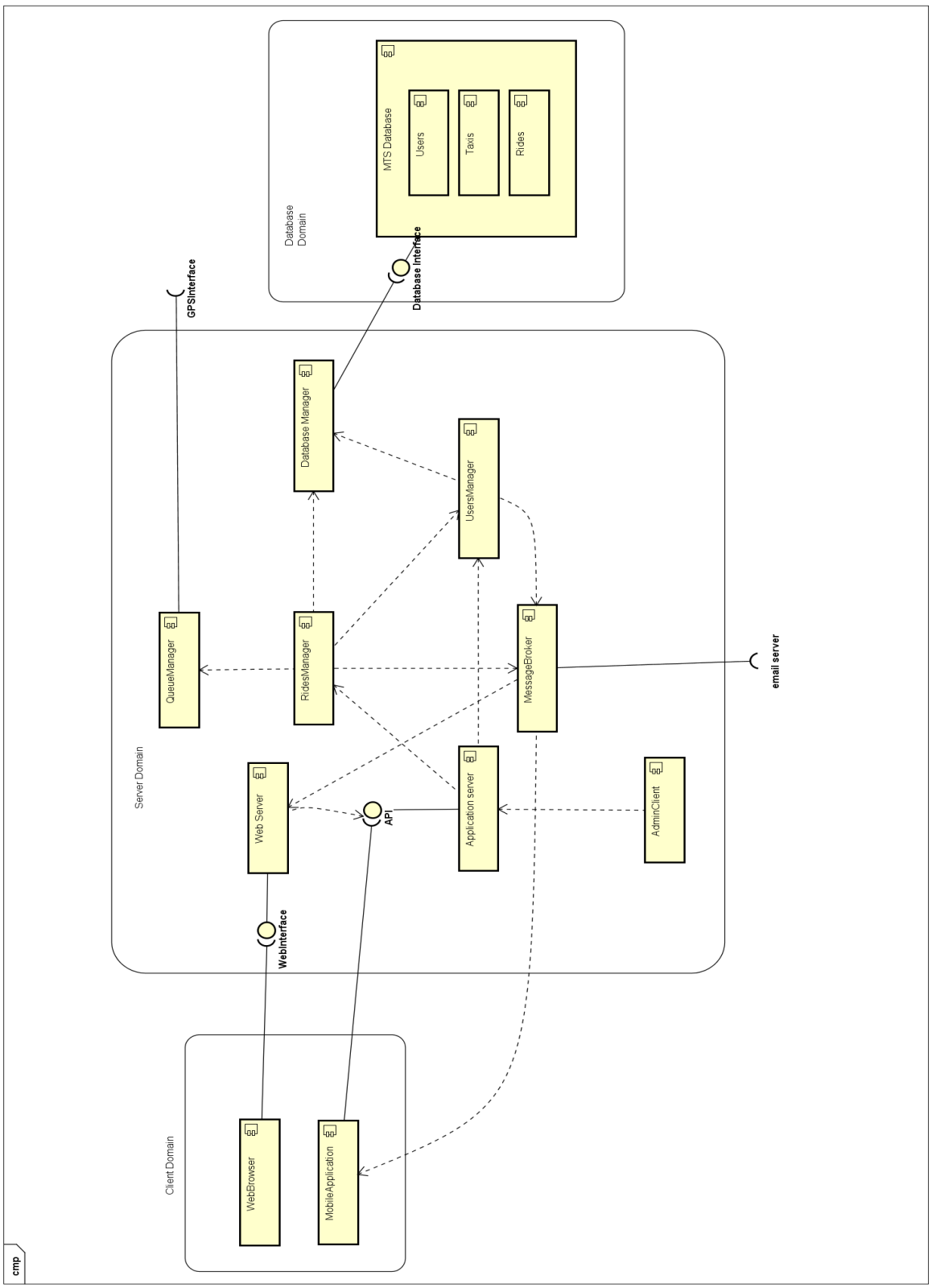
2.3.2 Publish-Subscribe

Shows a high-level view of the logical components used in the Publish-Subscribe pattern.



2.4 Components View

Component Diagram: this diagram describes the logical components that constitutes the physical tiers previously described.



- *Web Server*

This component represents the front-end of the system that interacts with the web application's users. Can answer their requests with a static content page and dialog, if needed, with the *Application Server*.

- *Application Server*

Represents the front-end of the system that interacts with mobile application's users and the Admin's GUI. Can dialog with other internal logic components.

- *Database Manager*

Handles the only access point of the middle tier to the DB tier.

- *Rides Manager*

Handles requests and reservations and, in general, everything that is connected with rides. It is a central part of the application, and is the only component that interacts with the *Queue Manager*.

