Politecnico di Milano, December 2015

DESIGN DOCUMENT

myTaxiService

Petar Korda
Krishnan Ranjithkumar

Contents

1 Introduction

1.1 Purpose

This document represents the Design Document for the software system myTaxiService, and the most significant goal of this document is to completely provide the high - level and low-level descriptions of the different components used in the system and providing the insight in to interaction among the components ,algorithm design and user interface of the system. This document also consist of typical runtime view of the system as well.

This document is intended to be used by developers, programmer and testers, who will implement and test the system, system analysts and requirement analysts for inter related systems, project managers, customers and users of the system.

1.2 Scope

Software system myTaxiService is a project meant to optimize the taxi services of the city, by providing the mobile and web application that will allow users to request a taxi, or book the ride in advance and cancel the ride. It will also provide a fair management of taxi queues for the taxi drivers, and maximize the profit for the city.

In particular, city is divided into taxi zones, and each zone is assigned to a taxi queue (the system automatically calculates the distribution of taxis in zones based on the GPS information it receives from taxis). If the taxi is available (this information is provided by the taxi driver, who informs the system about his/hers availability through a mobile application), its identifier is stored in a queue of taxis in the corresponding zone. When a user requests a taxi from a zone (through a mobile or web application), the system forwards it to the first taxi queuing in that zone. Taxi driver can then confirm or deny the request through the mobile application. If the request is denied system will forward the request to a next taxi in the queue (and move the first taxi to the end of the queue), or if the queue is empty it will make sure that another taxi gets a request. When the taxi driver confirms the request, the passenger will be informed by the system about the taxi identifier and the waiting time. The passenger can also book a taxi, specifying the starting point and the destination, and the system then allocates a taxi 10 minutes before the meeting time with the passenger.

1.3 Definitions, Acronyms, Abbreviations

Acronyms:

- RASD - Requirements and Specification Document
- DD - Design Document
- EC2 - Amazon Elastic Cloud Compute
- EBS - Amazon Elastic Block Storage

- S3 - Amazon Simple Storage
- ELB - Amazon Elastic Load Balancer
- DB - Database

Abbreviations:

- [Gn] - Goal n referred to the goal from the RASD

1.4 Reference Documents

- Specification Document: myTaxiService Project AA, 2015 - 2016
- IEEE Standard 1016: Software Design Specification
- Requirements and Specification Document: myTaxiService RASD
- Design Patterns: Elements of Reusable Object-Oriented Software

1.5 Document Structure

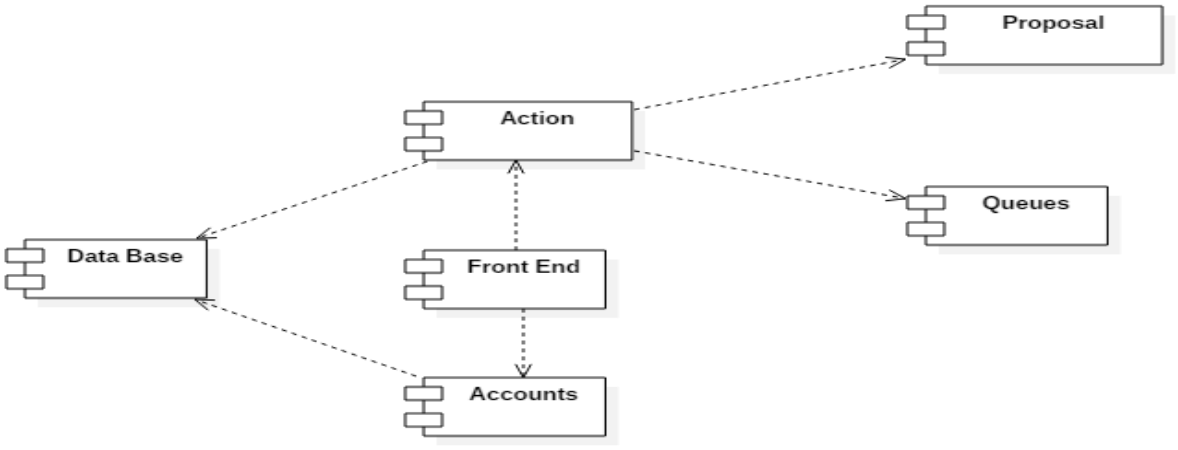
- **Introduction:** It gives description and organization of the document .
- **Architectural design:** It gives the low level description of the software systems such as component view, runtime etc.,
- **Algorithm design:** It gives the step by step procedure of the software system for both requesting and reserving a taxi.
- **User interface design:** It gives visual appearance of different layers of the system i.e., look and feel of the software system.
- **Requirements traceability:** It gives the description of required components for each goal of the system.

2 Architectural Design

2.1 Overview

In the following chapters the architectural design of the myTaxiService is given, starting with a high level view of components, their detailed views and interactions and their runtime sequence diagrams are shown for the cases of taxi requests and reservations. Also, infrastructure and the styles used are described with addition to mapping of software artifacts to the relevant hardware components.

2.2 High level components and their interaction



The system can receive several requests from the clients which are divided into two categories:

1. **Actions**
    - *Taxi Request* - User requests a taxi with a starting point and a destination, the system then sends him the proposal (fare amount, waiting time etc.) and if the user accepts and pays, it allocates a vehicle to his starting point.

    - *Taxi Reservation* - User reserves a taxi with a starting point, destination and time and date. The system then sends him the proposal (fare amount, waiting time etc.) and if the user accepts and pays, it schedules the allocation of the vehicle 10 minutes before the specified time.

    - *Request/Reservation Cancelation* - At any time user can cancel his request/reservation.

    The module *Actions* is responsible for these types of requests and is consisted of several components: *RequestManager*(which takes care of requests), *ReservationManager* (which takes care of reservations) and *NotificationManager* (which is used for sending notifications to users about requests/reservations).

2. **Accounts**

    - *Registration* - Guests can register to the application.

- *Log in* - Registered users can register to the application using their credentials.

- *Manage Accounts* - Logged in users can modified their accounts in several ways (changing their information, adding their most commonly used addresses and payment methods etc.).

The module *Accounts* is responsible for handling these types of requests and it is consisted of a *UserManager* component.

Actions and *Accounts* modules interact with a database for various tasks: user registration, verification, recording requests/reservations etc.

Request/Reservation proposals are generated through *Proposal* module which is consisted of several other components: *ProposalManager* (receives the requests for generating proposals and puts all information together), *FareCalculator* (calculates the fare amount user has to pay), *WaitingCalculator* (calculates the waiting time for the users), *PaymentManager* (is used for transactions).

Queue module provides interfaces for finding a free taxi, dispatching the vehicle, notifying taxi driver and interaction with zone queues. It is consisted of several other components: *ZoneManager* (responsible for zones of the city), Queue (each zone has a queue of available taxies), *TaxiManager* (this component is located on the drivers devices and is used for driver and system interaction).

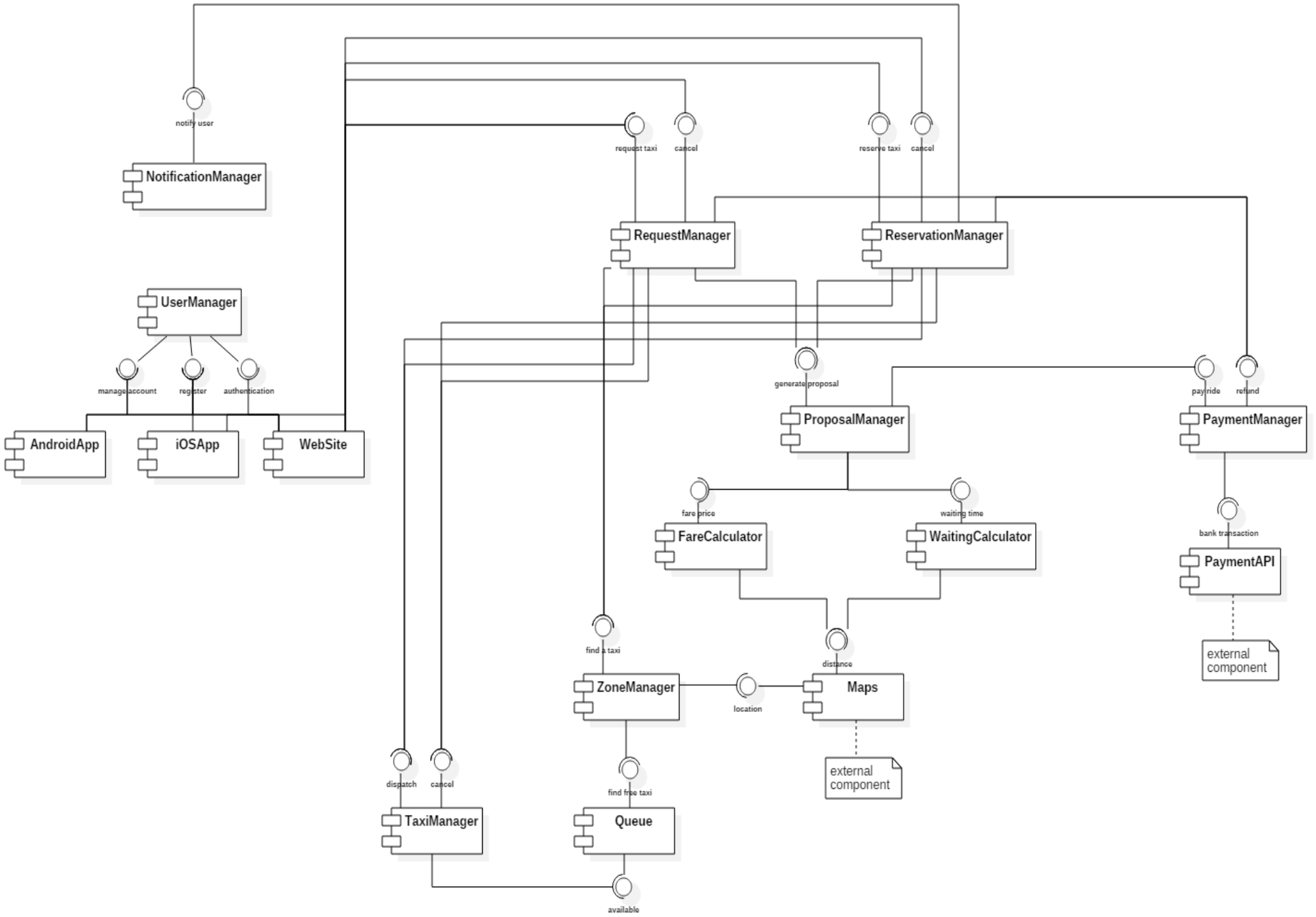
User interacts with the system through:

