RASD
Requirement Analysis & Specification Document

AUTHORS:

_GIANCARLO COLACI

_GIULIO DE PASQUALE

_FRANCESCO RINALDI

PROJECT:

_POWERENJOY

SUPERVISOR:

_ELISABETTA DI NITTO

The Document

This document belongs the first phase of the **Waterfall Model** of **Software Development**: the **Requirement** Phase. It is very importante because errors made in understanding requirements have the potential for greatest cost, because many other design decisions depend on them.

[The cost of correcting an error depends on the number of **subsequent decisions** that are based on it.]

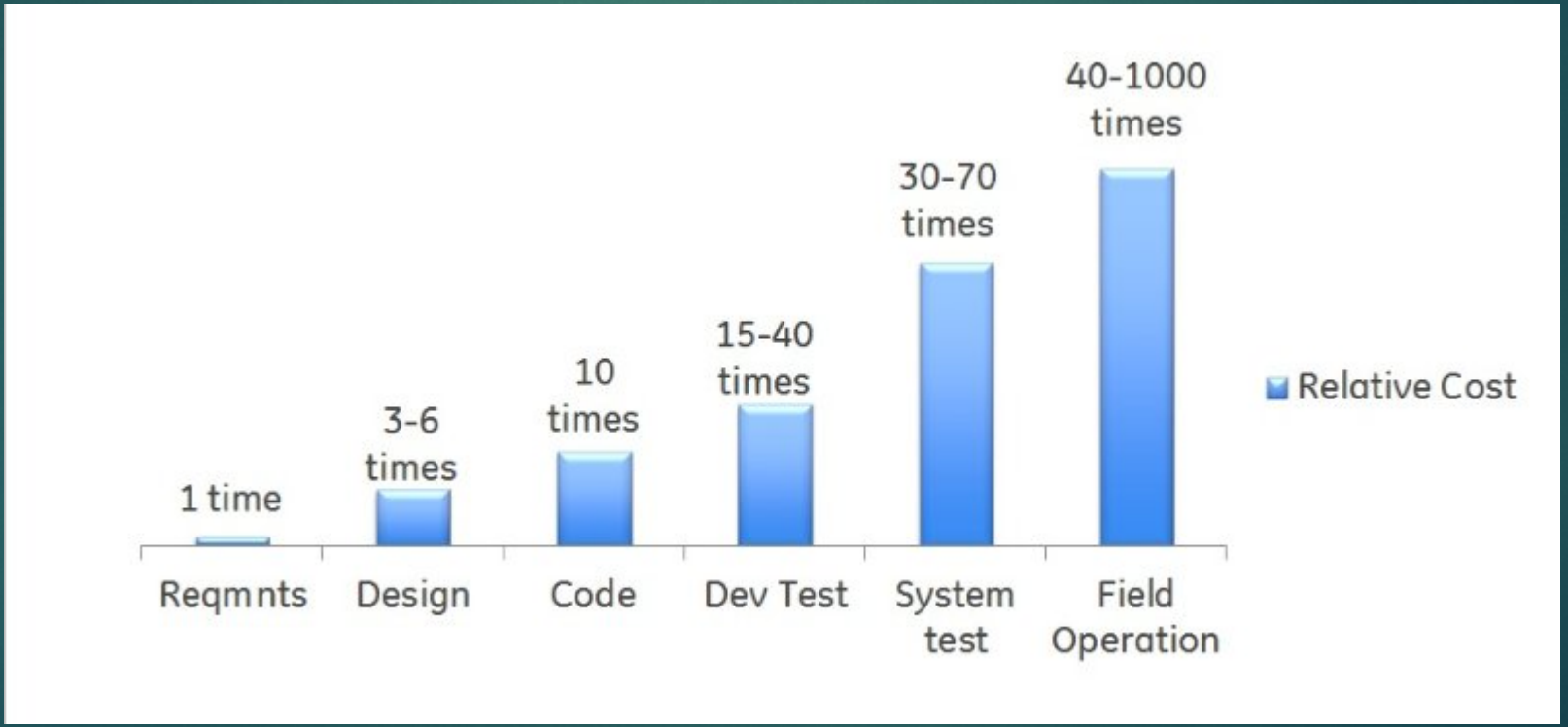


Illustration of the relative impact of requirement errors.

Actors Identifying

- **Guest:** a person that has never signed up to the system. He can only sign up, view available cars and contact the customer service.
- **User:** a person already registered in the system who can log in and use all the functionalities described. Unless specified, each user is active: therefore he / she can use the service with no restrictions.
- **Deactivated user:** a user with revoked privileges. He/she cannot use the service until the issues with his/her account are solved. (e.g., expired license)

Domain Properties

- The user will only provide correct and authentic information while signing up.
- The user will never try to exploit the system's features (e.g. sleep in the car).
- The user will never fake his/her position to cause a denial of service.
- The user will request help only if he/she needs it.
- The user will never let another person drive the car he/she reserved.
- The user will never be robbed of the access credentials and the verification code.
- The user will agree and respect our terms of use (e.g. to be geolocalized)

Glossary:

Verification code: it is a code that adds another level of security to our service: the user will need it to unlock the car.

Domain Properties

- Each car has an embedded key to turn on the engine;
- Each available car is fully functional thanks to the **MES**;
- Each car is equipped with a properly wortking **ADS** and an **ECS**;
- Each car is fitted with a properly working notification touchscreen display;
- Each seat is equipped with a sensor which is used to detect the number of passengers into each car;
- An available car will always be found in the supposed position.

Glossary:

ADS: Auto **D**iagnosis **S**ystem, an always on embedded peripheral which continuously monitors the status of the car (e.g. battery charge, tires pressure, impact detection)
ECS: Emergency **C**all **S**ystem, an always on embedded peripheral which can be used to call the consumer service or an emergency number quickly
MES: Maintenance **E**xternal **S**ervice, it is an external service that takes care of ordinary or extraordinary car maintenance.

Assumptions

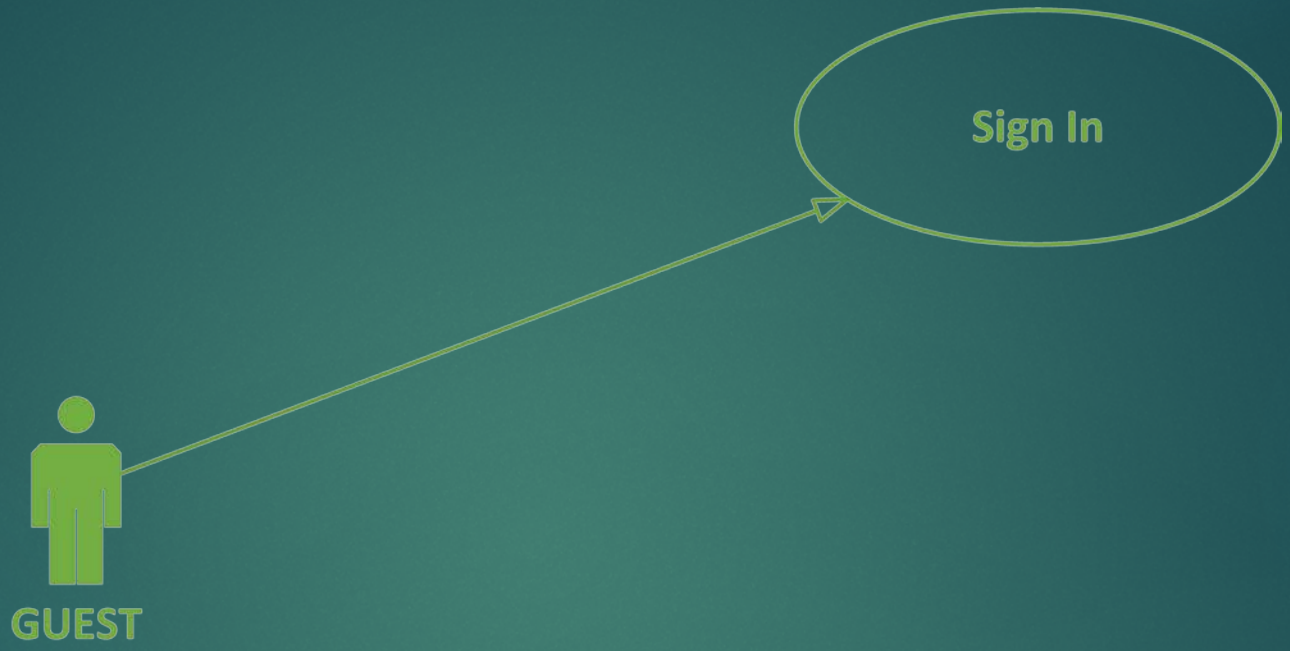
Due to some unclear points in the specification we made some **assumptions**, which are:

