**Google API:**

It is an external Google component that provides the Mobile APP and On\_Board system some interfaces, makes the PowerEnjoy system able to get the data about information from the same database of Google maps, also the geographical location of points from the same database of Google+Local.

**Traffic system:**

It is an external component that provides the Account Manager some interfaces, makes the PowerEnjoy system to get data about drive license from the database of traffic system.

**Bank system：**

It is an external component that provides the Account Manager some interfaces, makes PowerEnjoy system able to get data about the credit card from the database of banks.

**Mobile APP:**

It is the component which embeds on the mobile devices of users, it offers user a GUI that

* Allows the users to access the functionality the system provide him
* Displays the data obtained by the interaction with *Mobile APP Facade*

And it charges not only the presentation behaviors but also the communication duty in order to deliver the tasks user required as well as the message, and the response from *Mobile App Facade.*

**On\_Board System:**  
 It is the component which embeds inside the panel of the car, it offers a GUI that:

Allows the users to access the functionality the system provide him to the car.

Displays the data obtained by the interaction with *On\_Board System Facade.*

And this component can also handle all communications with the *On\_Board System Facade* to accomplish the tasks. All messages from *On\_Board System* to the PowerEnjoy servers are also from this component.

**Mobile APP Facade**

This is a component implementing a *Facade Pattern:* this component offers the interfaces to accomplish the functionality that should be manipulated correspondingly by different managers. Only this facade has interface can be interact with the *Mobile App*.

**On\_Board System Facade**

This is a component implementing a *Facade Pattern:* this component offers the interfaces to accomplish the functionality that should be manipulated correspondingly by different managers. Only this facade has interface to can interact with the *On\_Board System*.

**Available Car Queue Manager**

This component is able to access to the *Data Layer* component to get the car information, and the manager updates the available cars as soon as the car state in *Data Layer* changes.

**Account Manager**

It is the component which support the functionalities about registration and log in. In order to do these, the Account Manager is able to :  
 Access the external Traffic component to verify the data;

Access the external Bank System to verify the data;

Access the Notification manager to exchange messages;

Access the Data Layer to store user data as well as check the history record.

**Reservation Manager**

This component handles reservation by:

Access the Data Layer component in order to check or change the state of reservation;

Make a countdown as well as get an reservation;

Access the *Payment Manager* only if the reservation fails;

Access the Data Layer component in order to change the car state as well as store reservation information;

Access the Ride Manager only if the user picks up the car successfully;

**Ride manager**

This component handles the car which is picked up successfully, it accesses to the *Reservation Manger*, commence the operation by the trigger of the message from *Reservation Manager,*and the *On\_Board System* is triggered at the same time. The Ride Manager access the *On\_Board System* and supply the functionality the user can get on aboard.

The *Ride Manager* accesses to the *Data Layer* at the end of the ride in order to update the car state as well as store the ride information,also accesses to the *payment manager to* transmit the payment information.

**Payment Manager**

It can receive the payment information by accessing the Reservation Manager and Ride Manager, after calculate the accurate money, the payment manager accesses to Bank system to get the money, and also accesses to the Data Layer to store the payment details.

**Notification Manager**

It can access to the Account Manager in order to exchange the massage between system and users.

**Optimal Path Calculator**

This component accesses to the On\_Board system in order to get the address information from uses, and also able to access both the Google API and Data Layer to calculate the optimal path from start point to destination.

**Data Layer**

This is the component that allows Reservation Manager, Account Manager, Available Queue Manager ,Ride Manager,Optimal Path Manager and Payment Manager to access the data which stored in database.

**DBMS**

It is a external component that is able to interact with the database where all the system data are stored via query.