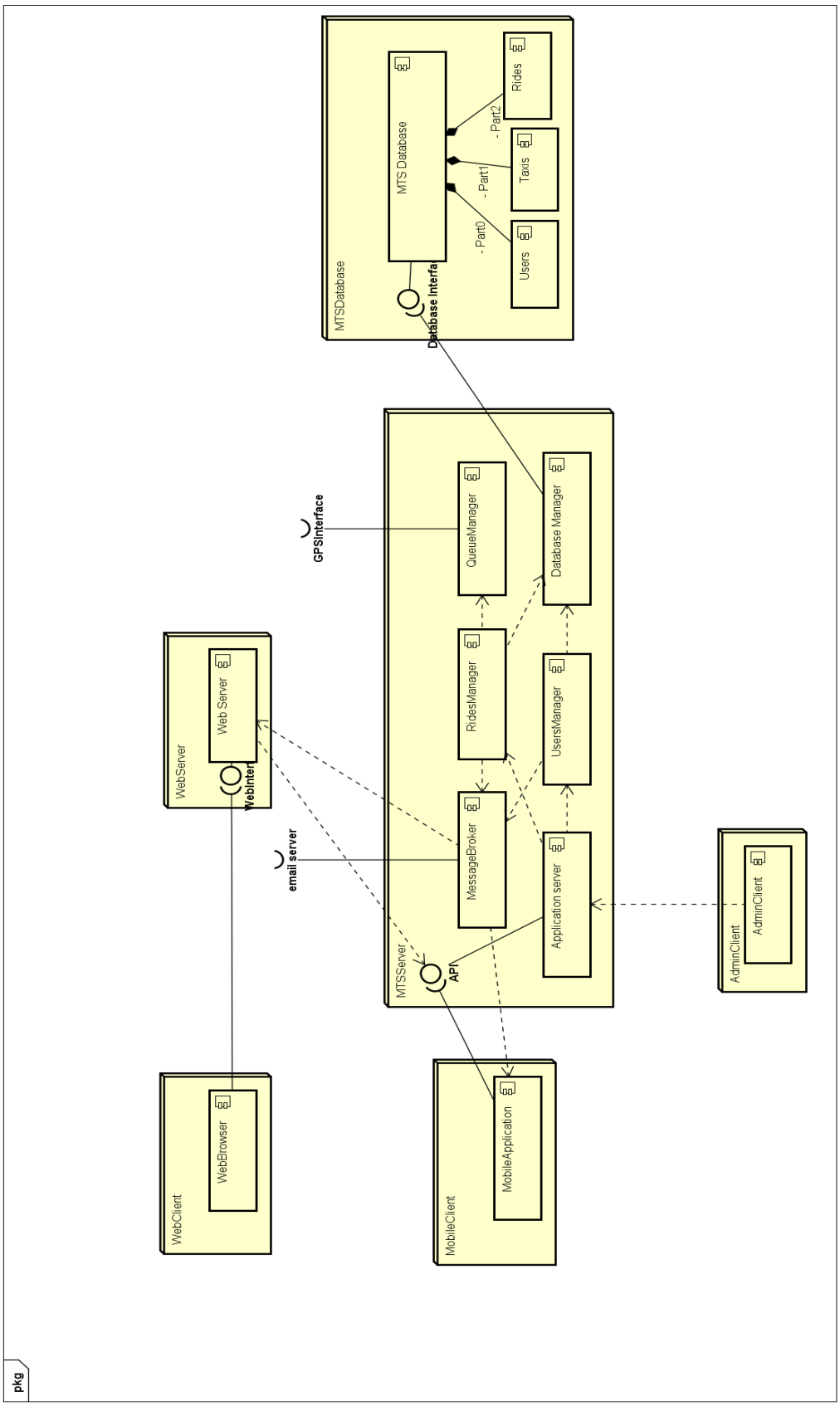
- *Queue Manager*

Manages the taxi zones and queues logic. It memorize (but does not store in the DB) the position of every active taxis received by the *GPS Interface*.

- *Message Broker*

Implements the Broker role in the publisher-subscribe pattern. Receives the publication messages from the *Rides Manager* and *Users Manager* and forward the appropriate communications to the clients.

2.5 Deployment view



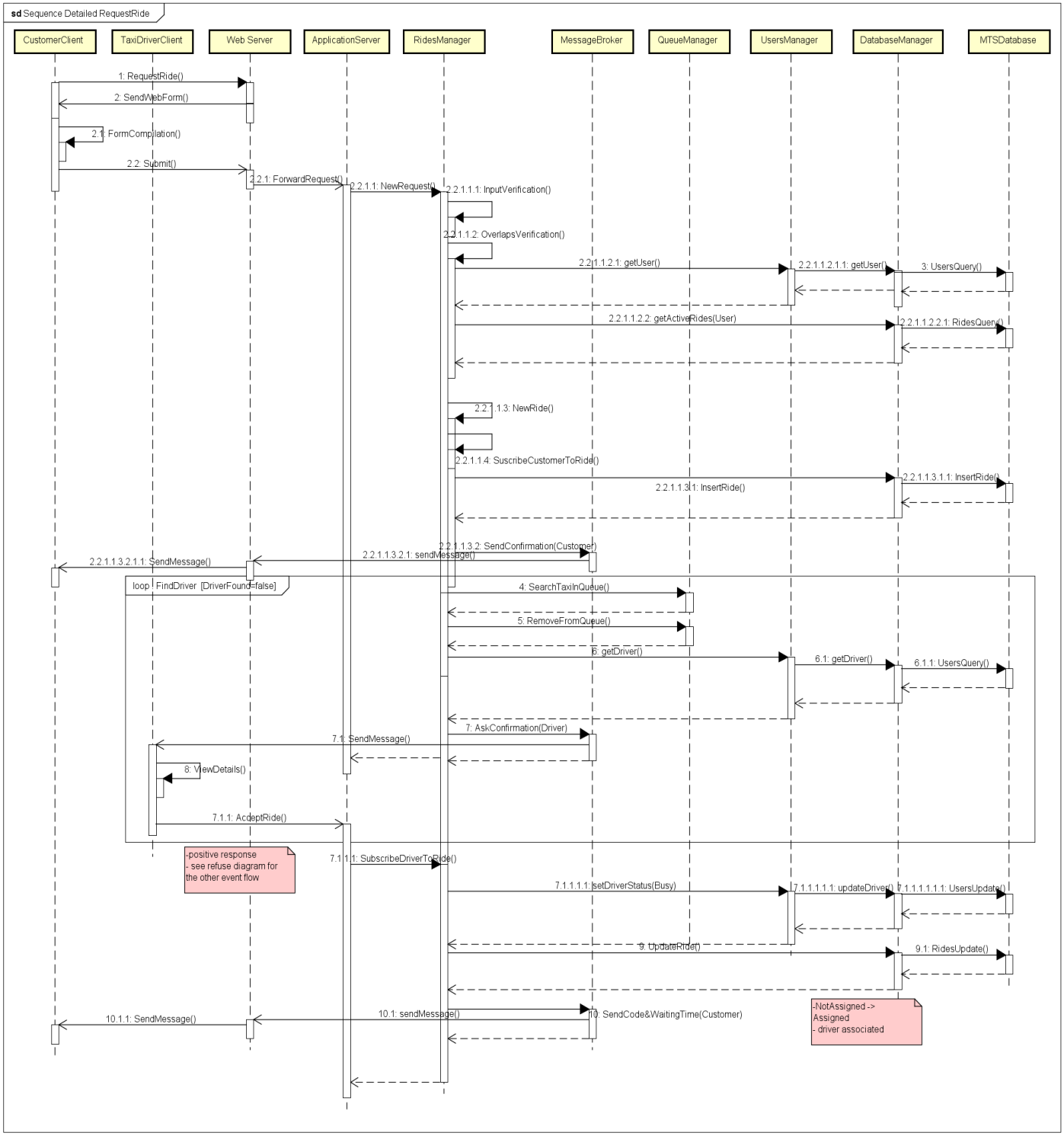
We organized the system's components in several different physical nodes.