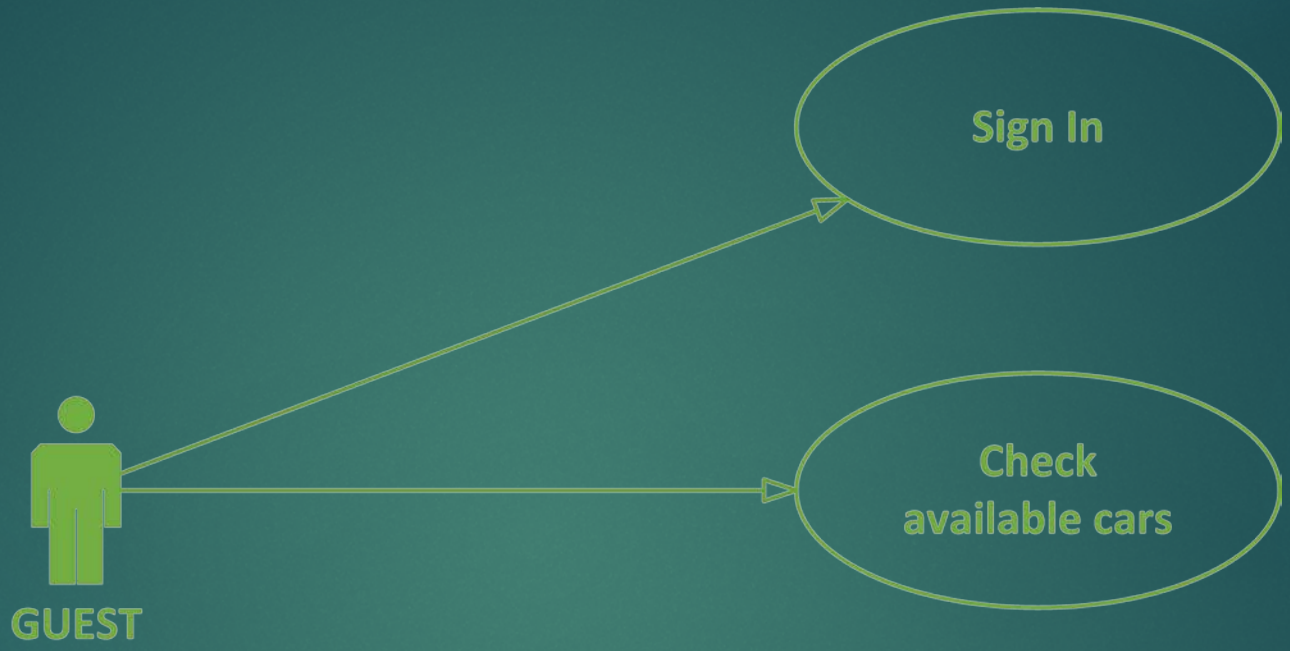
- The system allows users to locate any car in PowerEnjoy's area of operation;
- The system's Money Saving Option has to be enabled or disabled during registration or in the user profile;
- The system permits users to reserve an available car only in his/her city: with "a certain geographical region" we mean that the user can reach the vehicle in a reasonable amount of time;
- The system can locate its users in some way: for example, through Cellular Data or GPS;
- The system can check the proximity of the user to the reserved car: for example, through a numerical code printed on each cars' windshield or through GPS data from user's phone;
- The rental fees are applied as soon as one of these conditions is met: one minute is elapsed from the car's doors opening, or the engine is revved; (otherwise the user could stay within the car for free)
- Each user can't reserve more than one car at the same time;
- To be eligible for the 10% discount, almost two passengers have to remain seated at the same time for at least one minute;
- If the user will leave the car in a special recharging area, taking care of plugging it into the power grid, the discount will be greater than 30%: for example, we suppose 40%.

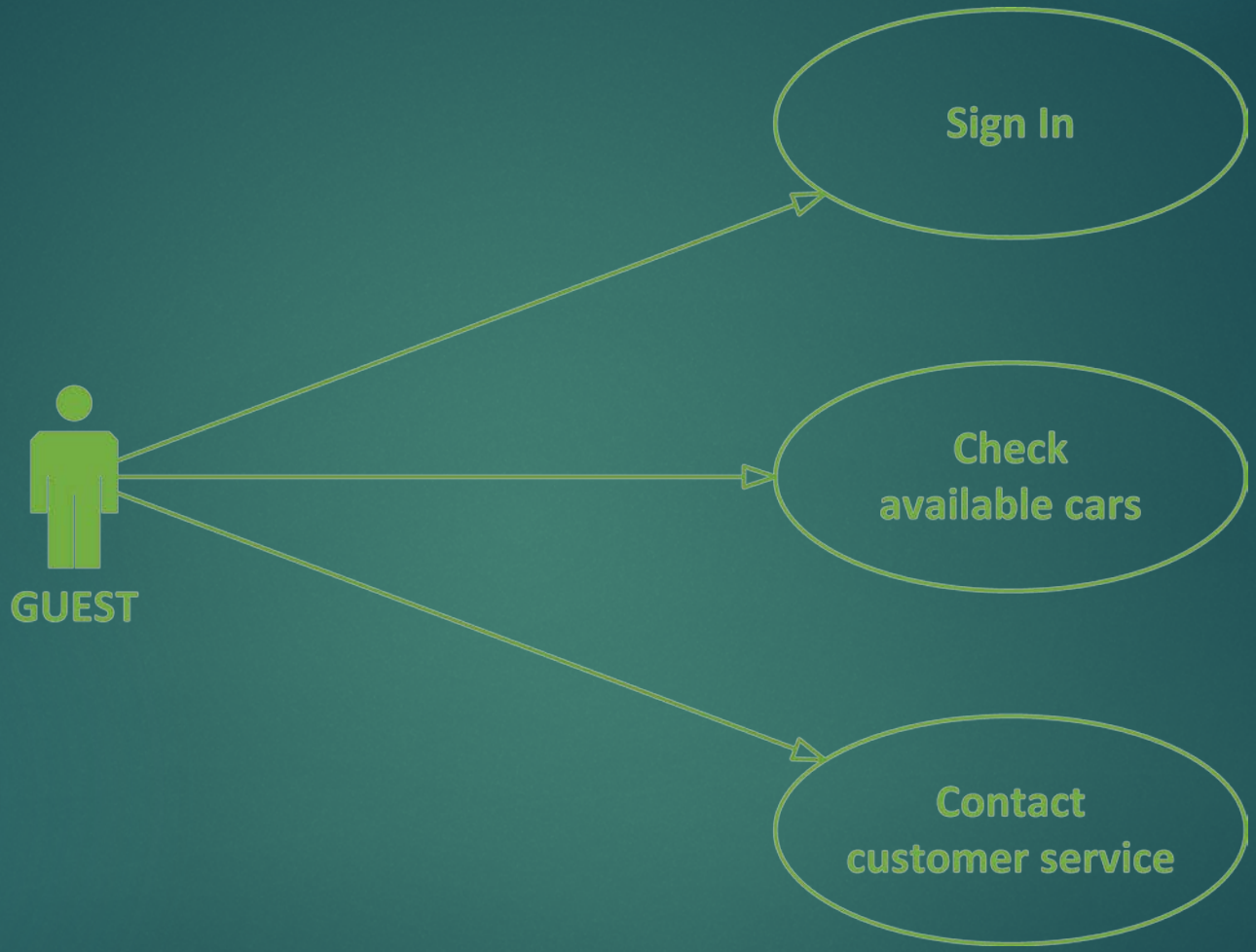
Privileges: Guest

- Register to the system by creating a new account;
- Check available cars' position and status;
- Contact the customer service.