- *Mobile application* - Android and iOS device supported
- *Web site*

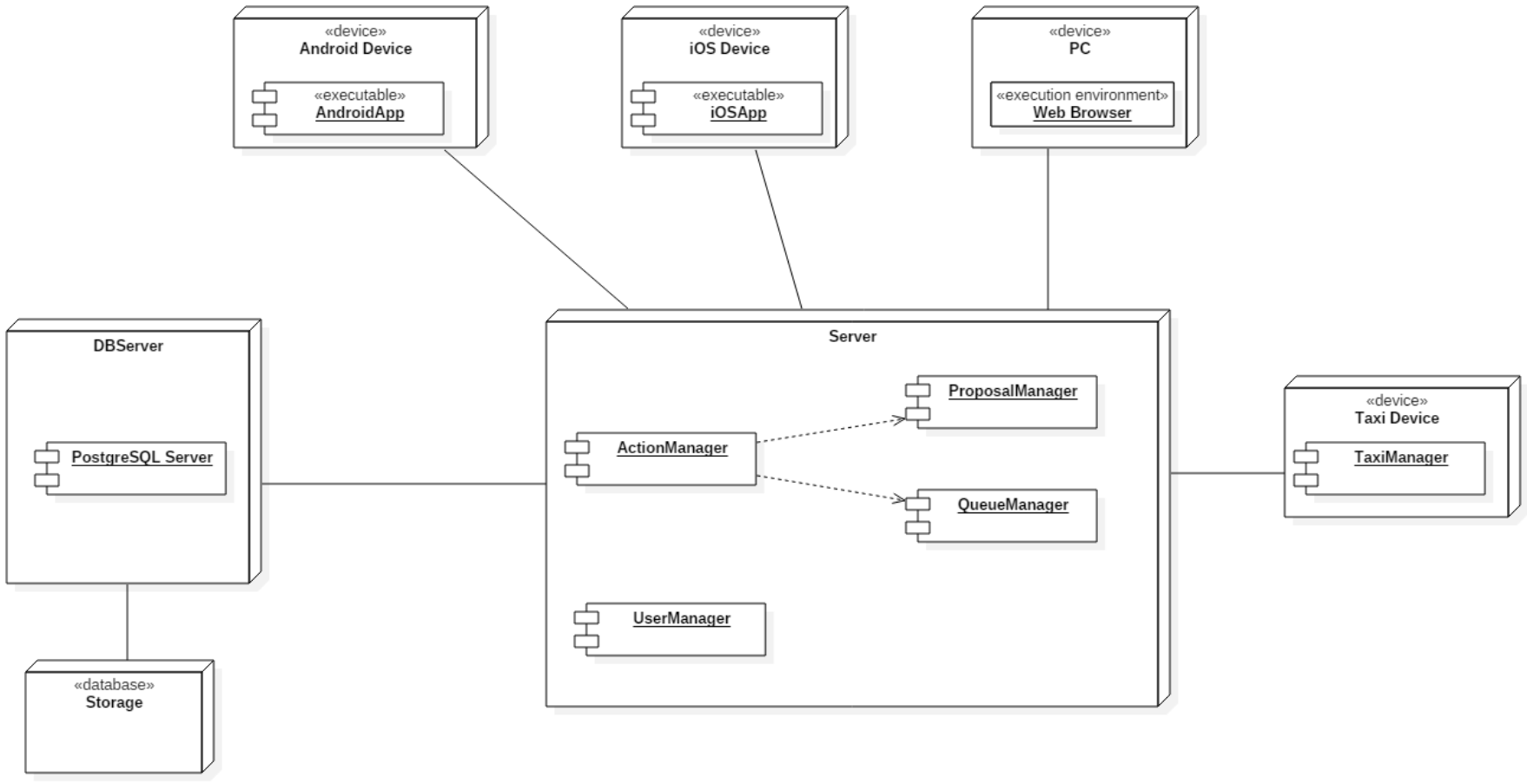
These are all included into the Front End component which is consisted of several other components: *AndroidApp*, *iOSApp* and *WebSite*.

2.3 Component View

In this section a detailed view of all components(described in the high level view) of the system and their dependencies is given through a UML diagram.


notify user
NotificationManager
request taxi
cancel
reserve taxi
cancel
RequestManager
ReservationManager
UserManager
manage account
register
authentication
AndroidApp
iOSApp
WebSite
generate proposal
pay ride
refund
ProposalManager
PaymentManager
fare price
waiting time
FareCalculator
WaitingCalculator
bank transaction
PaymentAPI
external component
find a taxi
distance
ZoneManager
location
Maps
external component
dispatch
cancel
find free taxi
TaxiManager
Queue
available