These are:

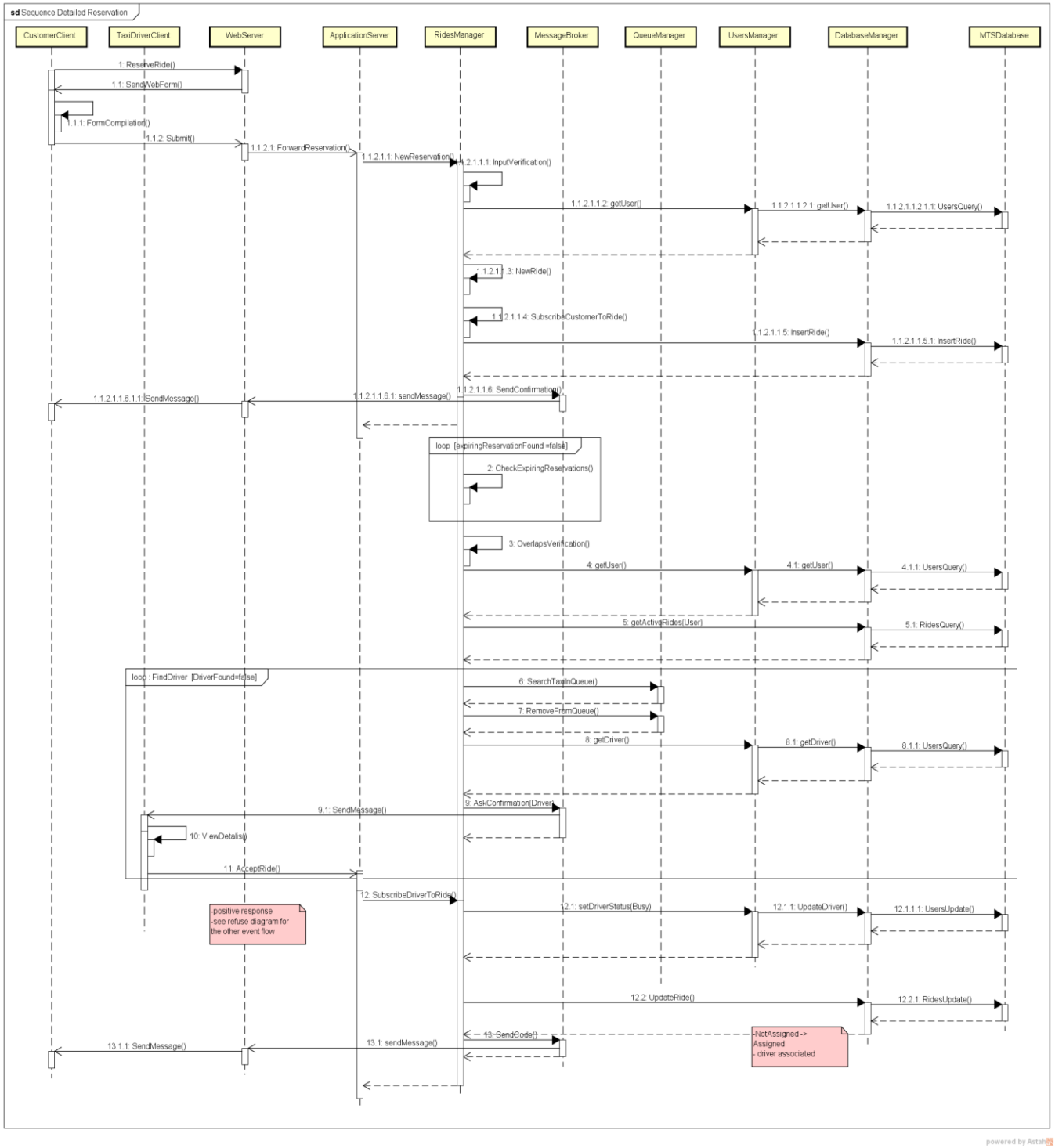
- *WebClient* -> Physical machine that runs the client's browser. It could be for example the personal computer of a customer. This node is separated from the system by the Internet network.
- *MobileClient* -> Physical mobile device that runs the mobile version of the application, previously installed. It could be for example the smartphone of a customer. This node is separated from the system by the Internet network.
- *WebServer* -> This node contains the web server component, which is not located on the same machine of the actual MTS Server. However, our idea is that they should be located within the same private network, even if this is not mandatory.
- *MTSServer* -> This node contains all the business logic of the application and is the central part of the whole system. It is separated from the front end users by the Internet network and it's the only node that is connected with the persistent data in the database.
- *AdminClient* -> Physical machine used by the administrators in order to access to their reserved functionalities. We think that it should be better to have this machines (there could be more than one) located within the same private network of the MTS Server to achieve better security.
- *MTSDatabase* -> Physical database in which all the persistent data of the application are stored. Also in this case we think that it should be better to have the database in the same private network of the other "server nodes", both for security and performance concerns.