GUEST


Sign In
GUEST


Sign In
Check
available cars
GUEST

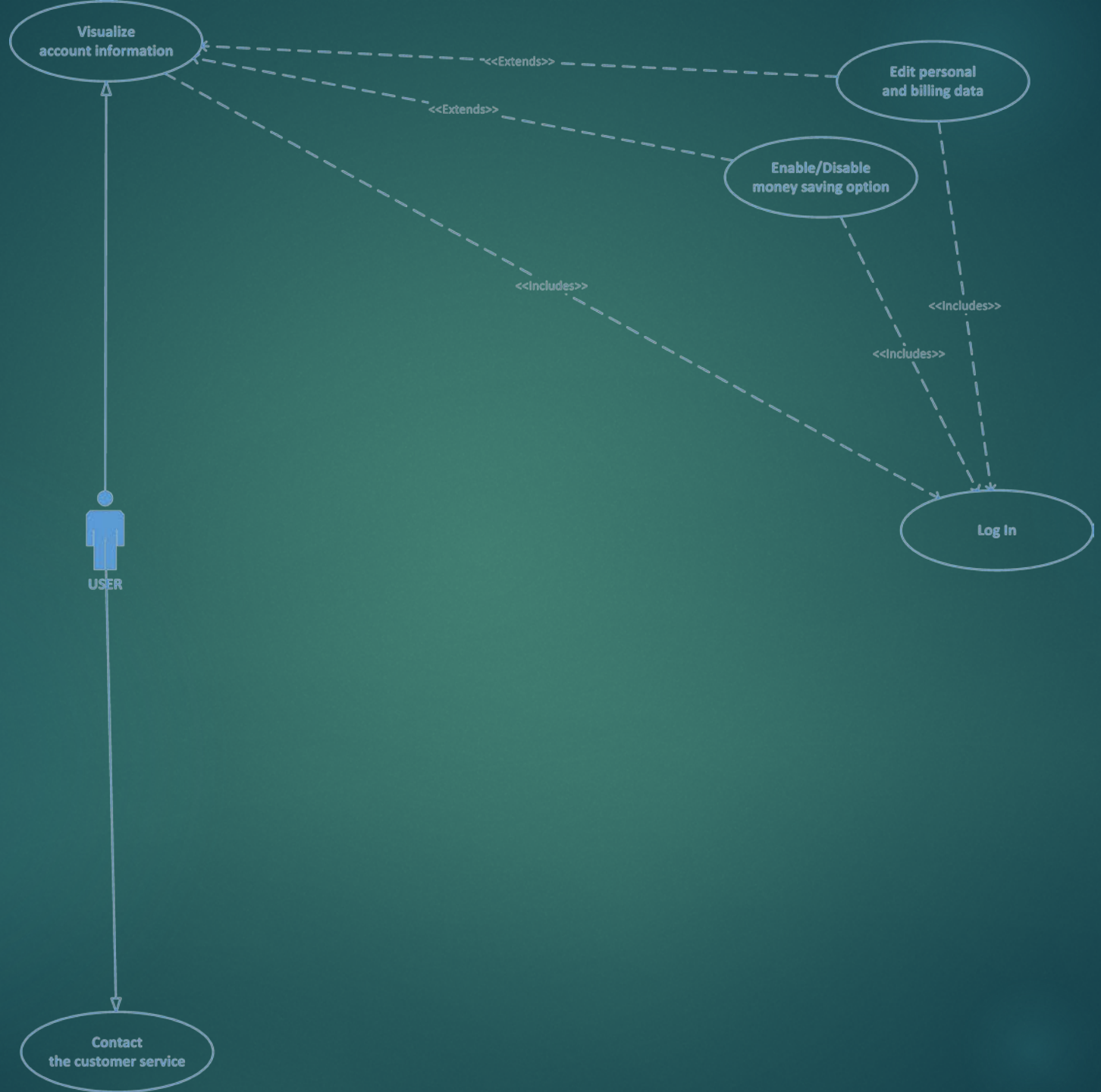

Sign In
Check
available cars
Contact
customer service
GUEST

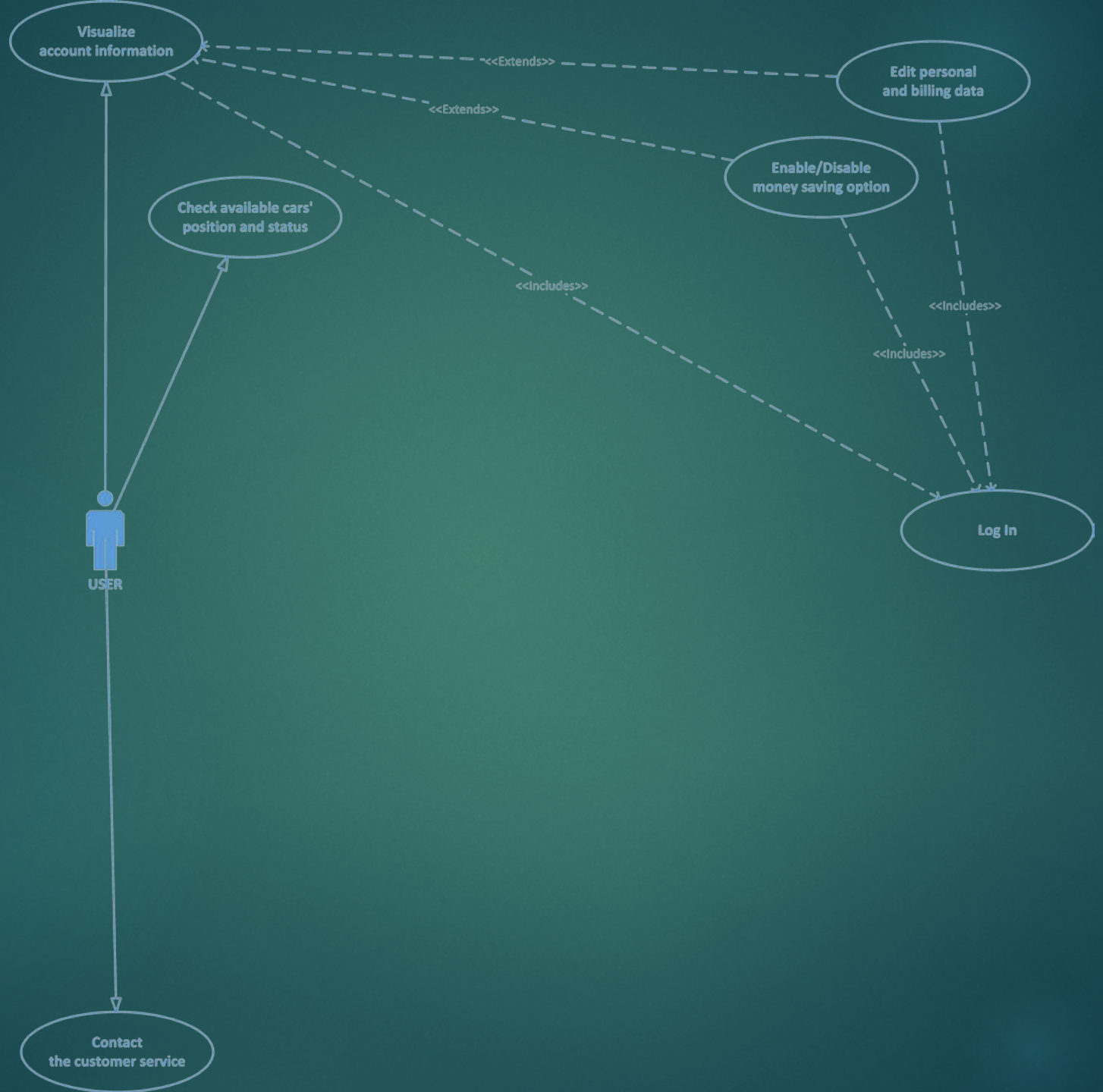
Privileges: User

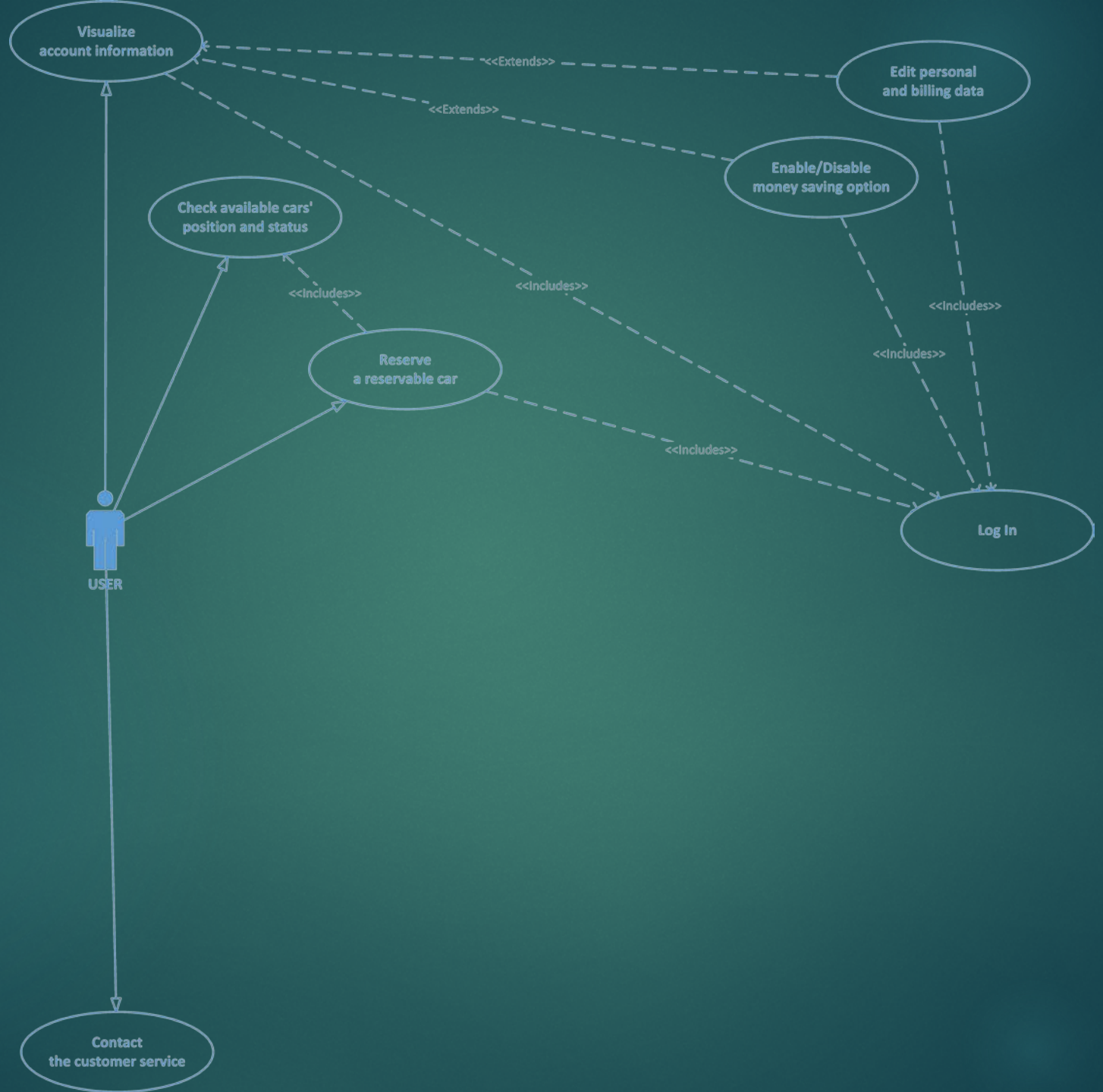
- Log into the system;
- Consult reservations' history;
- Edit account information:
  - personal and billing data;
  - enable/disable Money Saving Option;
- Check available cars' position and status;
- Reserve a reservable car;
- Rent the reserved car;
- Check active reservation status:
  - Remaining time until the reservation expires;
  - Reserved car's position and status;
  - Elapsed rental time;
- Terminate the rent;
- Contact the customer service.