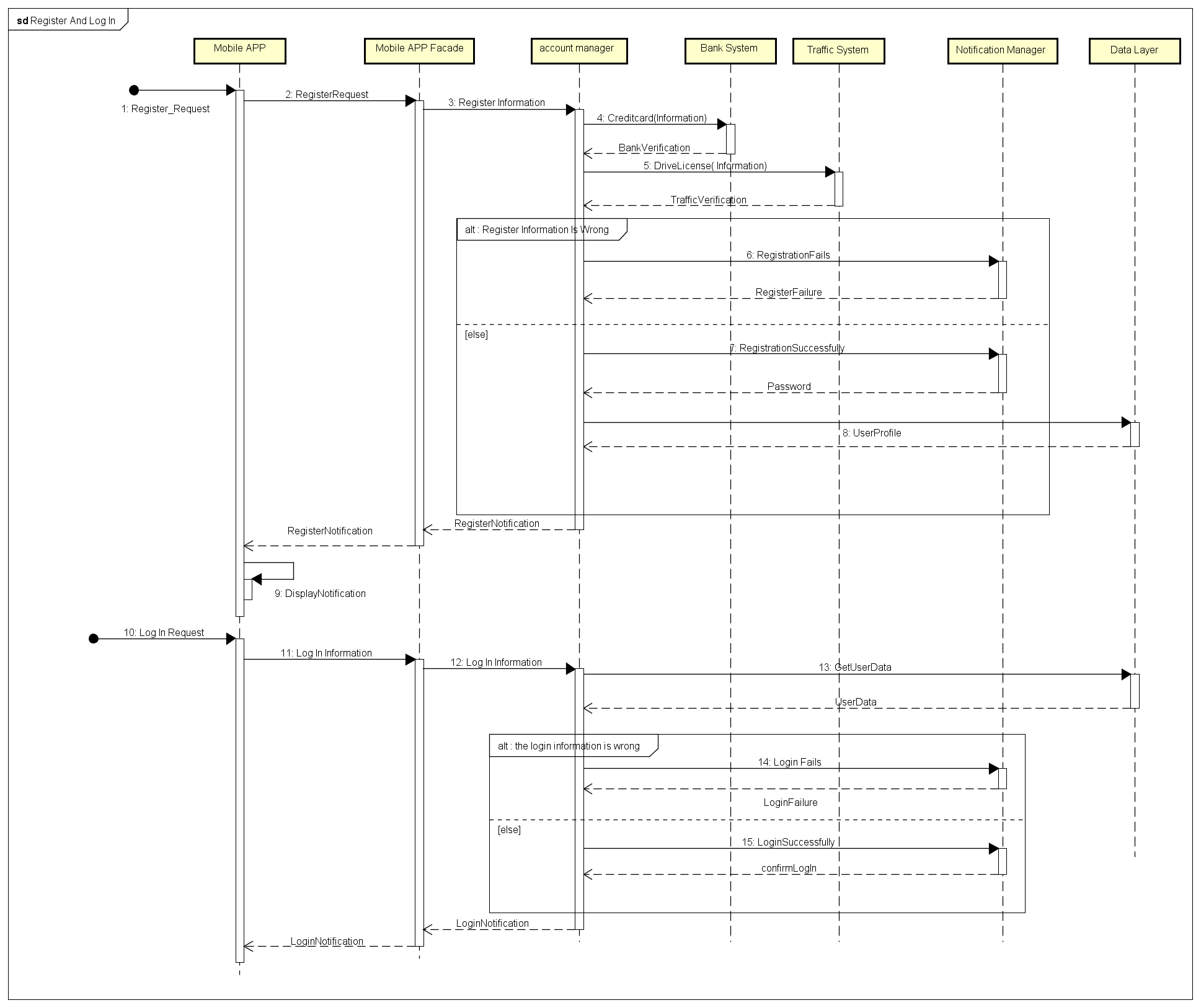
Runtime view

Register And Log In

The following (figure 2)sequence diagram shows us how the component interact with each other when the Mobile App get register and log in request from the interface.

Once guest accesses the Mobile APP and send a register request, the Mobile APP terminal gets the register information from the interface, then deliver it by remote server. The information flow through Mobile APP Facade and Account Manager, and be dispatched to the diverse external interfaces : traffic System and Bank System in order to verify the availability. These two external Interface both response a verification to account manager, the notification manager responses the different message corresponding to the verification result .If the guest register successfully, the register information will be stored in the DBMS. Note that the profile stored also include the password system generated, and the guest can decide by himself if he should launch a new round of register under the condition of registration failed.