2.6 Runtime view

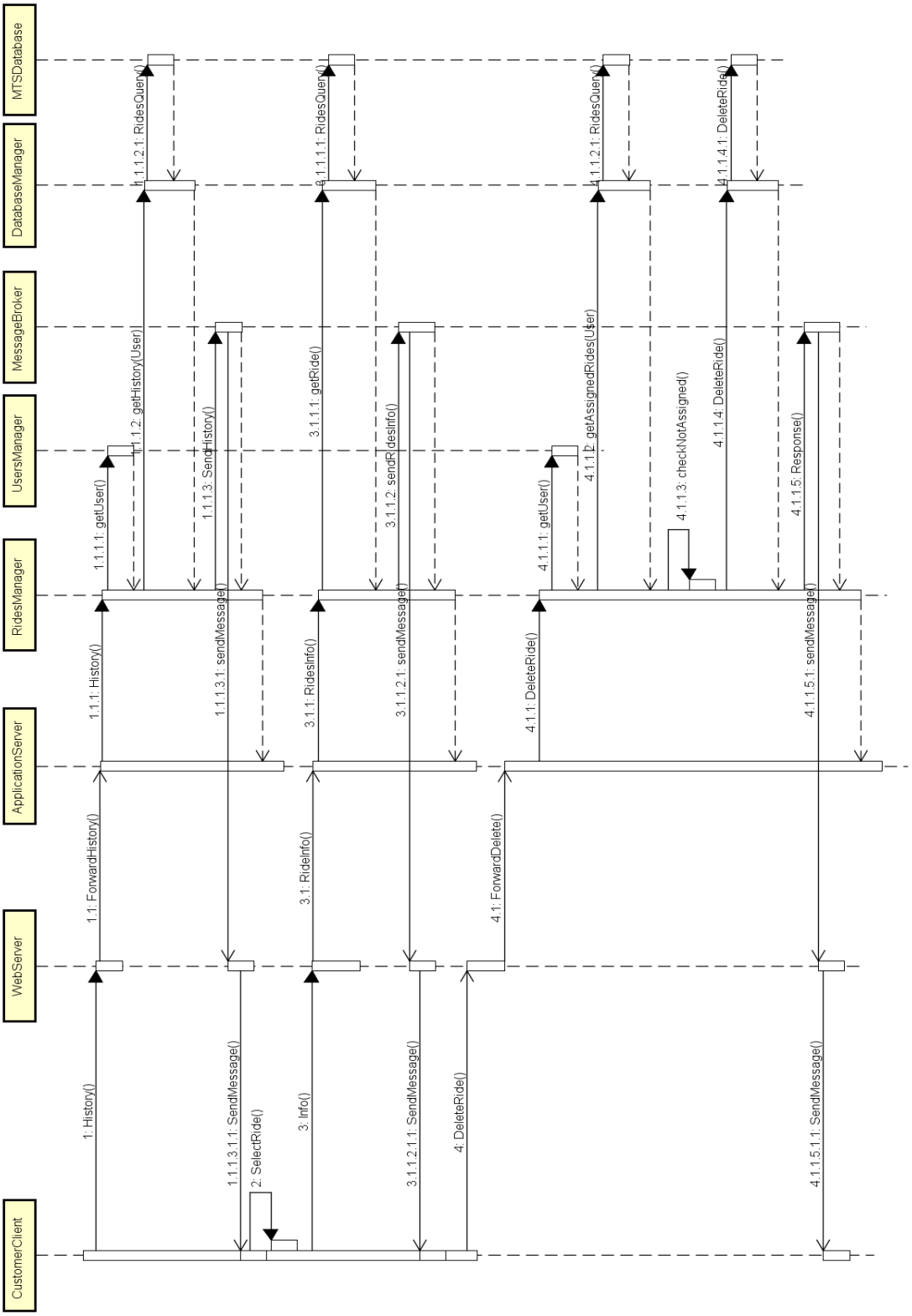
2.6.1 Request ride



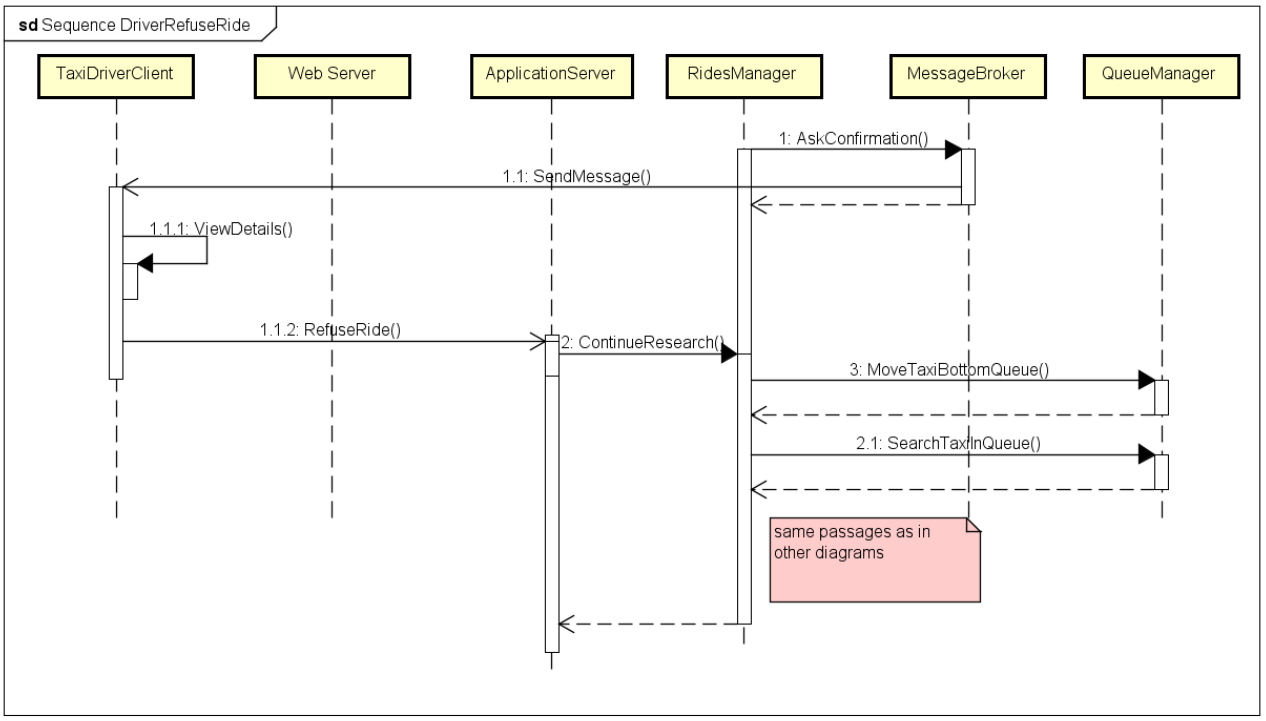
2.6.2 Reserve a ride



2.6.3 Customer deletes a ride

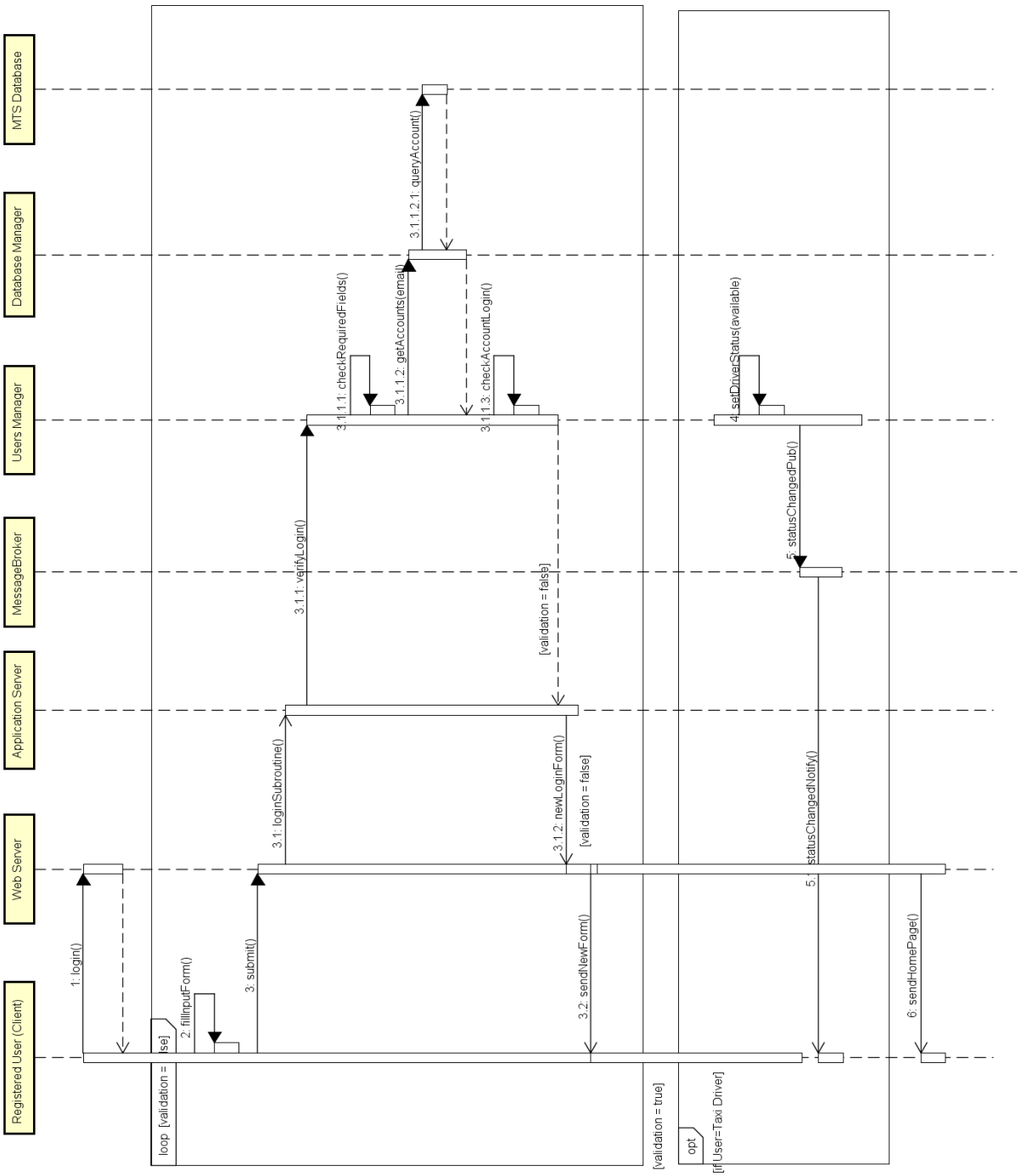


2.6.4 Driver refuses a request

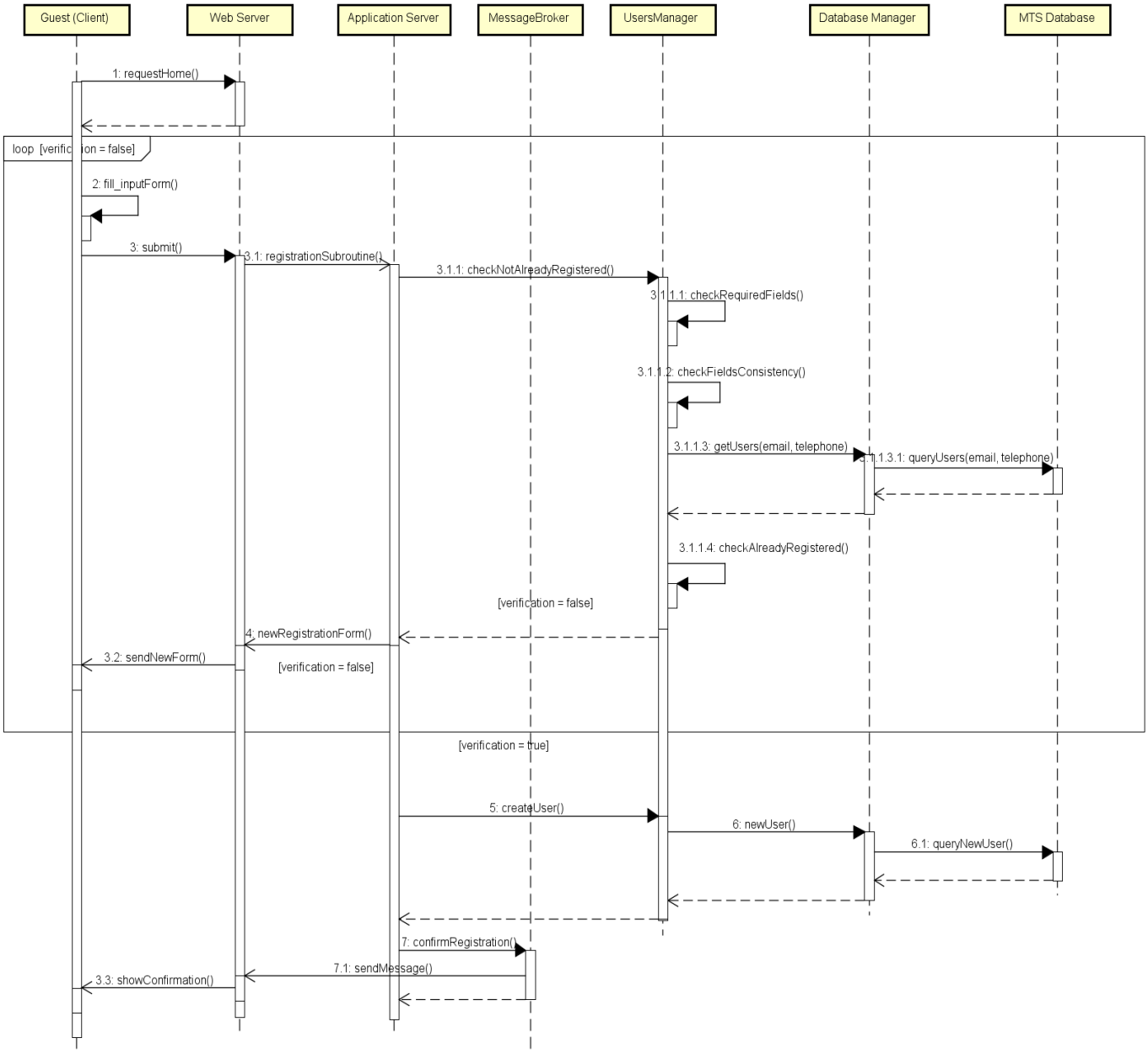


powered by Astah