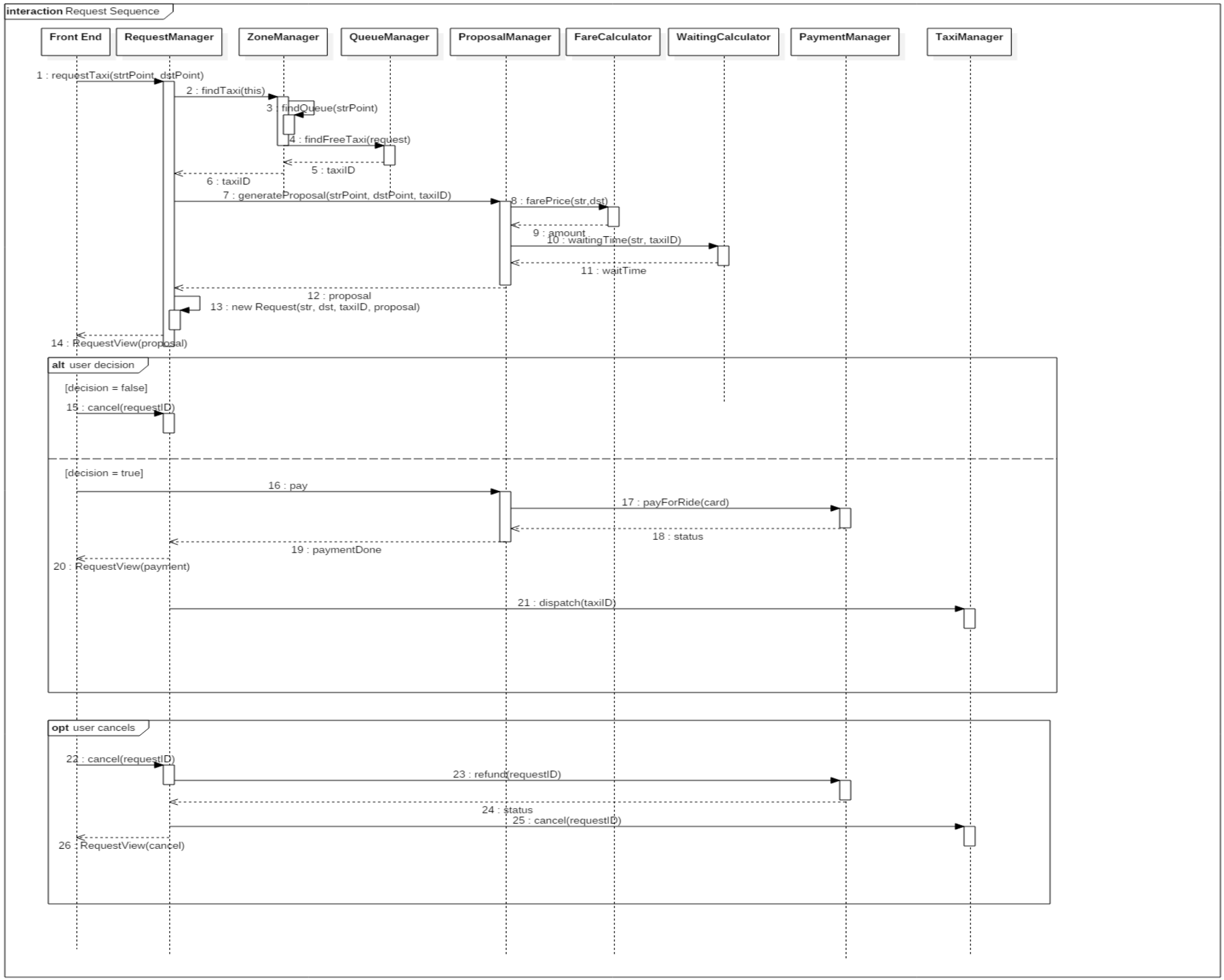
2.4 Deployment View

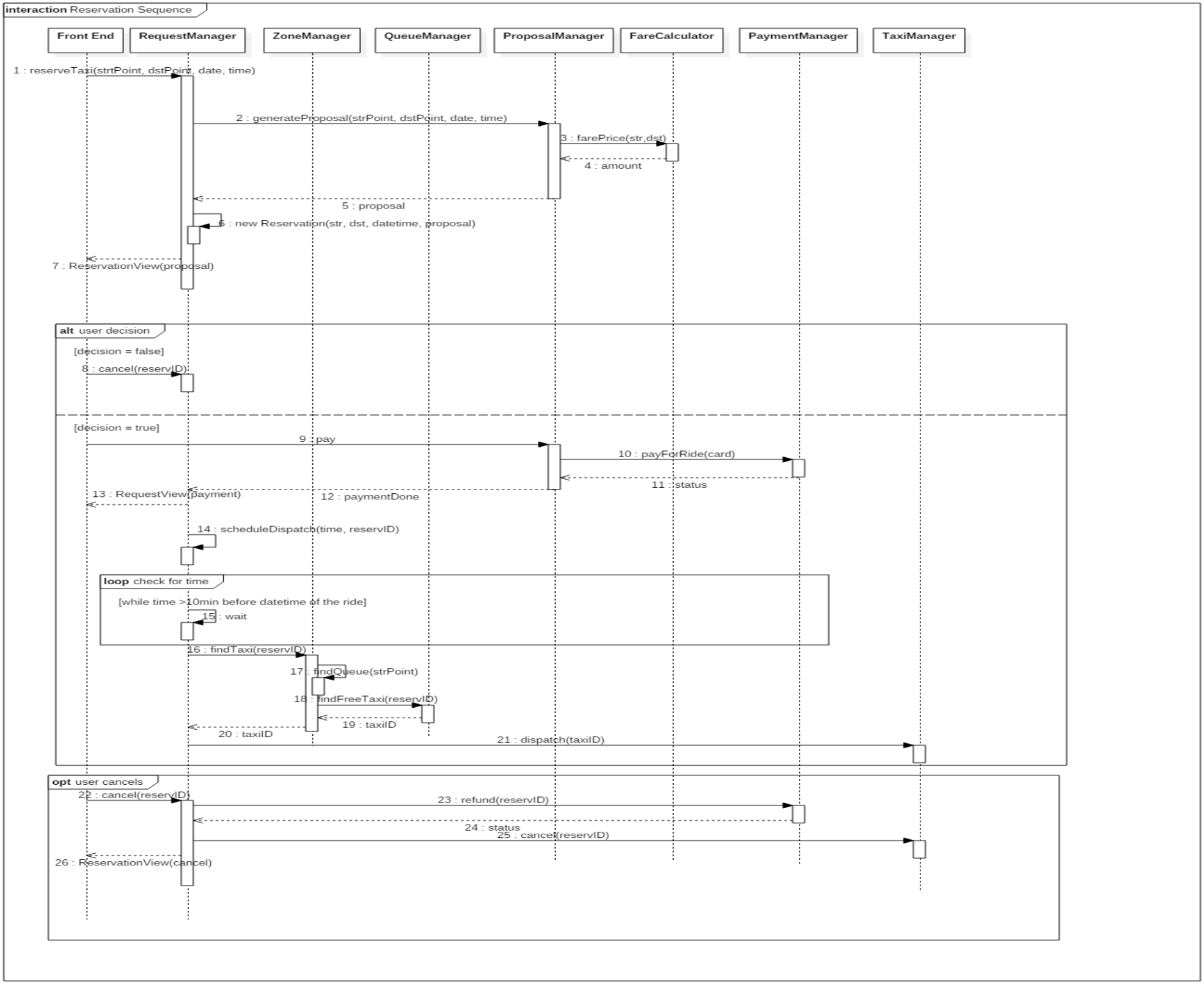


In this diagram the mapping of software components and hardware is shown. Communication between devices (user devices and driver device) is done through HTTP/HTTPS protocol, and communication between server and database server through TCP/IP. Each of these nodes and their architectural implementation is given in the section *Architecture styles and design patterns*.

2.5 Runtime View

In this section interaction between components in runtime is shown on the cases of Taxi requests and Taxi reservations. *RequestManager* and *ReservationManager* are both implement Runnable interface and can serve multiple requests/reservations.


interaction Request Sequence
Front End
RequestManager
ZoneManager
QueueManager
ProposalManager
FareCalculator
WaitingCalculator
PaymentManager
TaxiManager
1 : requestTaxi(strtPoint, dstPoint)
2 : findTaxi(this)
3 : findQueue(strPoint)
4 : findFreeTaxi(request)
5 : taxiID
6 : taxiID
7 : generateProposal(strPoint, dstPoint, taxiID)
8 : farePrice(str,dst)
9 : amount
10 : waitingTime(str, taxiID)
11 : waitTime
12 : proposal
13 : new Request(str, dst, taxiID, proposal)
14 : RequestView(proposal)
alt user decision
[decision = false]
15 : cancel(requestID)
[decision = true]
16 : pay
17 : payForRide(card)
18 : status
19 : paymentDone
20 : RequestView(payment)
21 : dispatch(taxiID)
opt user cancels
22 : cancel(requestID)
23 : refund(requestID)
24 : status
25 : cancel(requestID)
26 : RequestView(cancel)


interaction Reservation Sequence
Front End
RequestManager
ZoneManager
QueueManager
ProposalManager
FareCalculator
PaymentManager
TaxiManager
1 : reserveTaxi(strtPoint, dstPoint, date, time)
2 : generateProposal(strPoint, dstPoint, date, time)
3 : farePrice(str,dst)
4 : amount
5 : proposal
6 : new Reservation(str, dst, datetime, proposal)
7 : ReservationView(proposal)
alt user decision
[decision = false]
8 : cancel(reservID)
[decision = true]
9 : pay
10 : payForRide(card)
11 : status
12 : paymentDone
13 : RequestView(payment)
14 : scheduleDispatch(time, reservID)
loop check for time
[while time >10min before datetime of the ride]
15 : wait
16 : findTaxi(reservID)
17 : findQueue(strPoint)
18 : findFreeTaxi(reservID)
19 : taxiID
20 : taxiID
21 : dispatch(taxiID)
opt user cancels
22 : cancel(reservID)
23 : refund(reservID)
24 : status
25 : cancel(reservID)
26 : ReservationView(cancel)

2.6 Component interfaces

Here the interfaces of all the components are listed, with a description of their input and output.

2.6.1 Request Manager

- **Request taxi** :

  Clients call this interface when a user requests a taxi.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Requesting a taxi	**Starting point** (type: Location) **Destination point**(type: Location)	Step 1: returns a **RequestView(prop)** object with associated Proposal p Step 2 (if user accepts and pays): **RequestView(conf)** confirmation object //otherwise returns **RequestView(cancel)** cancelation object

- **Cancel** :

  Clients call this interface when a user wants to cancel a taxi request.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Cancelation of a taxi request	**Request ID** (type: int)	**RequestView(cancel)** object - cancelation confirmation

2.6.2 Reservation Manager

- **Reserve taxi :**

Clients call this interface when a user reserves a taxi.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Reserving a taxi	**Starting point** (type: Location) **Destination point**(type: Location) **Datetime** (type: Datetime)	Step 1: returns a **ReservationView(prop)** object with associated Proposal p Step 2 (if user accepts and pays): **ReservationView(conf)** confirmation object //otherwise returns **ReservationView(cancel)** cancelation object

- **Cancel**:

Clients call this interface when a user wants to cancel a taxi reservation.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Cancelation of a taxi reservation	**Reservation ID** (type: int)	**ReservationView(cancel)** object - cancelation confirmation

2.6.3 Proposal Manager

- **Generate proposal:**

This interface is called by the *RequestManager* or *ReservationManager* in order to generate a proposal for the user.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Generating a proposal	**Starting point** (type: Location) **Destination point**(type: Location)	**Proposal** object

	[TaxiID [if request]] (type: int) **[Datetime [if reservation]]** (type: Datetime)	

2.6.4 Fare Calculator

- **Fare price:**

  This interface is called by the *ProposalManager* when its generating a proposal.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Fare amount calculation for the ride	**Starting point** (type: Location) **Destination point**(type: Location)	**Amount** object

2.6.5 Waiting Calculator

- **Waiting time:**

  This interface is called by the *ProposalManager* when its generating a proposal for a request (for reservation waiting time is not calculated).

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Fare amount calculation for the ride	**Starting point** (type: Location) **Taxi location**(type: Location)	**Time** object

2.6.6 Zone Manager

- **Find a taxi:**

  This interface is called by the *RequestManager* or *ReservationManager* in order to forward a request to a taxi.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Finds an available taxi that wants to take care of a request.	**Request** (type: Request)	**taxiID** (type: int) - id of a taxi that accepted the request

2.6.7 Queue

- **Find free taxi**:

  This interface is called by the *ZoneManager* and forwards a request to a *Queue* associated with a zone in which a starting point is (or another zone if there are no free taxies in this one).