When the Mobile APP terminal gets the log in request as well as the log in information, it deliver the information to Account Manager through the Mobile APP Facade .The Account Manager checks the correctness of Log In information, then the Notification Manager responses messages corresponding to the check result. Note that the user can decide by himself if he should launch a new round of log in request under the situation of log in failed.



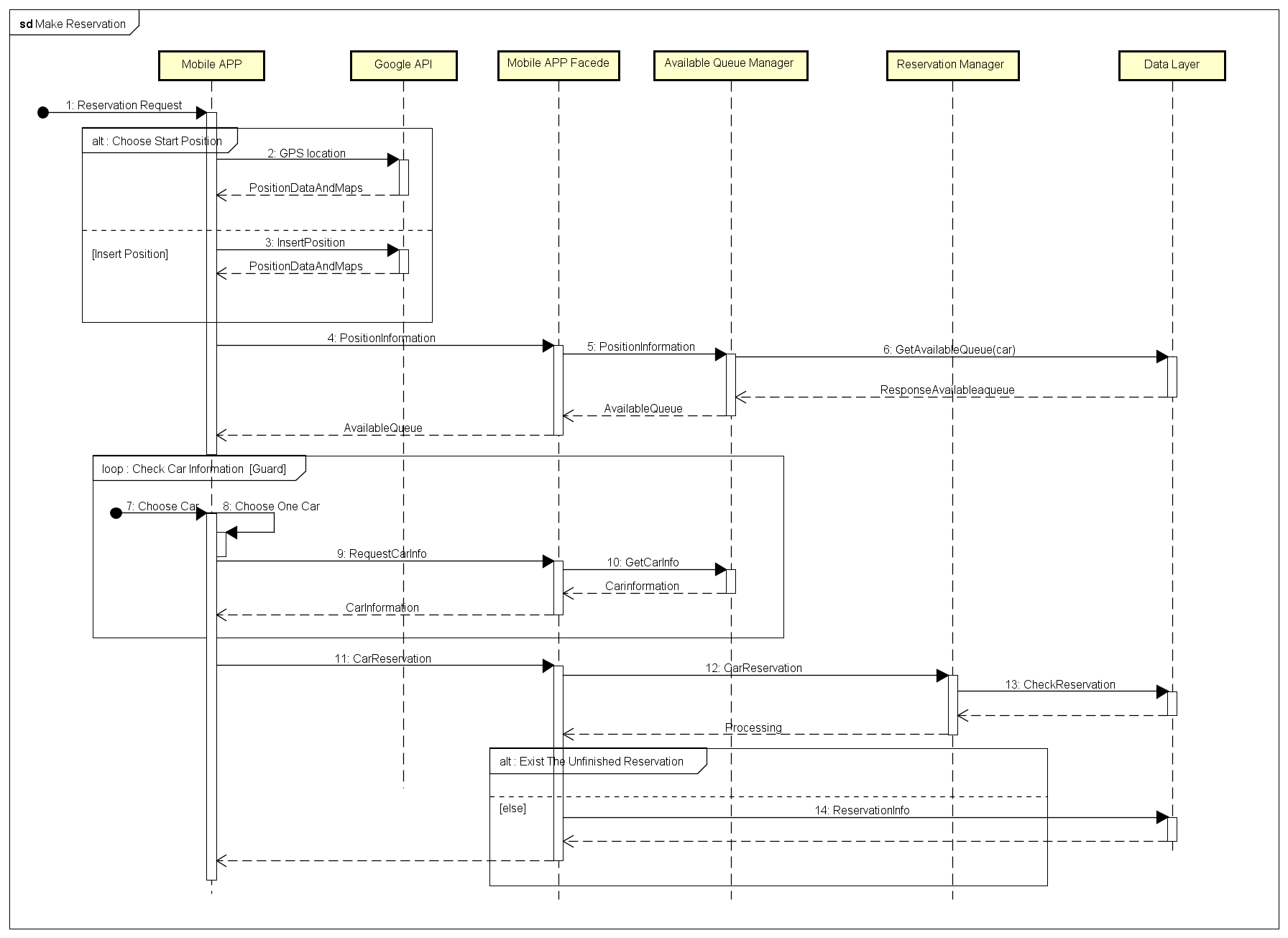
(fig 2)Register And Log In

Make An Reservation

In this sequence diagram (fig 3), it shows how components interact with each other to make a reservation from the Mobile APP side.