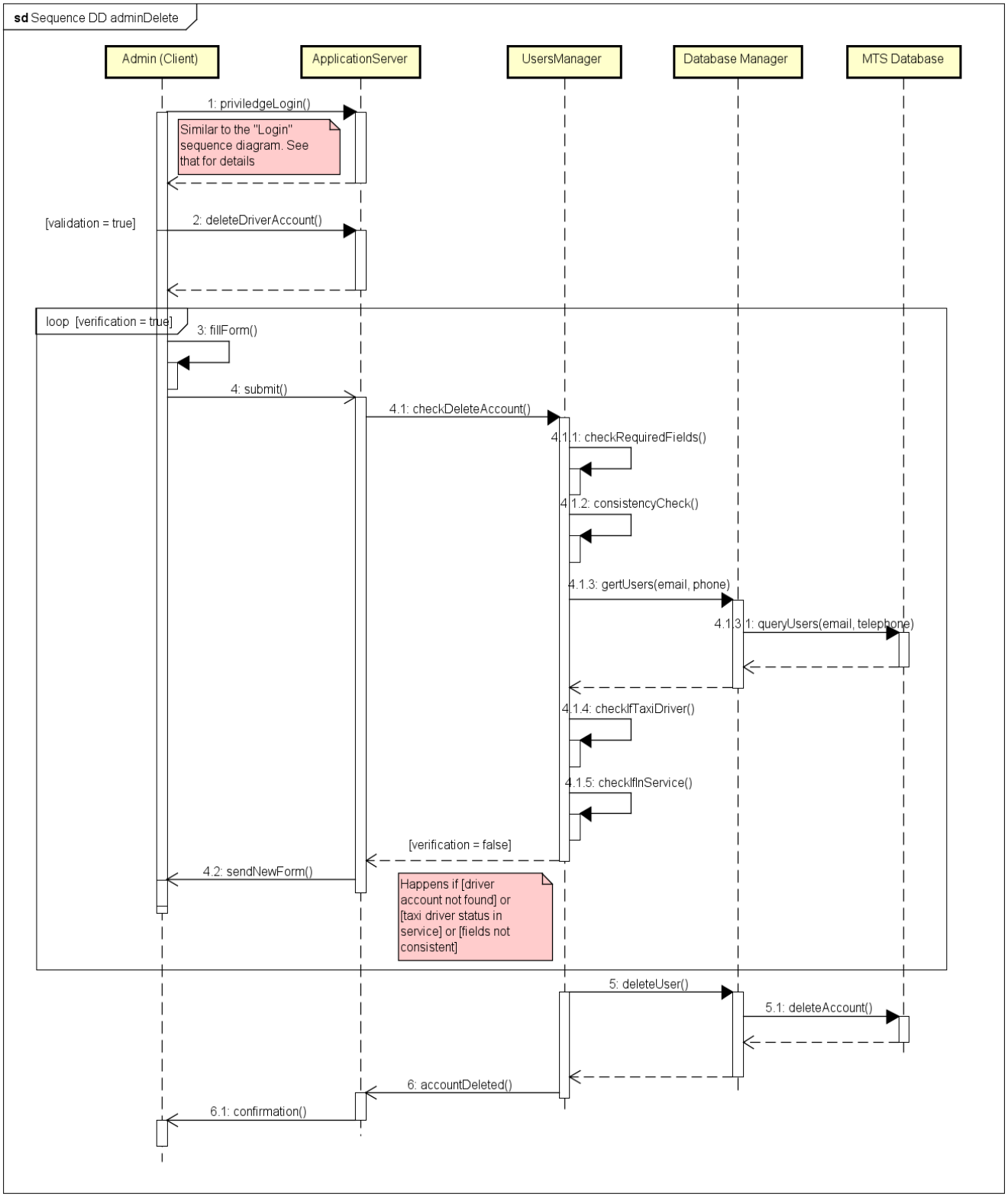
2.6.5 Login


MTS Database
Database Manager
Users Manager
MessageBroker
Application Server
Web Server
Registered User (Client)
1: login()
2: fillInputForm()
3: submit()
3.1: loginSubroutine()
3.1.1: verifyLogin()
3.1.1.1: checkRequiredFields()
3.1.1.2: getAccounts(email)
3.1.1.2.1: queryAccount()
3.1.1.3: checkAccountLogin()
[validation = false]
3.1.2: newLoginForm()
[validation = false]
3.2: sendNewForm()
loop [validation =
se]
[validation = true]
4: setDriverStatus(available)
5: statusChangedPub()
5: statusChangedNotify()
6: sendHomePage()
opt
[if User=Taxi Driver]