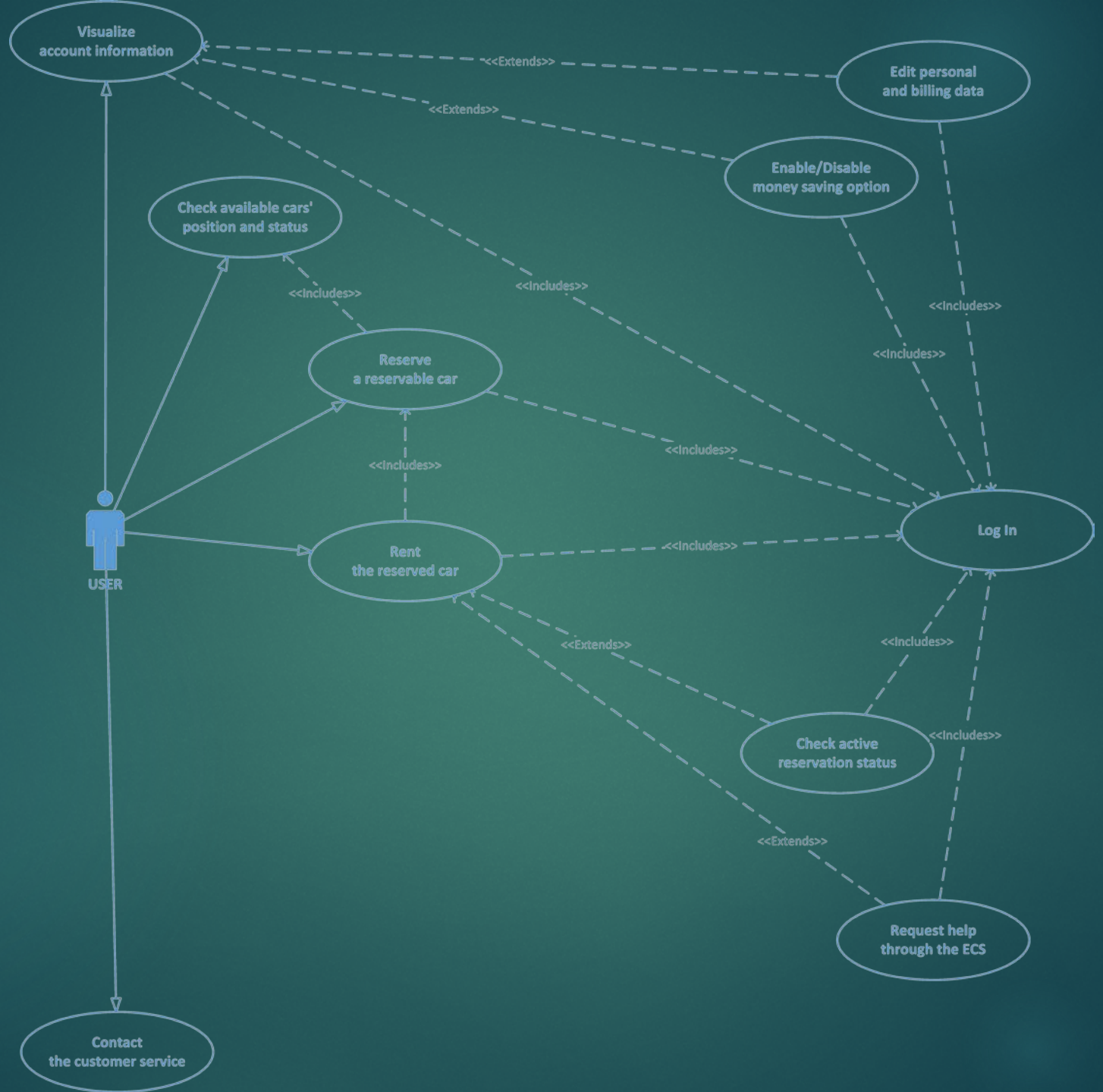

USER

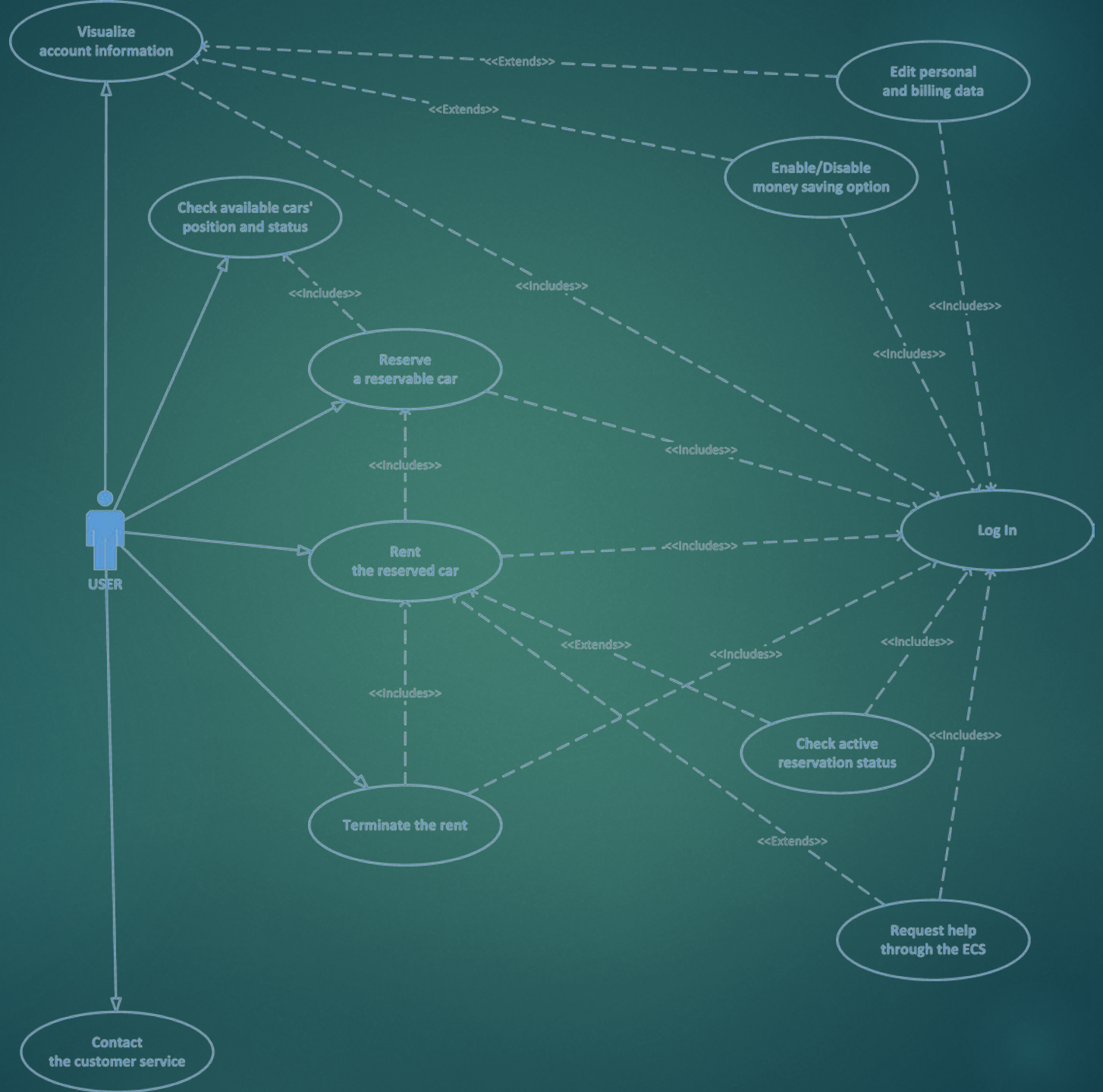

USER
Contact
the customer service

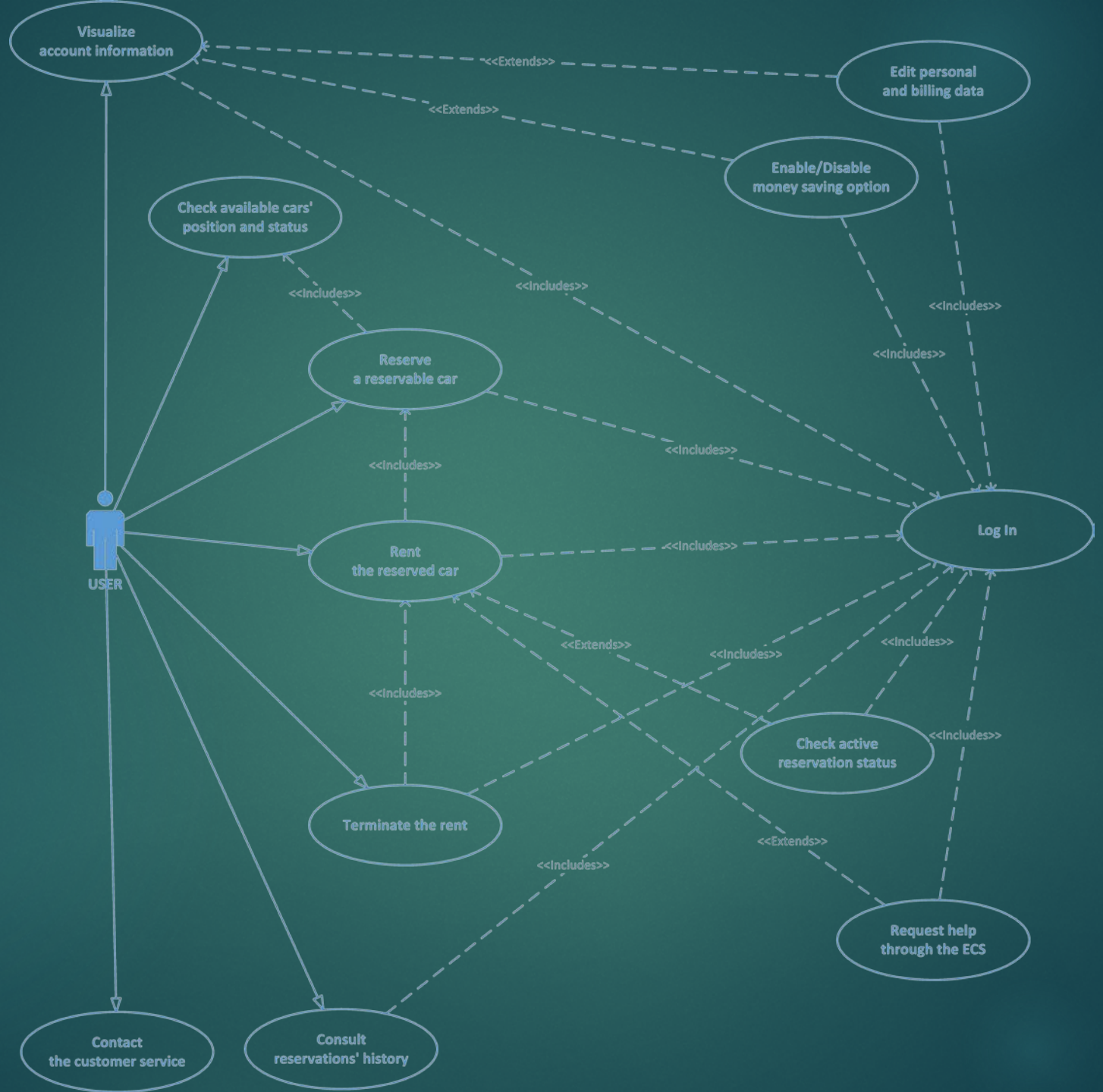

Visualize
account information
<<Extends>>
Edit personal
and billing data
<<Extends>>
Enable/Disable
money saving option
<<Includes>>
<<Includes>>
<<Includes>>
Log In
USER
Contact
the customer service


Visualize
account information
<<Extends>>
Edit personal
and billing data
<<Extends>>
Enable/Disable
money saving option
Check available cars'
position and status
<<Includes>>
<<Includes>>
<<Includes>>
Log In
USER
Contact
the customer service


Visualize
account information
<<Extends>>
Edit personal
and billing data
<<Extends>>
Enable/Disable
money saving option
Check available cars'
position and status
<<Includes>>
<<Includes>>
<<Includes>>
<<Includes>>
Reserve
a reservable car
<<Includes>>
Log In
USER
Contact
the customer service


Visualize account information
<<Extends>>
Edit personal and billing data
<<Extends>>
Enable/Disable money saving option
Check available cars' position and status
<<Includes>>
<<Includes>>
<<Includes>>
<<Includes>>
Reserve a reservable car
<<Includes>>
<<Includes>>
Log In
USER
Rent the reserved car
<<Includes>>
<<Extends>>
<<Includes>>
Check active reservation status
<<Includes>>
<<Extends>>
Request help through the ECS
Contact the customer service