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Goes through a queue of available taxies until one of them accepts it (otherwise if no taxi from this queue accepts it, an information is returned to *ZoneManager*)	**Request** (type: Request)	**status** (type: int) - id of a taxi that accepted the request (otherwise returns -1 meaning that there are no available taxies or everyone declined it)

- **Available**:

  This interface is called by the *TaxiManager* to notify the queue that the taxi is available for rides.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Notify the queue that a taxi is available for rides (put a taxi in a queue).	**Taxi ID** (type: int)	**confirmation** (type: int) - 0 if operation successful, -1 otherwise

2.6.8 Taxi Manager

- **Dispatch**:

  This interface is called by the *RequestManager* or *ReservationManager* in order to notify the driver that he can leave to pick up the passanger.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Notify driver to pick up the passanger	**Request** (type: Request)	**confirmation** (type: int) - 0 if operation successful, -1 otherwise

- **Cancel**:

  This interface is called by the *RequestManager* or *ReservationManager* in order to notify the driver that the request has been canceled.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Notify driver that the request has been canceled.	**Request** (type: Request)	**confirmation** (type: int) - 0 if operation successful, -1 otherwise

2.6.9 Payment Manager

- **Pay for ride**:

  This interface is called by the *ProposalManager* when the user pays for the ride.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Pay for the ride	**Credit Card** (type: Card)	**confirmation** (type: int) - 0 if operation successful, -1 otherwise

- **Refund**:

  This interface is called by the *RequestManager* or *ReservationManager* when the user cancels the request or reservation.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Calculate the refund amount and refund to user.	**Action to cancel**(type: Action)	**Amount** (type: Amount) - amount refunded, or NULL is unsuccessful

2.6.10 Notification Manager

- **Notify user**:

  This interface is called by the *ReservationManager* in order to notify the user about his upcoming rides.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Notify user about the ride	**Reservation** (type: Reservation)	**confirmation** (type: int) - 0 if operation successful, -1 otherwise

2.6.11 User Manager

- **Register**:

  This interface is called by the *Front End* when the guest submits a registration form.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Registers a guest	**Name** (type: String) **Email** (type: String) **Phone number** (type: String) **Password** (type: String)	**status** (type: int) - 0 if operation successful, otherwise send and integer n (error code)

- **Log in**:

  This interface is called by the *Front End* when the user submits a log in form.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Log in a registered user	**Email** (type: String) **[Phone number** if Email == NULL](type: String) **Password** (type: String)	**status** (type: int) - 0 if operation successful, otherwise send and integer n (error code)

- **Manage Account**:

  This interface is called by the *Front End* when a user modifies this profile.

OPERATION PROVIDED	INPUT PARAMETERS	OUTPUT
Change profile information (email, password, phone number) Add address into the commonly used addresses Add payment method Change settings	New email (type: String) New password (type: String) New phone (type: String)	**status** (type: int) - 0 if operation successful, otherwise send and integer n (error code)
	New address (type: Address)	
	New payment method (type: Card)	
	New settings (type: Settings)	

2.7 Selected architectural styles and patterns

2.7.1 Architectural styles

Architecture styles chosen for myTaxiService are:

- **Cloud at the IaaS level**

Amazon Cloud Computing platform (Amazon Web Services) is used in the means of the infrastructure for the system. The reasons for this choice are several, including:

Affordability - there are no initial costs for the infrastructure provided, and the payment is done only based on how and how much the resources used ("Pay only for what you use")

Scalability - AWS provides an Auto Scaling function, meaning that it can allocate resources based on the real need of the system, which is ideal for myTaxiService for few reasons: usage of the service varies depending on the time of the day (during the night there are typically less requests), or day in the year (during the working days there will probably be more requests than weekends), also we don't know how fast will the users adopt this application, so the smarter thing is for the infrastructure to grow based on the actual need and not spending a lot of money for hardware when unpredictable situations can occur.

Availability - Services provided by AWS provide specific features such as Availability Zones, Snapshots, Backups etc. that can be used to build a very high - available system when combined together.

In the next picture the components used and their relations are presented:

- Application Server - EC2 (Amazon Elastic Compute Cloud)

  Virtual machine that can be automatically resized based on the computation capacity needed. It will be used for the need of the application server. Two instances will be always active in two different Availability Zones (so that the application server is always available), which are going to be load balanced.

- Database Server - EC2 (Amazon Elastic Compute Cloud)

  Separate virtual machine used for the database server, with which application server interacts.

- Database Storage - EBS (Amazon Elastic Block Storage)

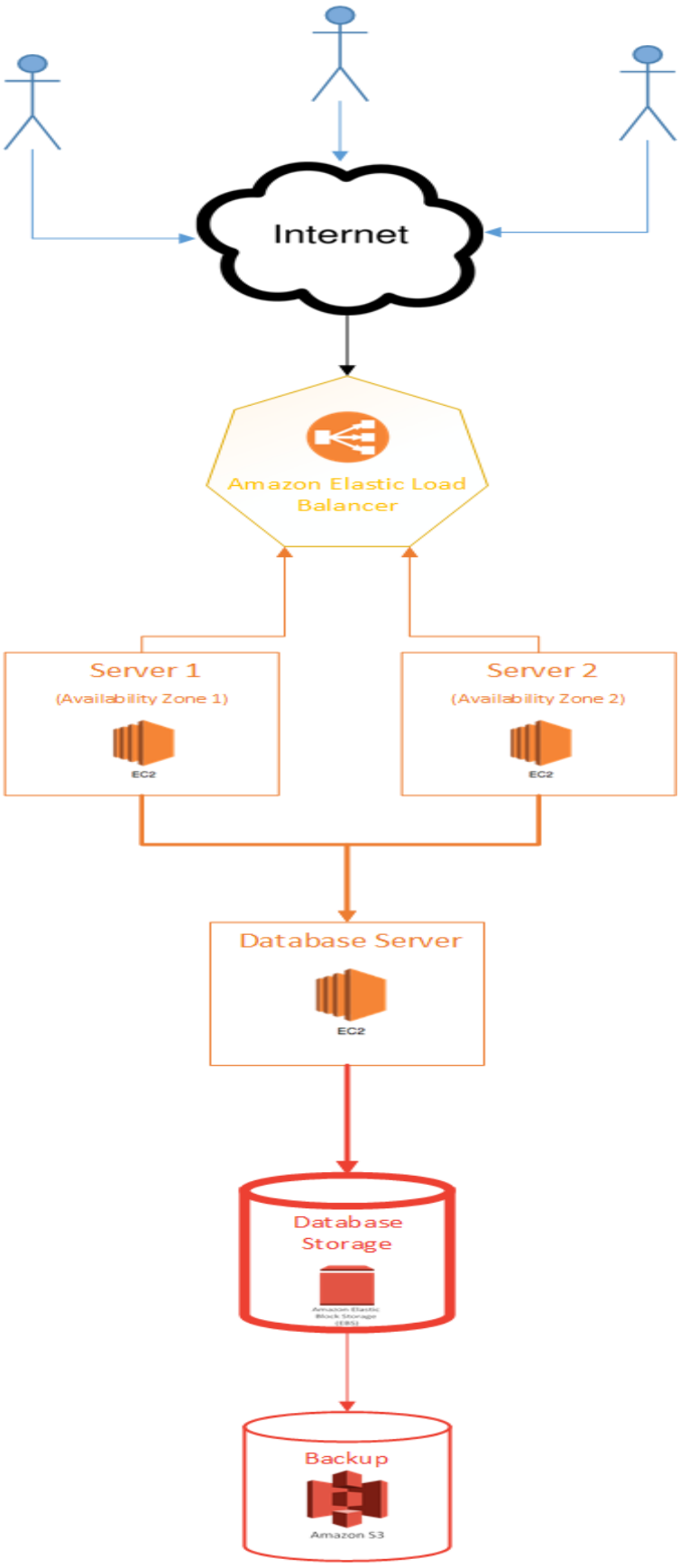
  Persistent block level storage volume attached to the database server. It will be used as the storage for the database. EBS is automatically replicated within the Availability Zone for failures protection and high availability.

- Backup - S3 (Amazon Simple Storage Service)

  Used for a database automatic backups and snapshots, so that the system can recover in case of a failure.