2.6.6 Registration

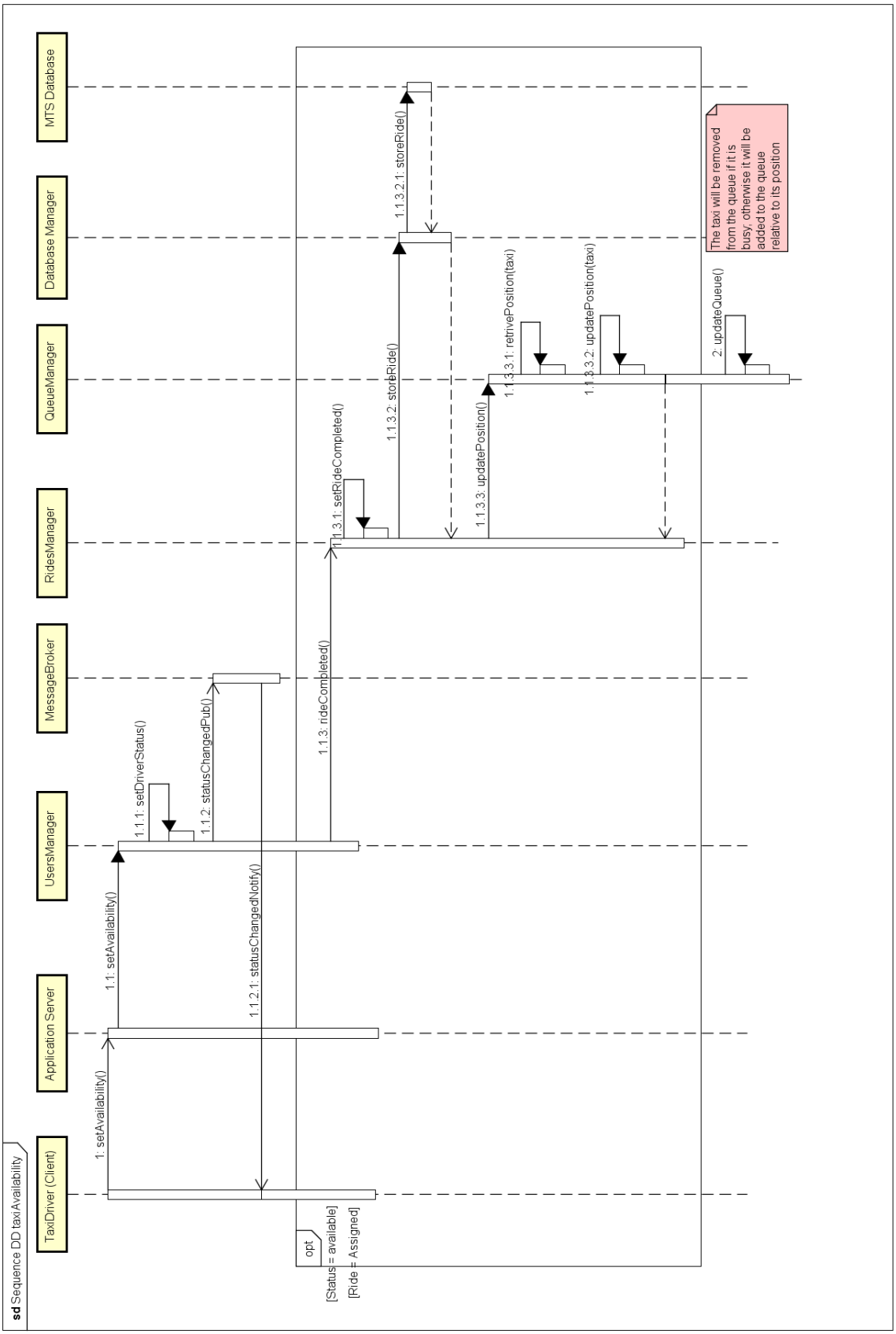


2.6.7 Admin deletes a taxi driver account



powered by Astah

2.6.8 Taxi driver changes his availability



2.7 Components Interfaces

For each component defined in the previous diagrams we summarize which kind of interfaces link it to the rest of the system, specifying the nature of those interfaces.

- ***Web Server***

  Type of interfaces provided: exchange of web pages via HTTP protocol.
  Type of interfaces required: remote procedure call to the system.
  Each service offered to web clients is mapped a customer service provided by the Application Server.

- ***Application Server***

  Type of interfaces provided: remote procedure call.
  Type of interfaces required: procedure call.

  Summary of the main interfaces offered:

  - Interfaces for registration of customers and login of registered users.
  - Interfaces for operations concerning taxi rides, such as taxi requests, reservations and rides' deletion.
  - Interfaces for access to the rides' history of a customer, with the possibility of checking info related to past rides.
  - Interfaces for operations reserved to taxi drivers, such as the communication of availability and the communication of acceptance/refusal of a ride.
  - Interfaces for operations reserved to administrators, such as the possibility of creating/deleting drivers' accounts, the possibility of modifying drivers' status or modifying rides' status.

  Summary of the main interfaces required:

  - Interfaces for the creation, deletion and management of taxi rides, and also for the associations between customers, drivers and rides.
  - Interfaces for the creation and management of users, as well as for validation of accesses.

- ***RidesManager***

  Type of interfaces provided: procedure call.
  Type of interfaces required: procedure call.

Summary of the main interfaces offered:

- Interfaces for the creation of requests and reservations and the deletion of rides.
- Interfaces for the termination of rides.
- Interfaces for the subscription of customers and drivers to rides.
- Interfaces for the retrieval of the rides history of a customer and info about specific rides.

Summary of the main interfaces required:

- Interfaces for taxi queue management.
- Interfaces for the management of users.
- Interfaces for the communications of messages/notifications.
- Interfaces for the access to MTS database.

- ***UsersManager***

Type of interfaces provided: procedure call.
Type of interfaces required: procedure call.

Summary of the main interfaces offered:

- Interfaces for the creation and deletion of users.
- Interfaces for the retrieval of users.
- Interfaces for validation check in case of login and registration
- Interfaces for availability setting in case of drivers.

Summary of the main interfaces required:

- Interfaces for the communications of messages/notifications.
- Interfaces for the access to MTS database.

- ***QueueManager***

Type of interfaces provided: procedure call.
Type of interfaces required: remote web service or remote procedure call.

Summary of the main interfaces offered:

- Interfaces for taxi retrieval and taxi removal from taxi queues.
- Interfaces for moving a taxi at the bottom of a queue and in general for updating taxis' position.

Summary of the main interfaces required:

- Taxis' GPS coordinates' retrieval.

- ***DatabaseManager***

  Type of interfaces provided: procedure call.
  Type of interfaces required: remote queries and operations on database.

  This component is a sort of stub acting the role of the real database.
  The interfaces offered are basically all kind of queries, insert, delete and update operations addressed to the actual tables of the real database.
  The interfaces required are the actual queries and operations on the real database.

- ***MessageBroker***

  Type of interfaces provided: procedure call.
  Type of interfaces required: remote procedure call, remote email service and web service provided by the web server.

  This component is responsible of communications with both mobile clients (directly via remote procedure calls) and web clients (indirectly via web server).
  The interfaces offered are basically procedures that forward any kind of relevant message to the clients, possibly also to multiple clients due to the pub-sub architecture style.

- ***MTSDatabase***

  Type of interfaces provided: remote queries and standard database operations.
  Type if interfaces required: none.

  All kinds of insert, update, delete operations and queries are offered.

3. Algorithm Design

This following pseudo-code describe a possible implementation of the algorithm that, in the *Queue Manager* component, search for an available taxi to satisfy a customer request. Please note that this code provide *only* an implementation suggestion. It does not constraint that, for example, the city's taxi zones should be memorized in a graph.

```
/***** QUEUE MANAGER COMPONENT *****/

---- searchTaxiInQueue() SUBROUTINE ----

// Parameters given by the caller:

customerPosition


// Initialization:

initZone = findZone(customerPosition) //retrives the current customer's zone

depth = getDEPTH() //The depth of the available taxi research should not be hard
                   //coded so that future extensions may be easier.
                   //In our model DEPTH = 1.

G = getZonesGraph //gets the GRAPH of the zones


//Algorithm core (Breadh-first search based):

for each node n in G:

        n.distance = INFINITY

        n.parent = NIL


create empty queue Q   //this is the queue on the zones to be visited


initZone.distance = 0

Q.enqueue(initZone)


while Q is not empty:


    currentZone = Q.dequeue()

    currentQueue = getQueue(currentZone)

    if(NOTEMPTY(currentQueue):   //If the queue of this zone is not empty...

        return currentQueue.DEQUEUE //... the first available taxi is returned
```

```
    for each node n that is adjacent to currentZone:  //looks for suitable
                                                      //adjoining zones
        if n.distance == INFINITY:
            if (currentZone.distance + 1 < depth):
                n.distance = currentZone.distance + 1
                n.parent = currentZone
                Q.enqueue(n)
```