User supplies a start position,once the position be found, the Google API response the position data and map around the start position to Mobile APP terminal, and also deliver the position information to Mobile APP Facade ,which ignites the execution of searching an available queue of car.

The Available Queue Manager responses a queue to the Mobile App terminal so user can make reservation. The car will be removed from the available queue as long as it is reserved. The reservation data will be created in the DBMS as well.



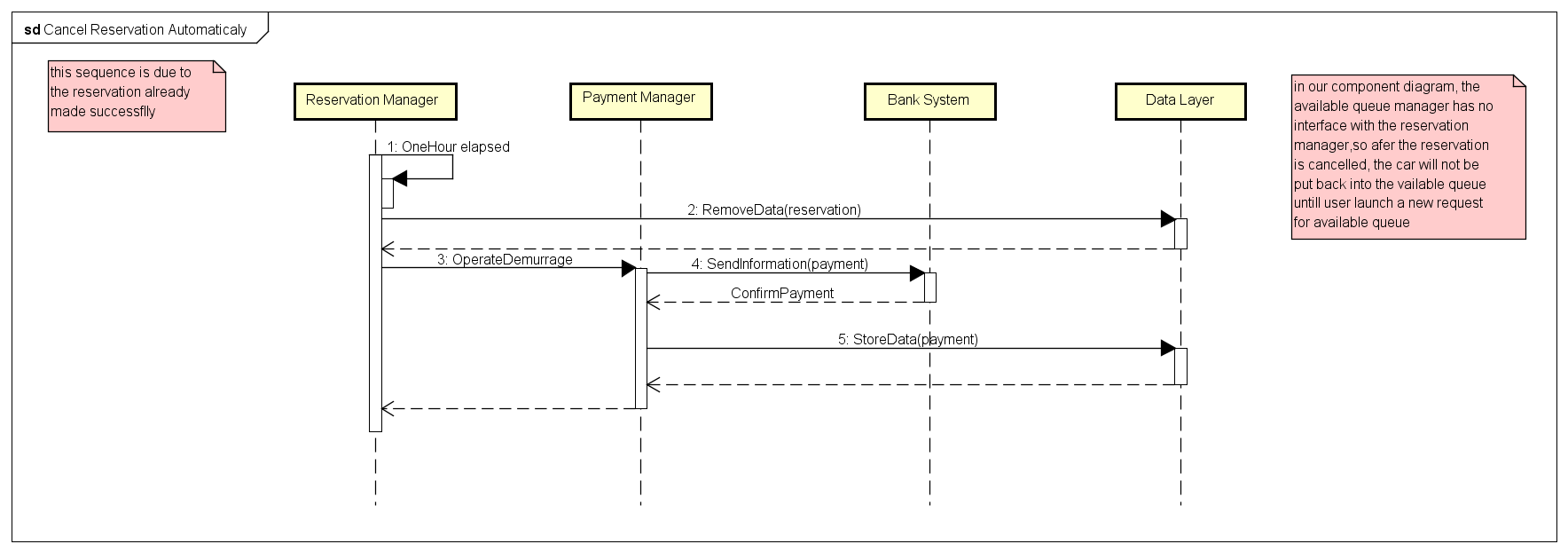
(fig 3) Make An Reservation

Cancel Reservation Automatically

In this sequence diagram(dig 4), we can see how the reservation be canceled automatically by system.

An one hour elapse can trigger the Reservation Manager to cancel the current reservation, the Payment Manager should also interact with Bank System consequently to deduce the money as the compensation fee.

Note that the user can get the notification of reservation canceled from the Mobile APP terminal.