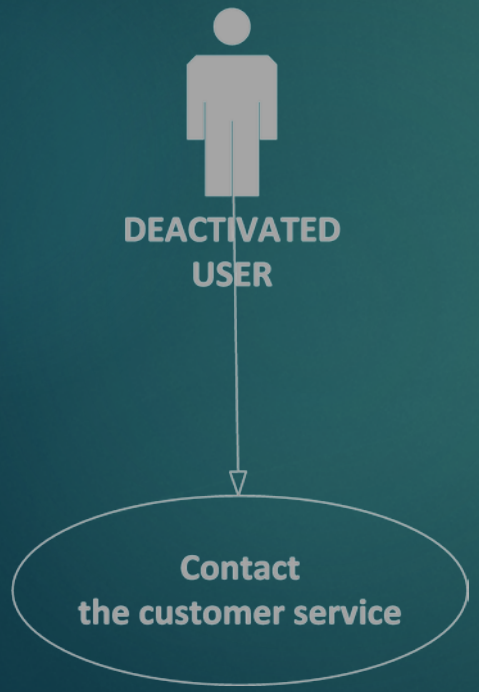

Visualize account information
<<Extends>>
Edit personal and billing data
<<Extends>>
Enable/Disable money saving option
Check available cars' position and status
<<Includes>>
<<Includes>>
<<Includes>>
<<Includes>>
Reserve a reservable car
<<Includes>>
<<Includes>>
Log In
USER
Rent the reserved car
<<Includes>>
<<Extends>>
<<Includes>>
<<Includes>>
<<Includes>>
Check active reservation status
<<Includes>>
Terminate the rent
<<Extends>>
Request help through the ECS
Contact the customer service

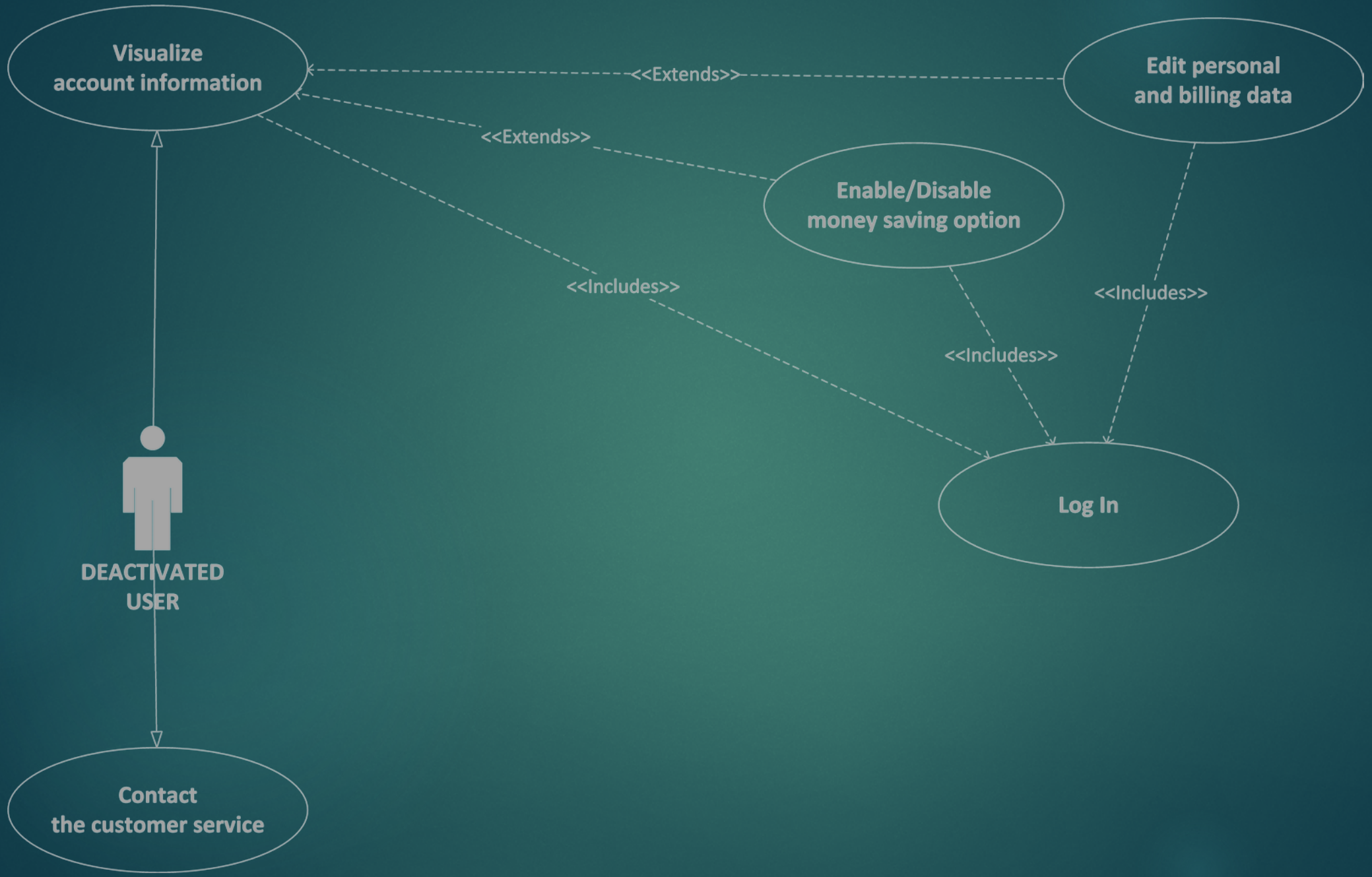

Visualize account information
<<Extends>>
Edit personal and billing data
<<Extends>>
Enable/Disable money saving option
Check available cars' position and status
<<Includes>>
<<Includes>>
<<Includes>>
<<Includes>>
Reserve a reservable car
<<Includes>>
<<Includes>>
Log In
USER
Rent the reserved car
<<Includes>>
<<Extends>>
<<Includes>>
<<Includes>>
<<Includes>>
Check active reservation status
<<Includes>>
Terminate the rent
<<Extends>>
<<Includes>>
Request help through the ECS
Contact the customer service
Consult reservations' history