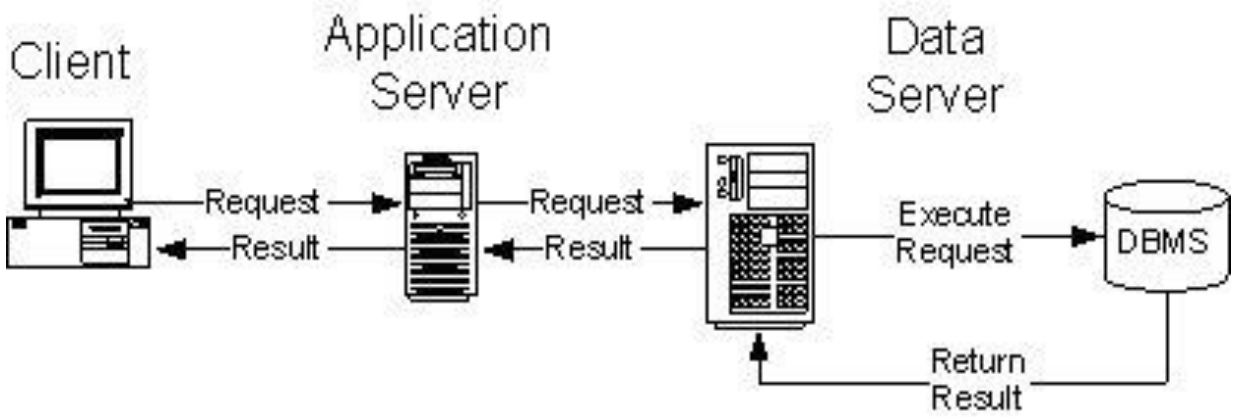
- Load Balancer - ELB (Amazon Elastic Load Balancer)

  Used for balancing the load between two application servers and directing the traffic to one of the servers if one of them is down


Internet
Amazon Elastic Load Balancer
Server 1
(Availability Zone 1)
EC2
Server 2
(Availability Zone 2)
EC2
Database Server
EC2
Database Storage
Backup
Amazon S3

- **3-Tier Client Server**

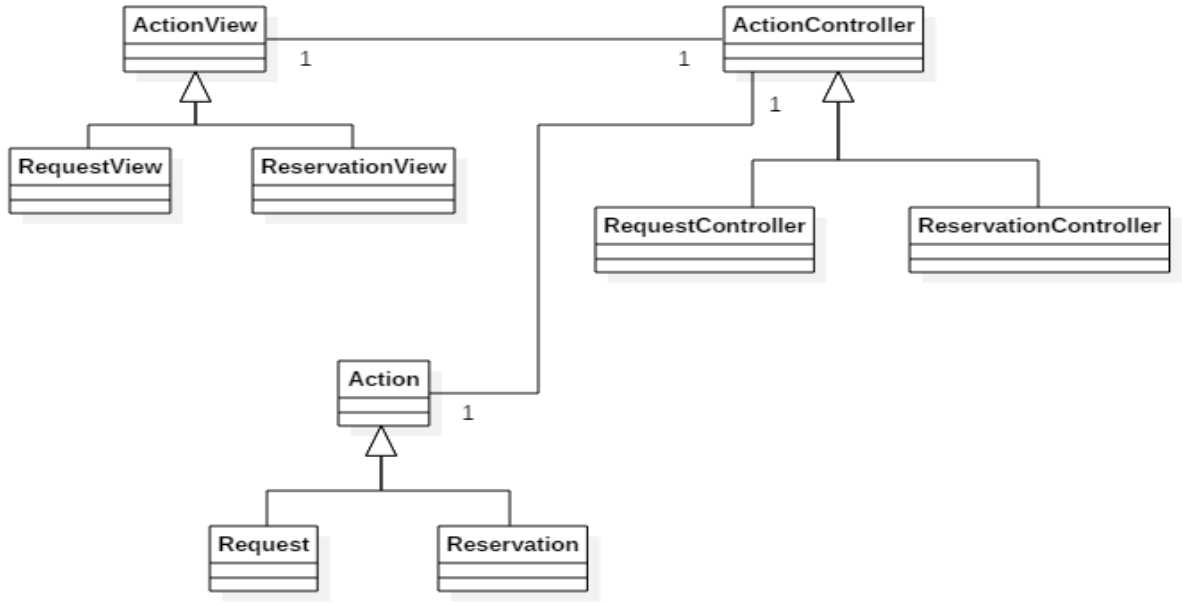
In the previous paragraph the infrastructure of the system is explained, and on top of that infrastructure a 3 - Tier Client Server style is being used.



2.7.2 Design Patterns

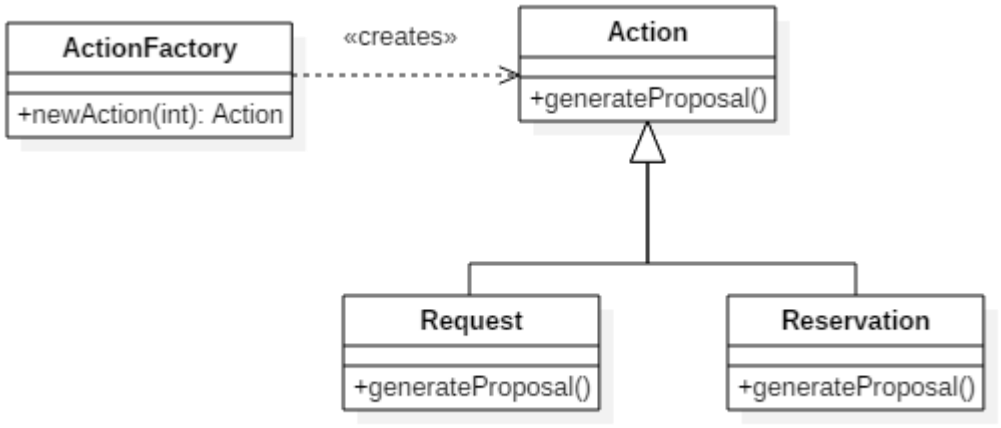
Chosen design pattern that are going to be implemented are:

- Model - View - Controller (MVC) for Actions (requests and reservations)



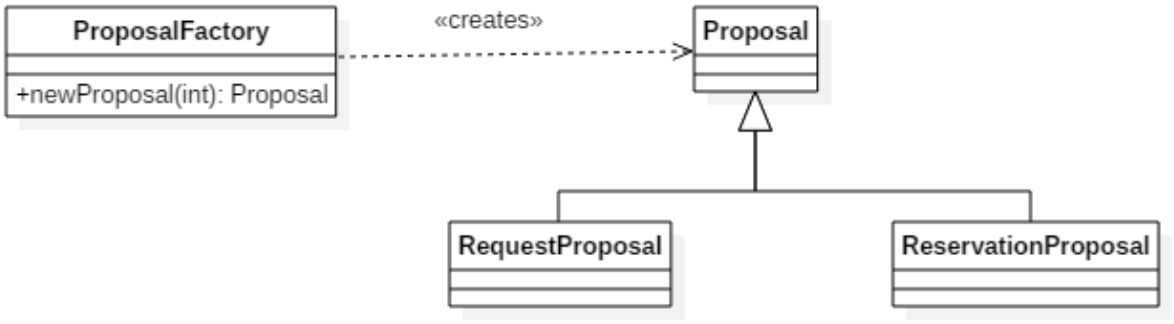
With this it the separation of the user interface, logic and data is accomplished. *ActionView (RequestView, ReservationView)* is responsible for the representation of requests and reservations, while *Action (Request, Reservation)* is the model and does all the manipulation with actual data. ActionController(RequestController, ReservationController) is in between views and models.

- Factory for Actions



All *Actions (Requests and Reservation)* objects are created through the *ActionFactory* by calling the *newAction* method (the parameter is the integer specifying whether a *Request* or *Reservation* object should be created). This is done in order to encapsulate the object creation.

- Factory for Proposals



All *Proposal (RequestProposal and ReservationProposal)* objects are created through the *ProposalFactory* by calling the *newProposal* method (the parameter is the integer specifying whether a *RequestProposal* or *ReservationProposal* object should be created). This is done in order to encapsulate the object creation.

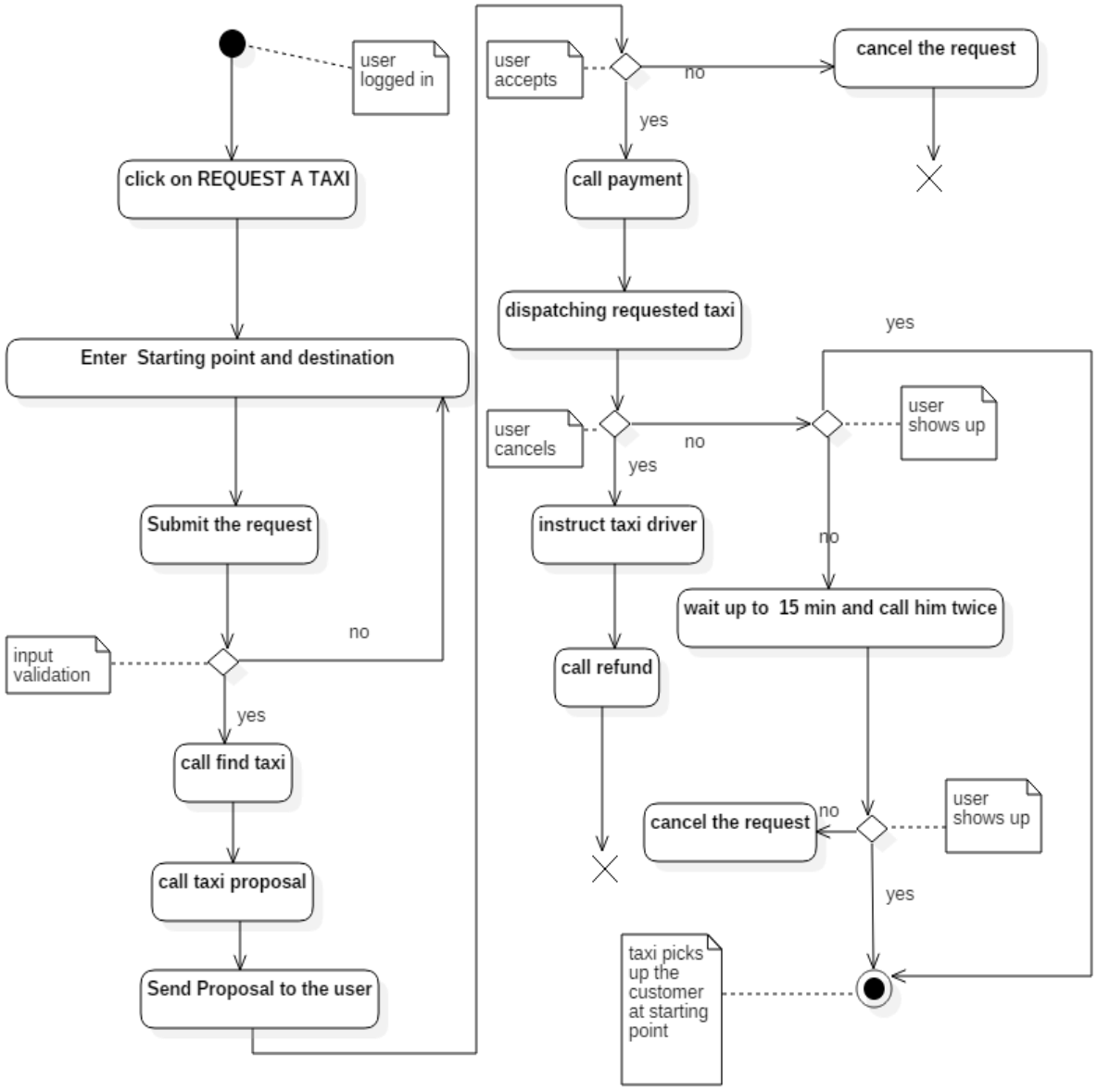
2.8 Other design decisions

We decided to build mobile applications for Android and iOS platforms only, because of the current market trend (more than 80% of the smart phones are either Android or iOS).