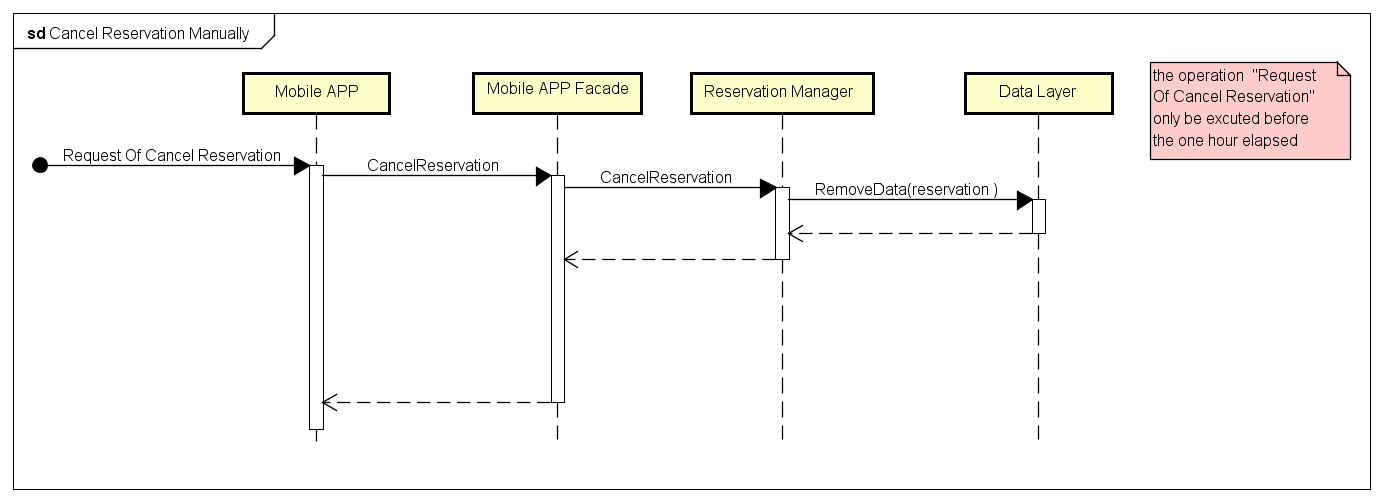
(fig 4) Cancel Reservation Automatically

Cancel Reservation Manually

This following sequence diagram(fig 5)tell us how the component work together to let user cancel the reservation from Mobile APP terminal.

The user launch a request to cancel the current reservation, the request is passed from remote server to the Reservation Manager,the Reservation Manager removes the corresponding data from DBMS and response the confirmation back.

Note that this event can only exists when the reservation lives less than one hour, and the available queue will be updated when user launched a research of available queue.



(fig 5)Cancel Reservation Manually

Complete Ride On Mobile APP Side

We can know from the following sequence diagram(fig 7) about how the components cooperate together to accomplish a complete ride on the Mobile APP side.

This operation is mainly focuses on the messages exchange and data exchange between component, the step is:

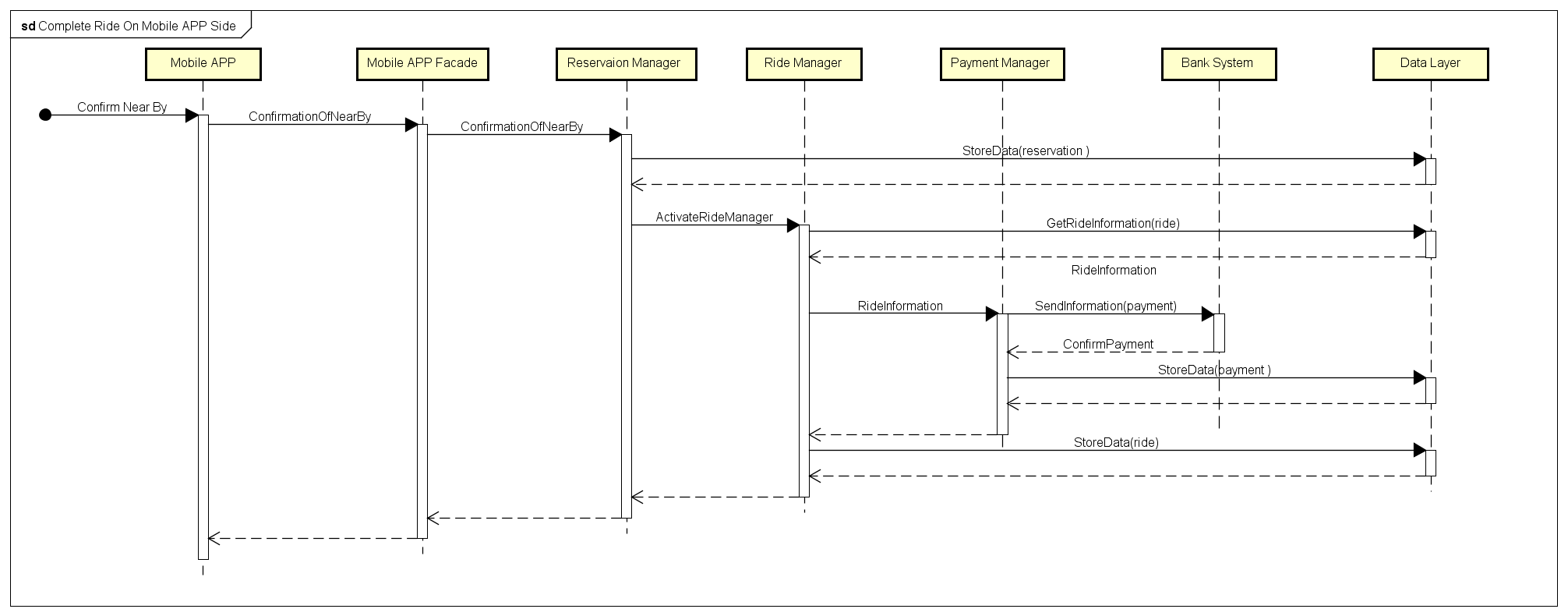
The user triggers the ride operation by a confirmation, the Mobile APP sends the confirmation by remote server, and activates the Reservation Manager. The reservation Manager ends current reservation a successful finished reservation, store the information into the database.

The Ride Manager is ignited, the Data Layer responses the ride information back as well as the complete ride finished, the Payment Manager will calculate the accurate money that cost in the current ride.

The Payment Manager sends the payment information to Bank System, the Bank System responses the confirmation of payment which signs the complement of payment. Then Payment Manager stores the payment data.

The Ride Manager stores the data, and the whole system response step-wisely.

Note that the ride operation is executed in the same time at both the Mobile APP side and the On\_Board System side, the ride is completely finished only if the executions finished in both side. These two sides have data communication which is not showed in this sequence diagram.



(fig 6)Complete Ride On Mobile APP Side

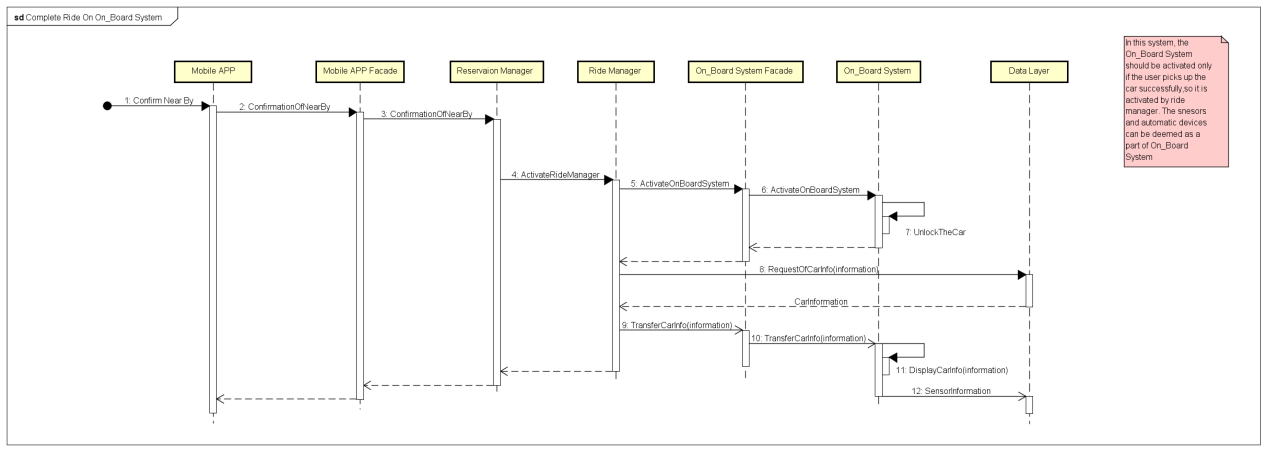
Complete Ride On On\_Board System

In the following sequence diagram(fig 7),we can see the ride operation details on the On\_Board System side.