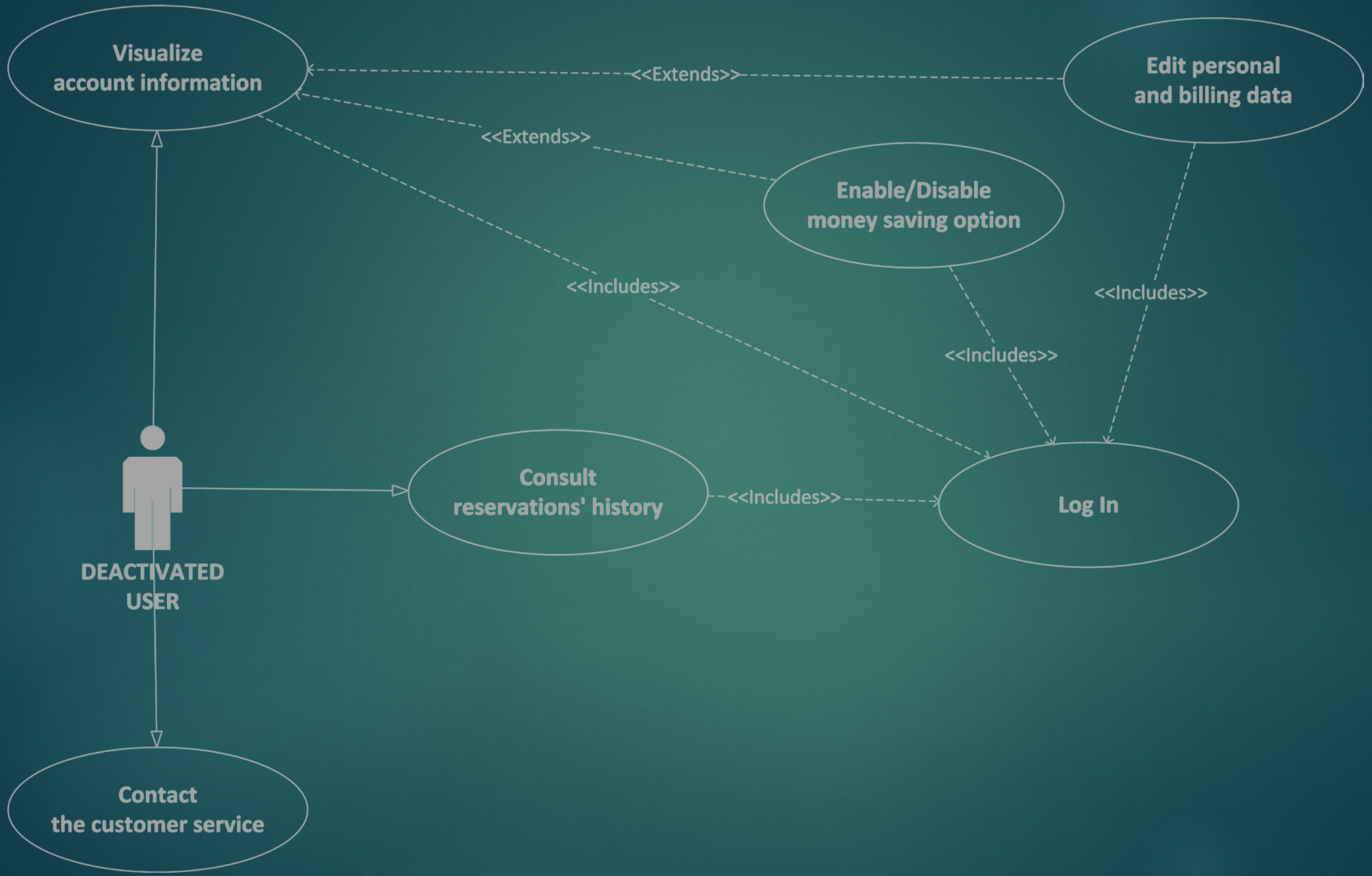
Privileges: Deactivated User

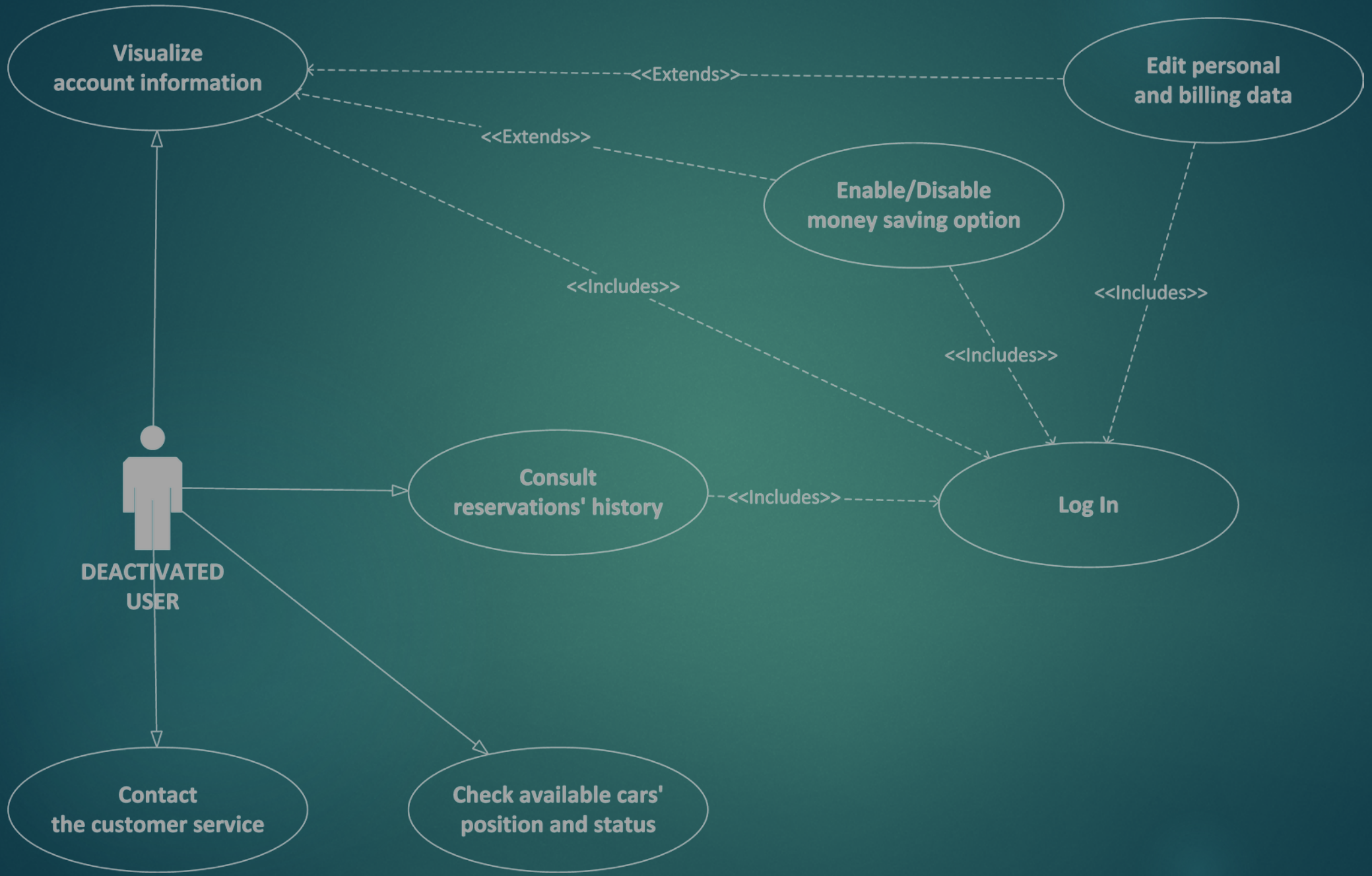
- Log into the system;
- Consult reservations' history;
- Edit account information:
  - personal and billing data;
  - enable/disable Money Saving Option;
- Check available cars' position and status;
- Reserve a reservable car;
- Rent the reserved car;
- Check active reservation status:
  - Remaining time until the reservation expires;
  - Reserved car's position and status;
  - Elapsed rental time;
- Terminate the rent;
- Contact the customer service.

DEACTIVATED
USER


DEACTIVATED
USER
Contact
the customer service

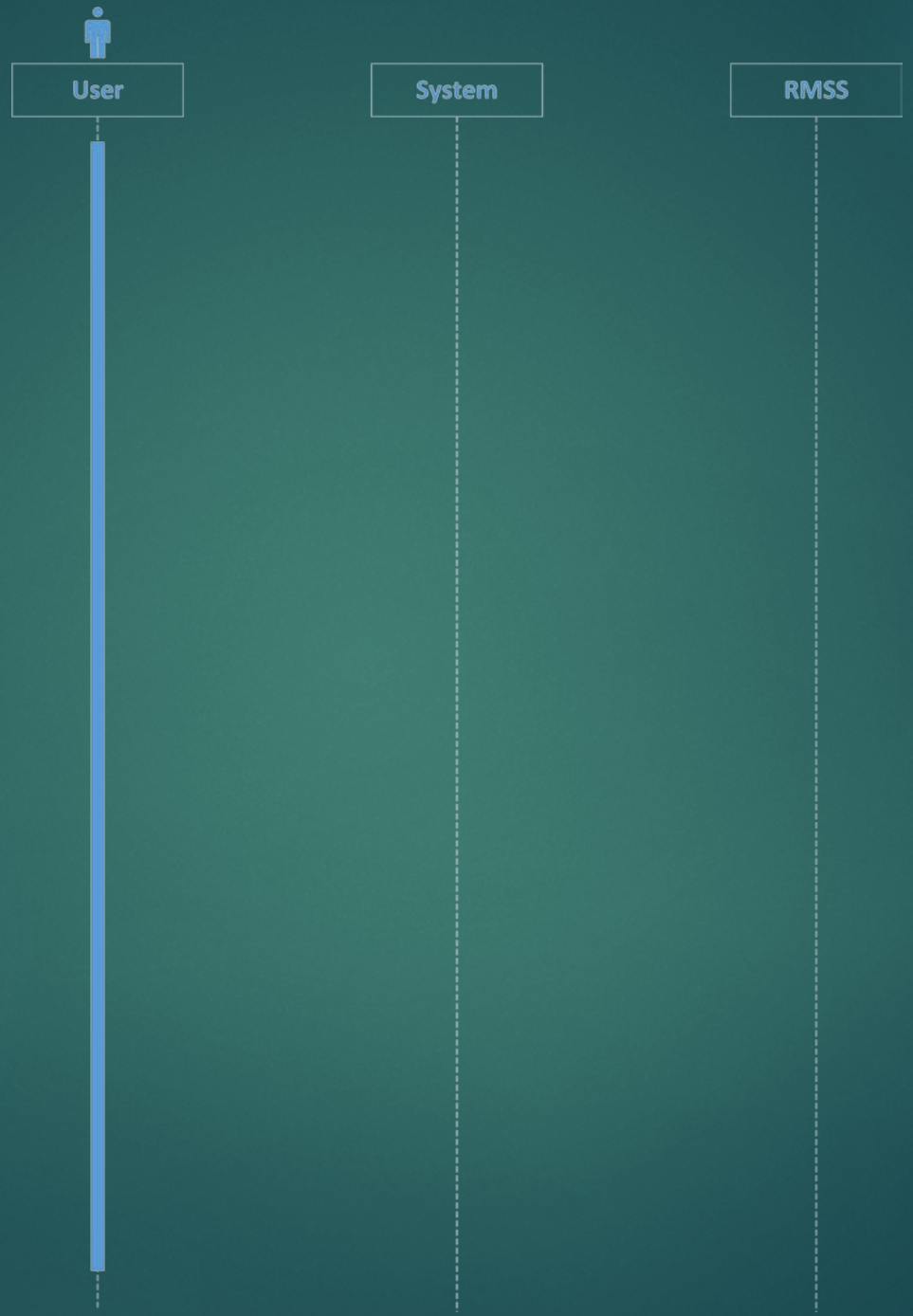

Visualize
account information
<<Extends>>
Edit personal
and billing data
<<Extends>>
Enable/Disable
money saving option
<<Includes>>
<<Includes>>
<<Includes>>
Log In
DEACTIVATED
USER
Contact
the customer service

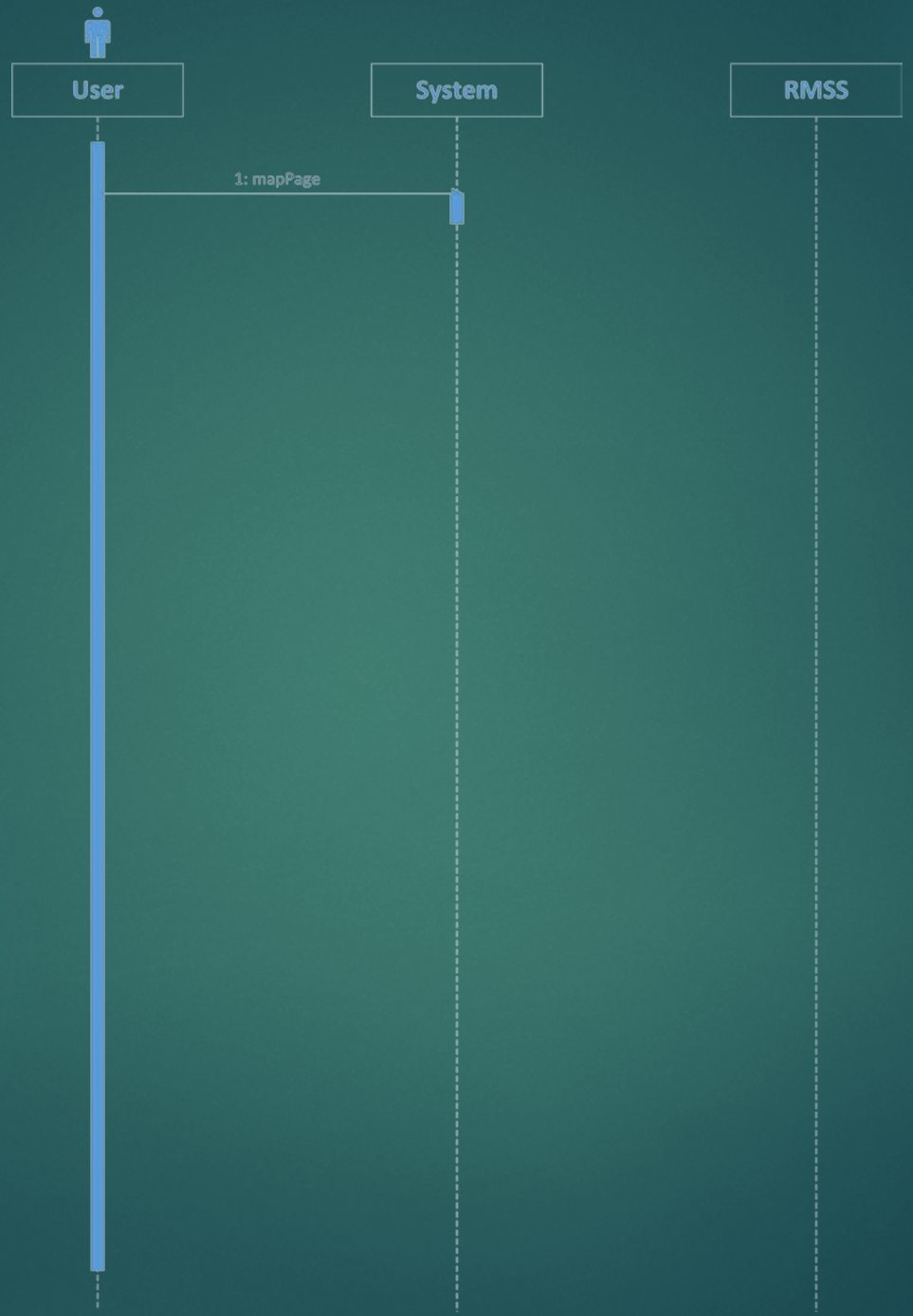

Visualize account information
<<Extends>>
Edit personal and billing data
<<Extends>>
Enable/Disable money saving option
<<Includes>>
<<Includes>>
<<Includes>>
DEACTIVATED USER
Consult reservations' history
<<Includes>>
Log In
Contact the customer service