---- moveBottomQueue SUBROUTINE ----

```
// Parameters given by the caller:
taxiToBeMoved

//Algorithm
zone = findZone(taxiToBeMoved)
queue = getQueue(zone)
queue.ENQUEUE(taxiToBeMoved)
```

```
/***** RIDES MANAGER *****/
```

---- request to assign a driver to a customer SUBROUTINE ----

```
customerPosition = getCustomerPosition //extract this info from the ride
taxi = searchTaxiInQueue(customerPosition) //call the subroutine in the QUEUE
                                           //MANAGER
driver = extractDriver(taxi)
askConfirmation(driver) //calls the MESSAGE BROKER to send the request to the
                        //driver
/* ... here this algorithm is over. Other logic parts of the application will be
used when the driver answers */
```

4. User Interface Design

Please refer to the "*External Interface Requirements"* paragraph in the RASD document.

5. Requirements traceability

The following table associate the requirements defined in the RASD document into the design component delineated in this document.

In the left column are listed all the requirements related to the goals, while in the right column there are the main components, as described in the component diagram and in the architectural style section, that allows that particular system functionality.

Note that components that are obviously required for almost all the functionalities, like the *Web Server* or the *Application Server*, sometimes will be omitted in order to preserve readability. The *Database Manager* won't appear in this table for the very same reason.

Goals / Requirements	**Design Components**
GOAL G1	
[R1] Customers should be able to access the service through both the web and the mobile application, even at the same time.	Web Server Application Server (Three tier architecture)
[R2] Customers must be able to register to the taxi service from the mobile or web homepage.	Web Server Application Server (Three tier architecture)
[R3] Only registered customers can access MyTaxiService's services.	Web Server Application Server Users Manager (Three tier architecture)
[R4] The system should allow the log out functionality.	Web Server Application Server Users Manager (Three tier architecture)
Notes	The servers specified in the architecture description and the internal component "Users Manager" verify and eventually allow the costumer to access the system functionalities.

GOAL G2	
[R1] Only registered customers can request a taxi ride.	Web Server Application Server Users Manager (Three tier architecture)
[R2] Customers must insert a valid origin location in order to request a ride.	Rides Manager – Checks the syntactic correctness of the inputs
[R3] The system will not allow more than a request if the previous one (either request or reservation) has not been accomplished yet.	Rides Manager

GOAL G3	
[R1] The system should allow taxi reservations for a specific path communicated by the customer.	Rides Manager
[R2] The system must not allow overlaps between reservations (or requests) made by the same customer.	Rides Manager
[R3] The system allows reservations only 2 hours before the time and date specified by the customer.	Rides Manager
[R4] The system will assign a taxi driver for the reserved ride 10 minutes before the time and date specified by the customer.	Rides Manager

GOAL G4	
[R1] Taxi drivers should be able to communicate their current availability state to the system.	Message Broker – communicate the change of status
[R2] If available, taxi drivers should be able to receive incoming requests.	Ride Manager – provides the requests Queue Manager – select the taxi driver Message Broker – send the requests
[R3] After receiving an incoming request, the taxi driver should be able to either confirm or not his intention to take charge of the request.	Message Broker – receive and
[R4] Taxi drivers must be able to log in the mobile application with preassigned credential and be identified as drivers.	Application Server Users Manager
[R5] At the end of their workshift, taxi drivers must be able to log out of the mobile application in order to communicate to the system that they are no longer active.	Application Server Users Manager

GOAL G5	
[R1] The system should always search an available taxi giving maximum priority to the taxi zone related to the request and lower priority to the immediate near zones. Any other taxi zone should be ignored.	Queue Manager – uses the breadth first algorithm to find the zones and queues
[R2] If no taxis are available in the zones specified in the previous requirement, the system should	Ride Manager Queue Manager

put the request on hold and periodically check again the taxi availability.	Message Broker – tells the customer that no taxis are available

GOAL G6	
[R1] The system should send updates through email and/or in-app notification, as specified by the customer.	Message Broker
[R2] Absence of taxis available, reservations overlaps, taxi average waiting time and taxi assigned to customers are events that must be notified to the customer.	Ride Manager – provide the events Message Broker – sends the events

GOAL G7	
[R1] Customers must leave a valid phone number in order to complete the registration phase.	Users Manager – check the consistency of the fields
[R2] Taxi drivers must be able to access to the customer's phone number when the system has paired them.	Message Broker – sends the request and all the related information
[R3] Customers must receive the taxi drivers' contact number after the system has paired them.	Message Broker – sends the updates of a request and the related data
[R4] Customers must receive the taxi code in order to be able to recognize its driver.	Message Broker – sends the updates of a request and the related data

GOAL G8	
[R1] Customers can cancel a request or reservation only if it has not been assigned to a taxi driver yet.	Rides Manager
[R2] Customers must be able to visualize the list of all their requests and reservations.	RidesManager Database Manager

GOAL G9	
[R1] Administrators must be able to create a taxi driver's account.	Application Server Users Manager
[R2] Administrators must be able to delete a taxi driver's account.	Application Server Users Manager
[R3] Administrators must be able to change the status of taxi driver.	Application Server Rides Manager Users Manager

[R4] Administrators must be able to change the status of a ride.	Application Server Rides Manager
Notes	The application server provide Admins the direct access to the Users Manager.

GOAL G10	
[R1] Customers and taxi drivers must be able to visualize, both in the mobile and web application, a support phone number which they can call to obtain assistance.	Web Server Application Server

Notice also that the component design has partially satisfied some of the *Non Functional Requirements*: for example, the ones about the reactivity of the application are now simply achievable through our light-oriented system.

6. References

6.1 Hours of work

Alessandro Pozzi ~22 hours

Marco Romani ~22 hours

6.2 Software and tools used

- Microsoft Word (https://products.office.com/it-it/word) to redact and to format this document.
- Astah Professional (http://astah.net/) to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Diagrams.
- GitHub (https://github.com) to share the working material of this project.
- Notepad++ (https://notepad-plus-plus.org/) to write the algorithm's code.