The On\_Board System Facade is activated by the Ride Manger, then the On\_Board System is activated consequently. The On\_Board System terminal unlocks the car, and it signify the start of a ride.

The Ride Manager get the car information, then On\_Board System terminal displays the information on the screen. the On\_Board System also send the sensor information to the Data Layer asynchronously.

Note that the On\_Board System include all the sensor embedded into the car, the display screen, the GPS, also the autonomic devices like the lock in the door. The On\_Board System sends the car information real-timely to the Data Layer. The accomplishment of a complete ride is also gained from the sensors information.

 Complete Ride On On\_Board System

Save money Option

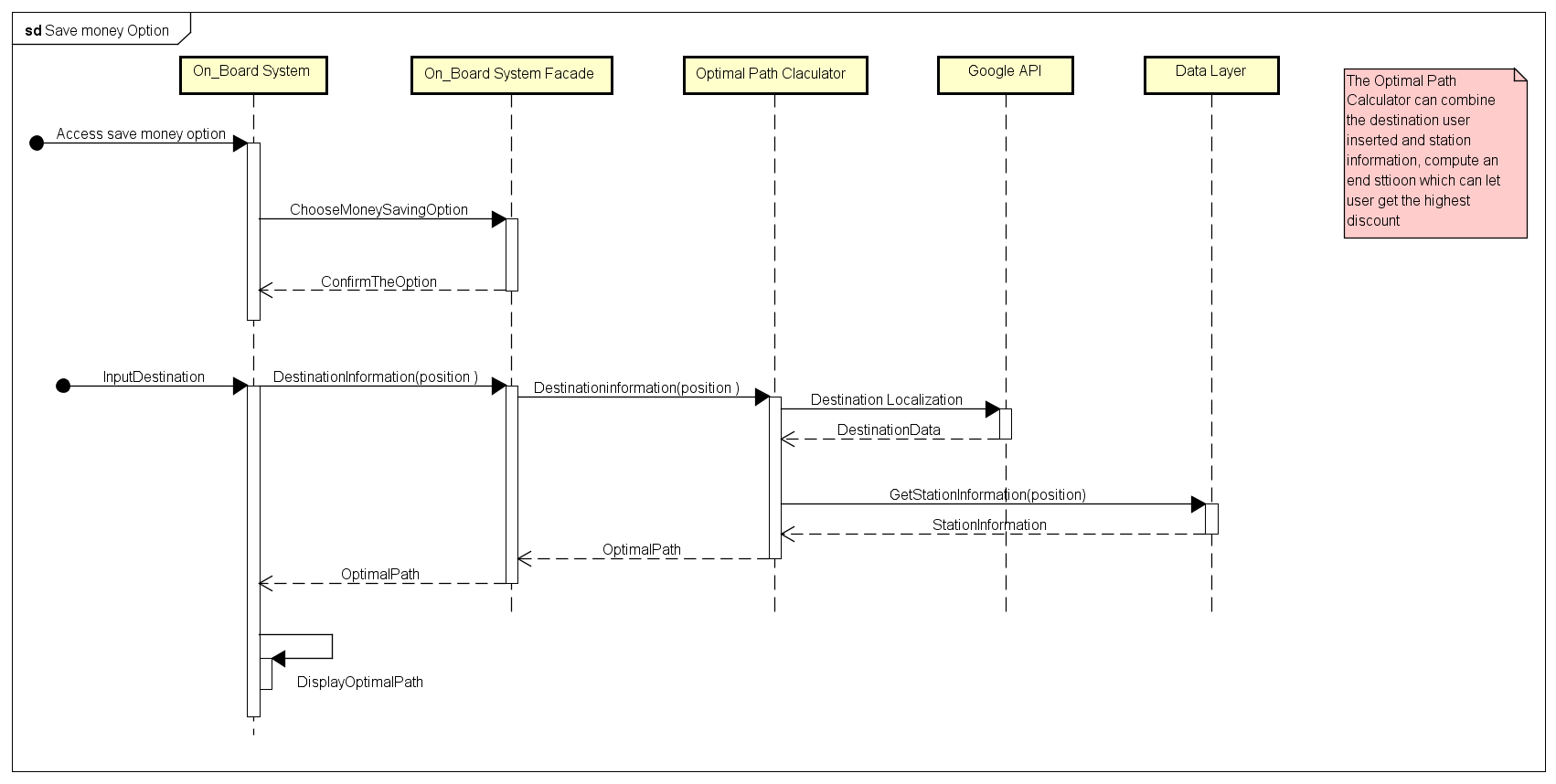
In the last sequence diagram (fig 8), we can know exactly how the component work together, supplies the way to users to access the saving money option.