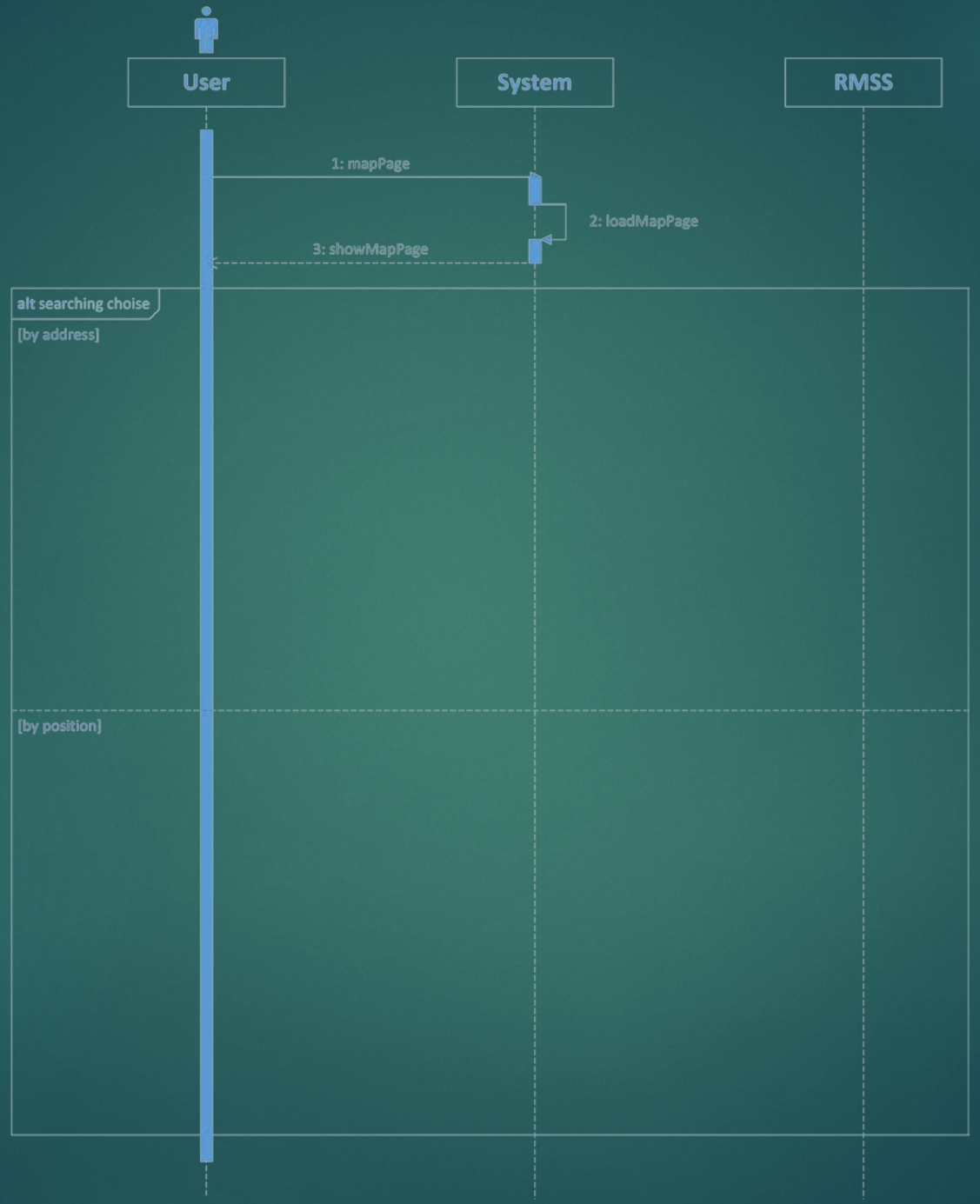

Visualize account information
<<Extends>>
Edit personal and billing data
<<Extends>>
Enable/Disable money saving option
<<Includes>>
<<Includes>>
<<Includes>>
DEACTIVATED USER
Consult reservations' history
<<Includes>>
Log In
Contact the customer service
Check available cars' position and status

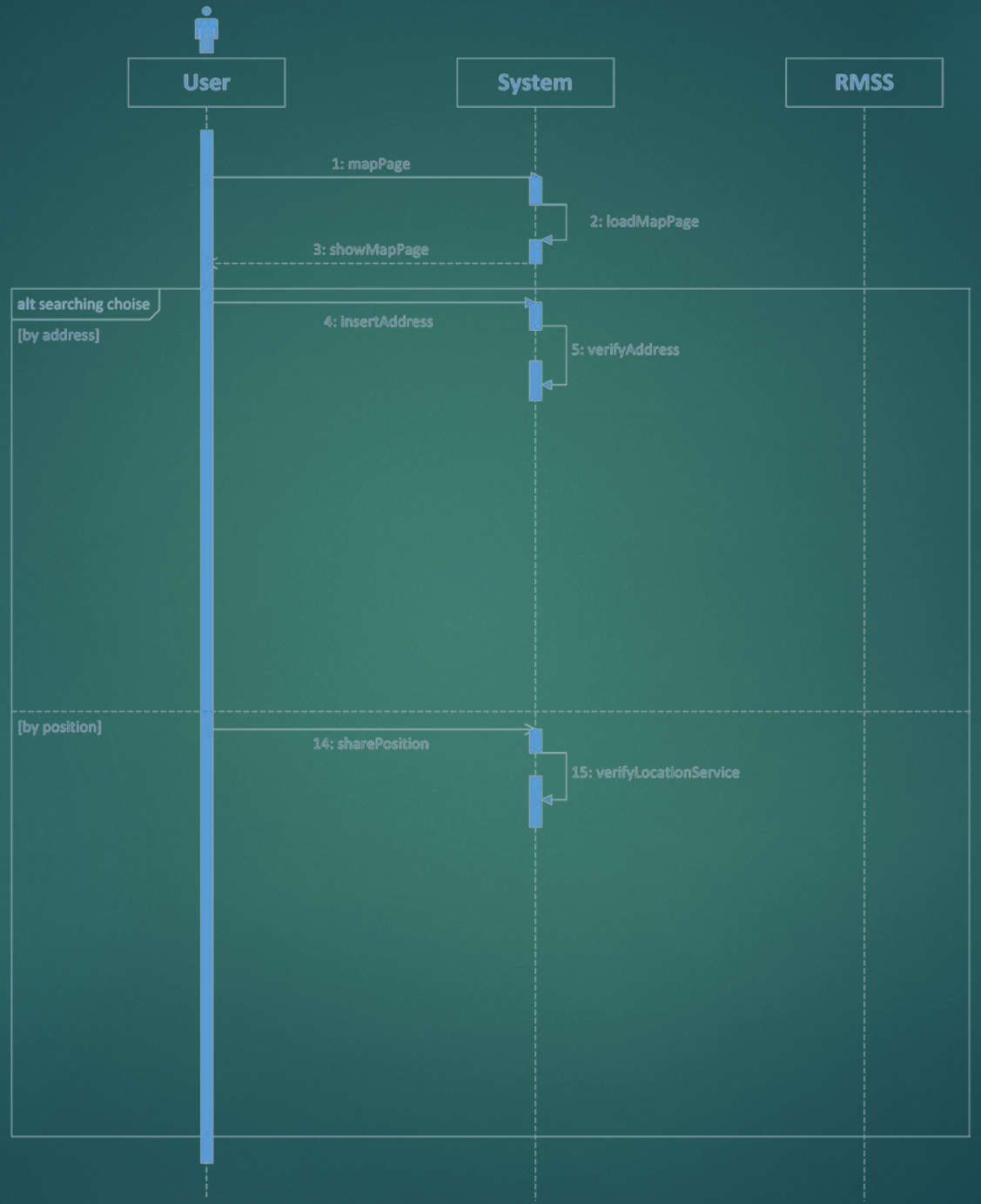
5.1.5 Check available cars' position and status