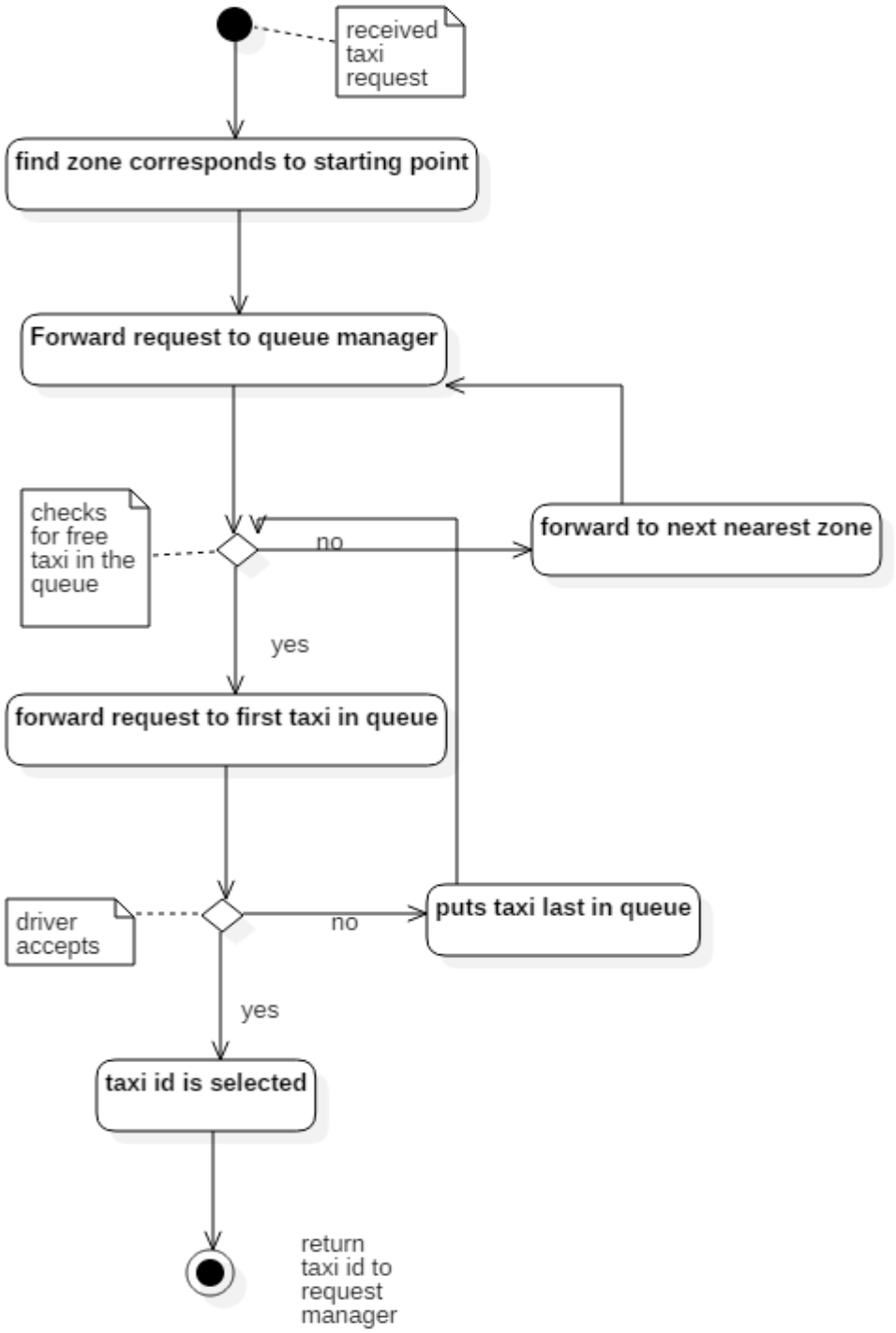
3 Algorithm Design

Algorithms for Taxi requests and Taxi reservations, and those that accompany those are given in this section.