The request is received from the interface of On\_Board System. When the On\_Board System confirm the request successfully, the On\_Board System will have the interface to get the destination information from user.

The On\_Board System facade sends the destination information to the Optimal Path Calculator, the Optimal Path Calculator gets the destination data from Google API, the station information from the Data Layer, and responses the optimal path to the fittest station.

The optimal path is returned to the On\_Board system and be showed on the screen.

Note that the station information also includes the information of plugs sensor in the power station, so that the Optimal Path Calculator can also consider the free plugs in that power station.



Save money Option

**3 Algorithm design**

For a clearer description about our project, we clarify the algorithm about queue management, pay management and money saving option, which we think they are important aspects of development of application.

**3.1 Queue management**

This algorithm needs the position information about all the available cars in the city, as well as the accurate start position information from the user. Every time user launches a request for the available cars,the Available Queue Manger gets the coordinates of available cars in the city, then it starts the research of an available queue of cars around the start position which insert by user or oriented by GPS.

cars in the queue will be updated when a new request of searching is launched. It can be added with the cars which is released from a ride or a reservation, it can also be removed by reservation.

Cars in the available queue are enumerated in the increase order in “distance to the start position”and the decrease order both in dump energy of car. The available car which is far than 3 KM are be consider as unreachable cars, and won’t be enumerated in the available queue.

The research is done in according to the algorithm bellow, written in pseudo code:

Function searchAvailableQueue

cars(array)

destination(GPS coordinate)

while Distance(car.location,start)<=3

Sort(

cars as car;

by Distance(car.location,start)\*10000000+ (100-car.battery)

)

return cars;

End function

**3.2 Pay management**

This algorithm needs the sensor information about both the car and power station. When a ride finished, the system sends all the sensor information to the Payment Manager in order to compute the money user should pay in this ride.

The computation is done in according to the algorithm bellow, written in pseudo code:

Function PaymentCalculate

Sensor(array)

Discount1:=0.8

Discount2:=0.7

Compensation:=1.3

ComputeFinalPrice(

price:=Sensor.rideTime\*moneyPerMinute

numPassengers:=Sensor.passenger

dumEnery:=Sensor.battery

if( numPassengers>2)

price:=price\*Discount1

if( dumEnergy/fullBattery>0.5)

price:=price\*Discount1

if( dumEnergy/fullBattery<0.3)

price:=price\*Compensation

if( numPassengers>2)

price:=price\*Discount

if(Sensor.plug)

price:=price\*Discount2

if(( Distance(car.location,powerStation))>3)

price:=price\*Compensation

)

return price

End

**3.3 money saving calculating**

This algorithm is used for calculate the optimal station near the destination which user can get the height discount. It needs the destination from On\_Board System, also the distribution of power station.The algorithm consider both the distance and the accessibility of power plugs, return user the optimal station not only reduce the cost of user but also contribute the cars in a reasonable way.

powerStation(array)

destination(GPS coordinate)

function moneySavingOption(powerStation,destination)

suggestedStation<-none

currentDistance<-infinite

for ps in powerStations

distance<-distance(destination,ps.location)

if ps.numPowerPlugs>0 and (

distance<currentDistance or(

distance=currentDistance

and suggestedStation.numPowerPlugs<ps.numPowerPlugs

)

)

then

suggestedStation<-ps

currentDistance<-distance

return suggestedStation

end