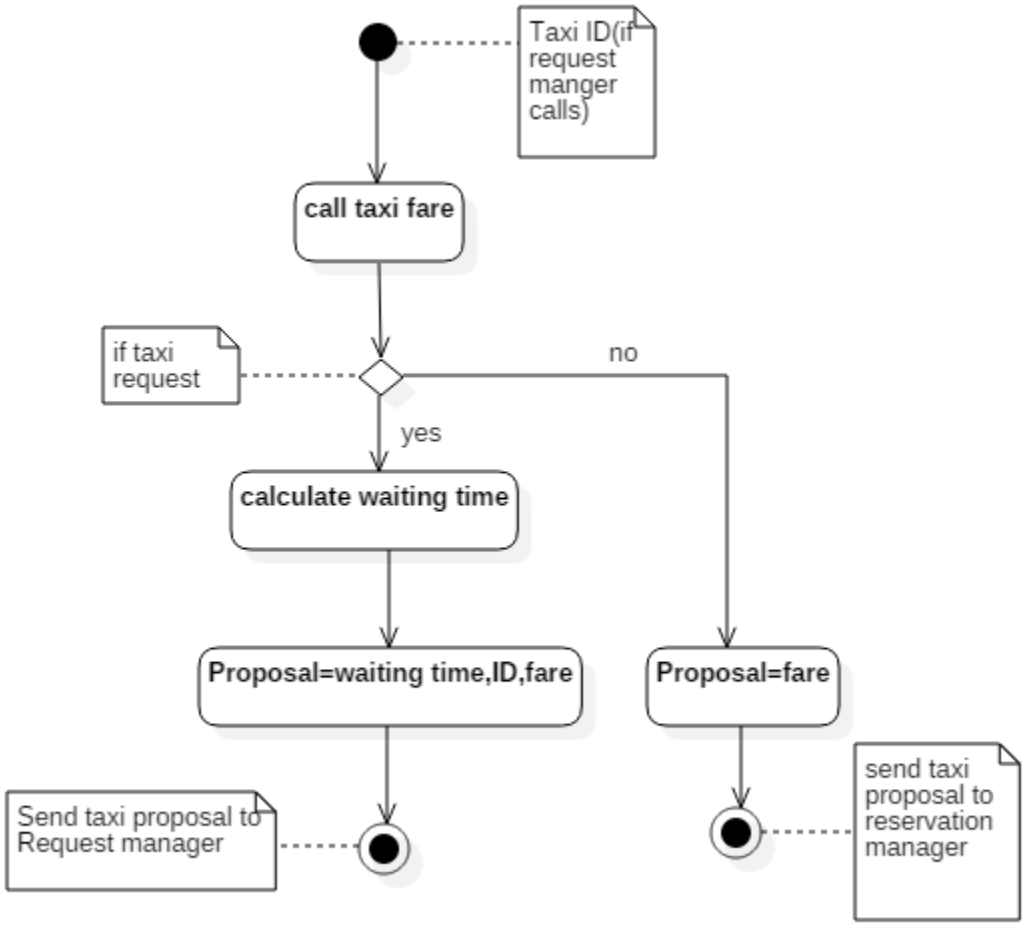
3.1 Taxi Request



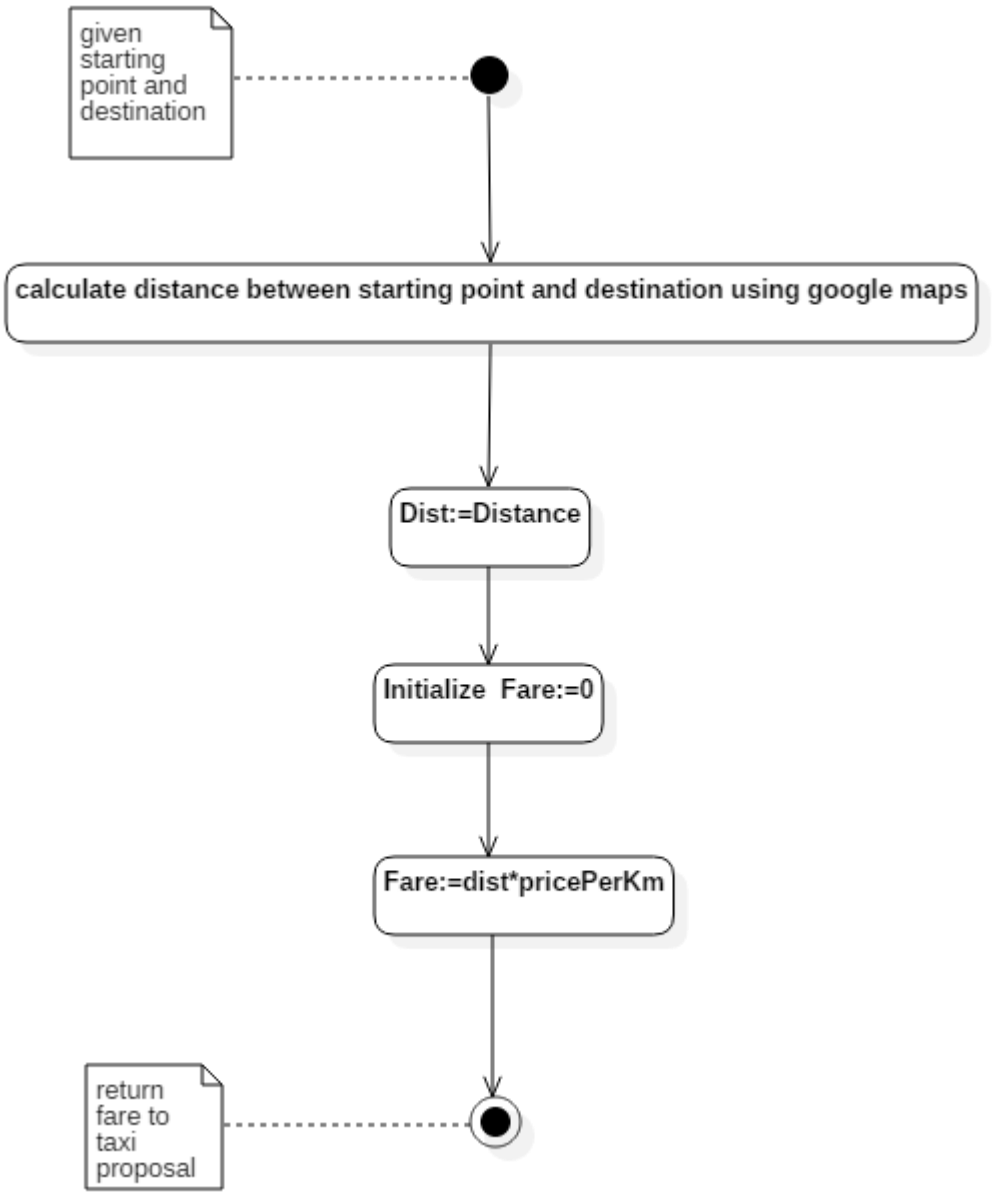
Finding an available taxi to take care of the call is done by the following algorithm (this is same for requests and reservations):



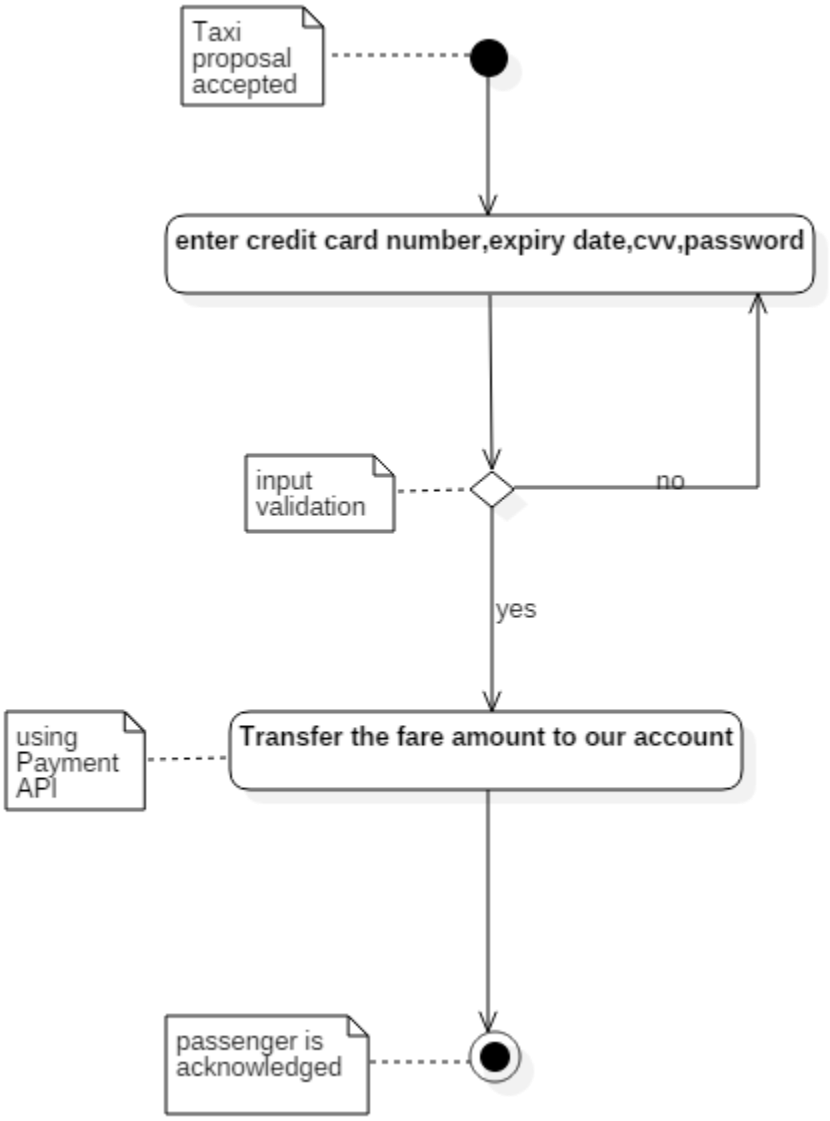
Generating a Taxi Proposal is done in the following manner (same for reservation also):



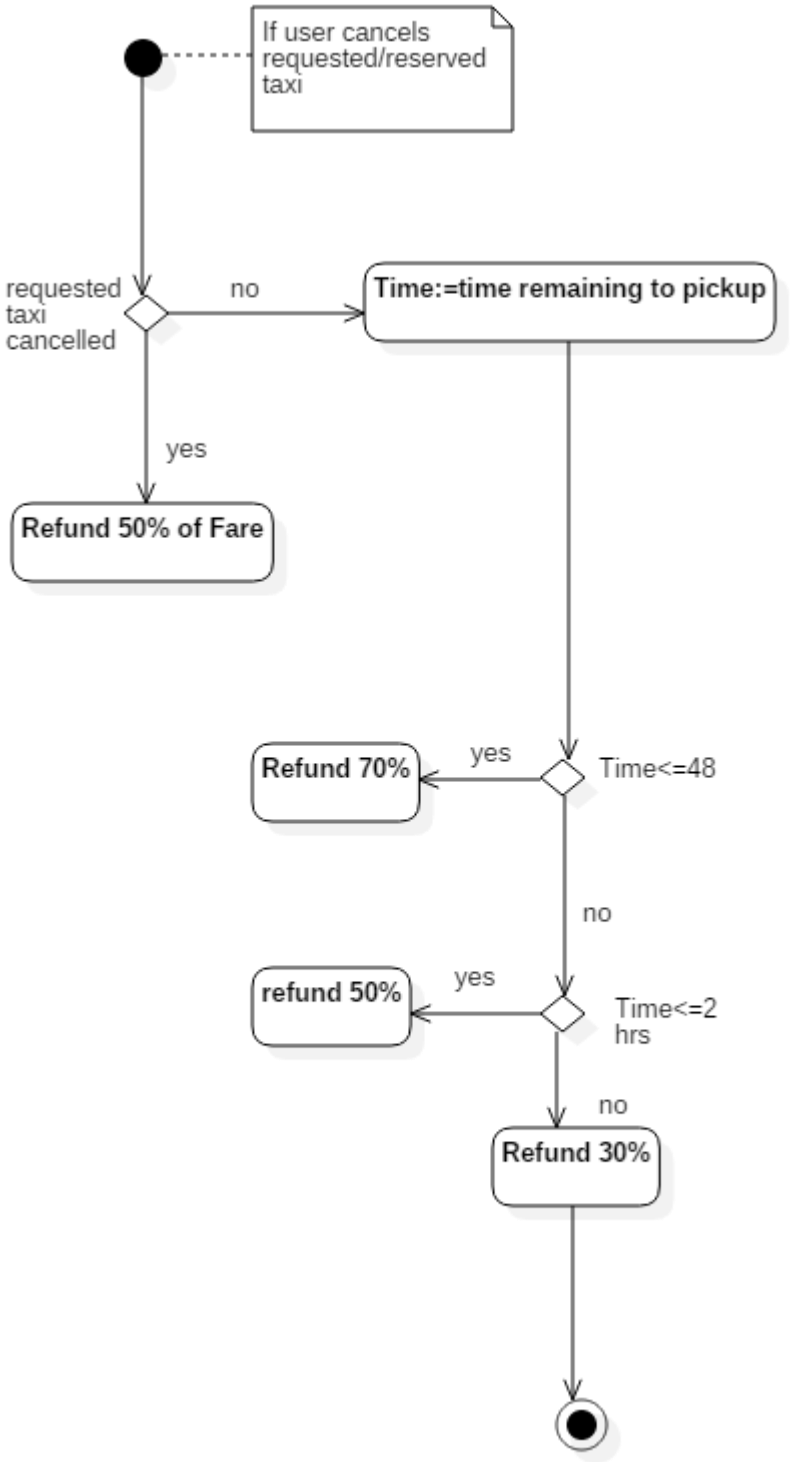
Taxi fare is calculated in following way:



Algorithm for payments (same for reservations):

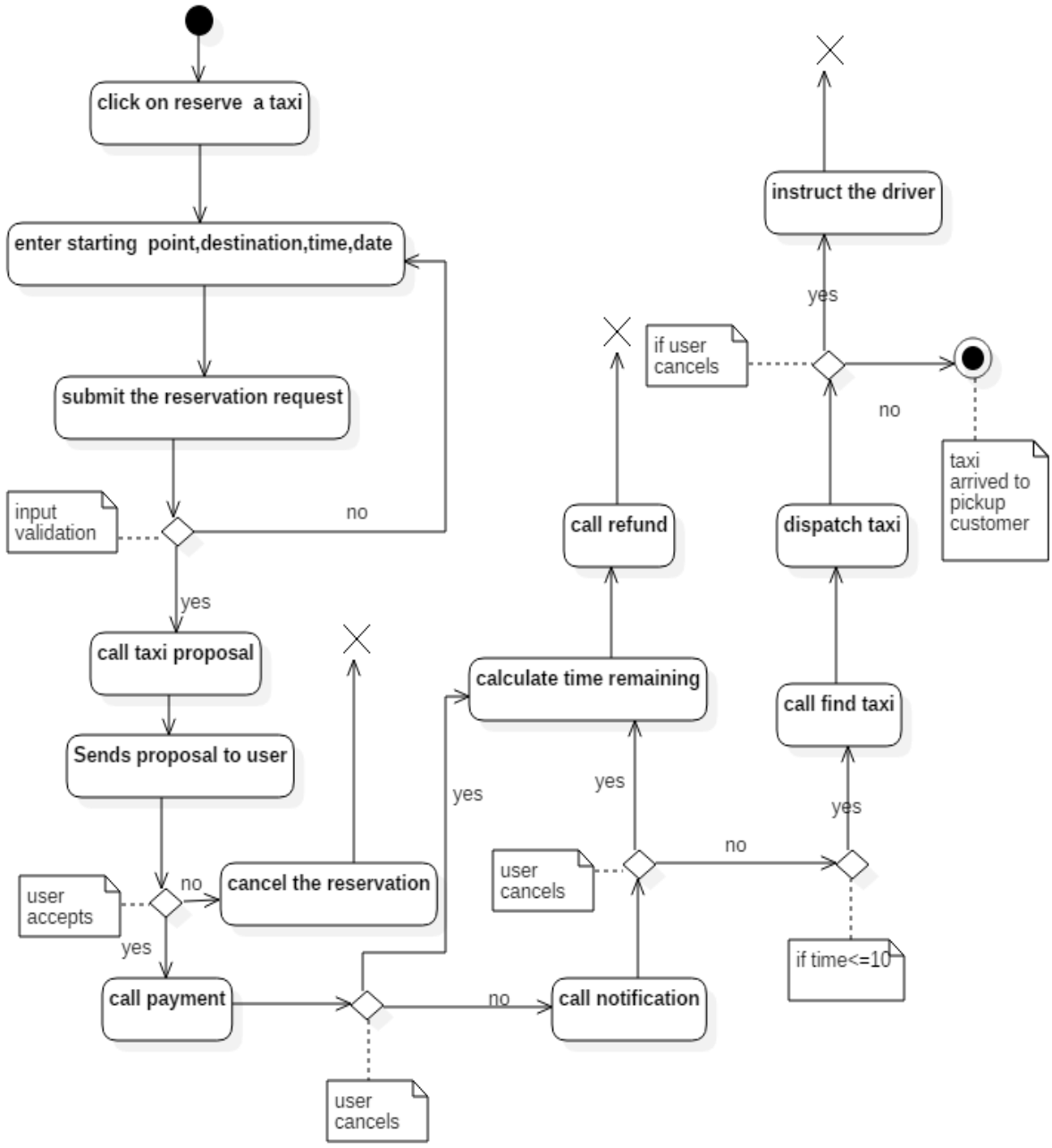


If the user cancels the request (also reservation), the algorithm bellow takes care of refunding:

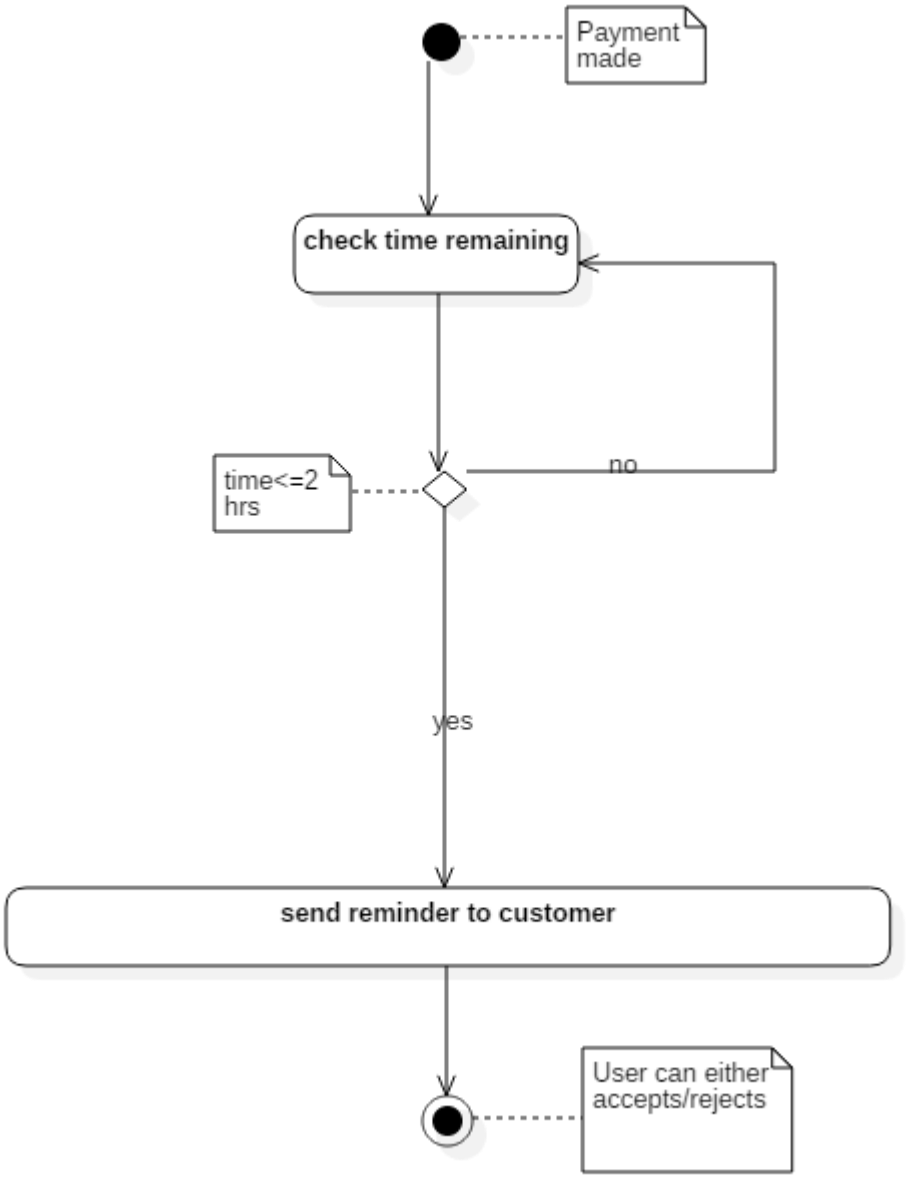


3.2 Taxi Reservation

In the case of taxi reservation the following algorithm is applied (taxi proposal, payment, refund and find taxi algorithms are described in the previous section):



For reservation notifications the following algorithm is applied:



4 User Interface Design

Refer to the section 3.1.1 of the myTaxiService Requirements and specification document (RASD).

5 Requirements Traceability

REQUIREMENT	DESCRIPTION	ARCHITECTURAL/DESIGN ELEMENT
[G1]	Allow visitors to register to the application	UserManager Front End Database
[G2]	Allow users to log in to the application, using the same credentials for mobile and web app	UserManager Front End Database
[G3]	Passenger can request a taxi	RequestManager ZoneManger QueueManager
[G4]	The system will notify the passenger with the taxi code, fare amount and waiting time, when a taxi driver accepts the system's request	ProposalManager FareCalculator WaitCalculator RequestManager
[G5]	The passenger can either accept or cancel the taxi proposal has to pay the fare amount	ProposalManager PaymentManager RequestManager
[G6]	Taxi is dispatched and will arrive at the requested location	RequestManager ReservationManager TaxiManager
[G7]	Passenger can reserve a taxi ride specifying his starting point, his destination and the desired time	ReservationManager ProposalManager
[G8]	When the passenger reserves a taxi ride, he will be informed with a code of the incoming taxi 10 minutes before the specified time of the ride	ReservationManager ZoneManager QueueManager NotificationManager
[G9]	The passenger can cancel his request	RequestManager PaymentManager
[G10]	The passenger can cancel his reservation	ReservationManager PaymentManager
[G11]	Available driver can confirm or reject the certain request for the ride	TaxiManager