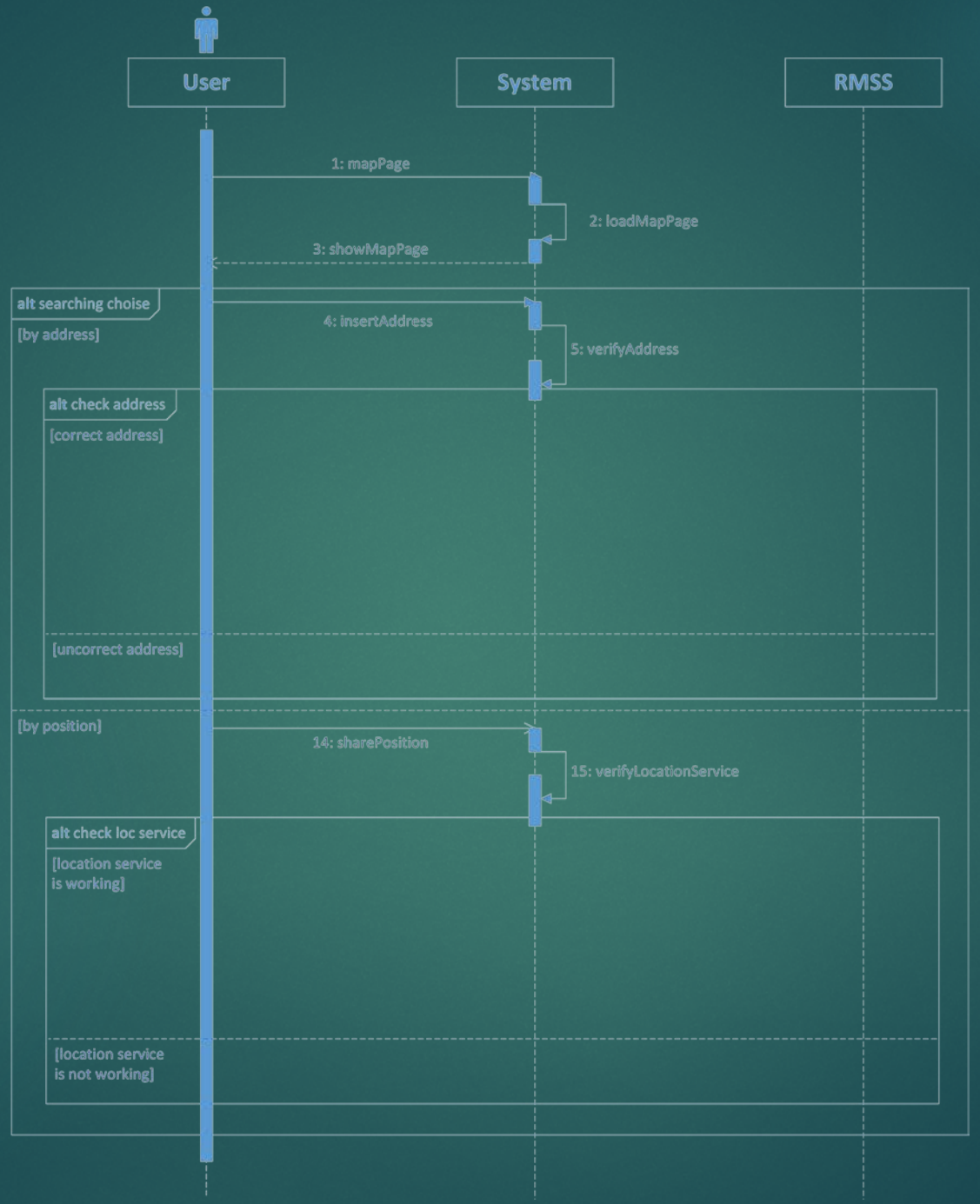
Actors	Guest, User or Deactivated User
Preconditions	The actor has a working Internet connection.
Events	1. The actor opens the map 2. The system requires the actor to enter an address or to use his position to localize cars 3. The actor types the address where he wants to find an available car or clicks "Use my position" button 4. The system verifies the correctness of the information and send a request to the RMSS 5. The system receives an answer from the RMSS with the available cars and shows them to the actor on the map 6. The actor taps the icon that stands for the available car he chooses 7. The system shows to the actor the status of the selected available car
Postconditions	The actor obtains all the information about the position and the status of any available car in a certain area.
Exceptions	The address field is empty. The inserted address is not found or the location service does not work. There is no available cars in the selected area. In these cases the system notifies the error and cannot complete the request.

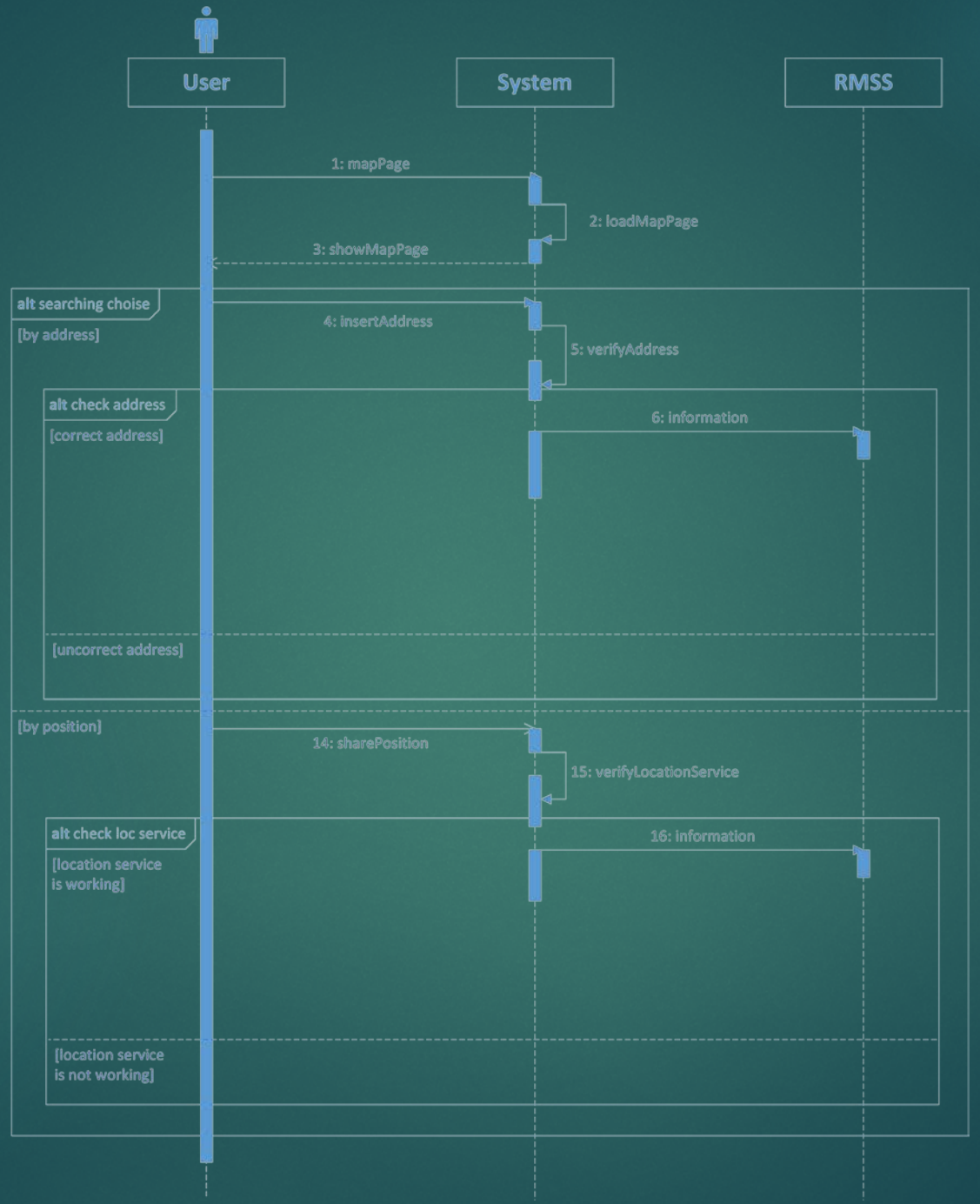

User
System
RMSS

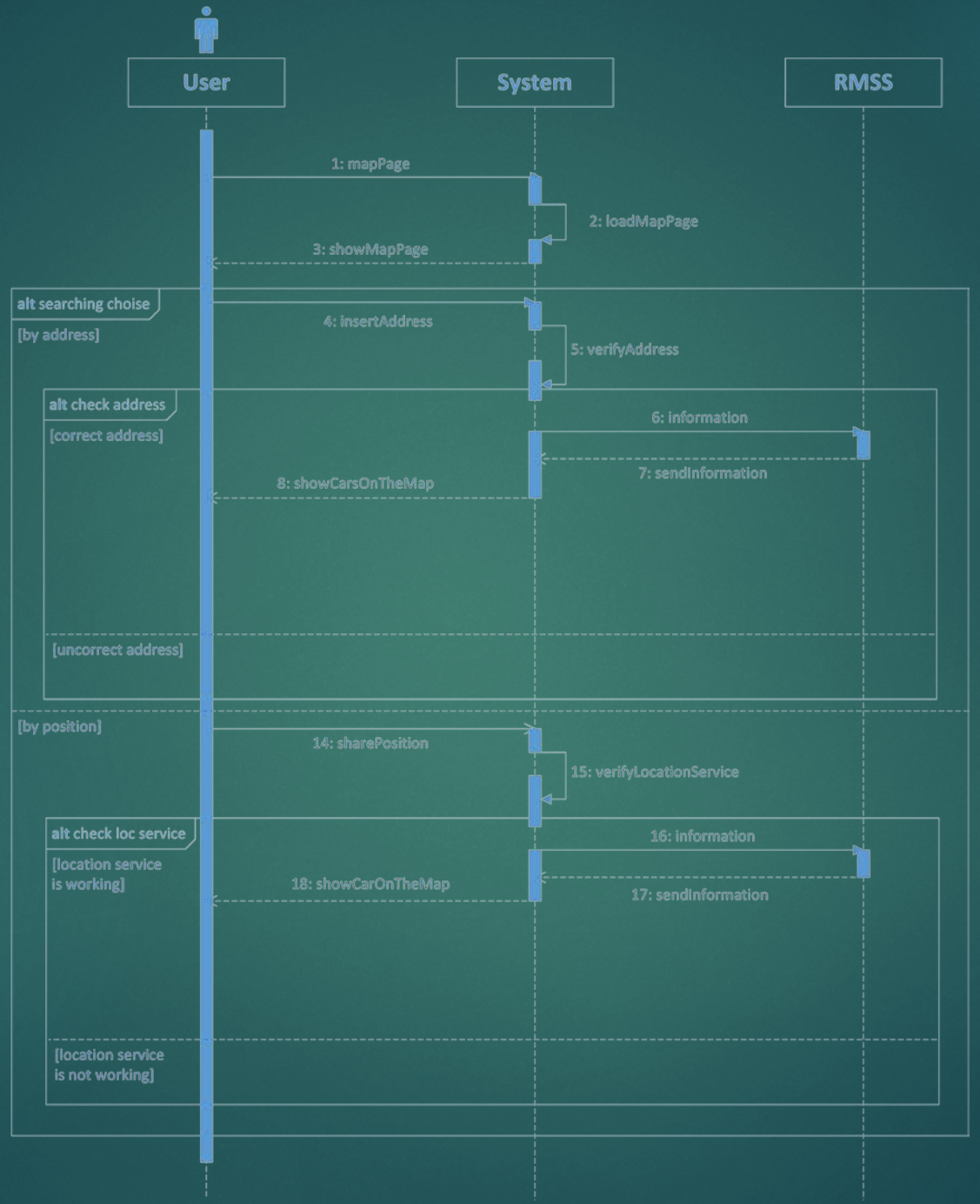

User
System
RMSS
1: mapPage

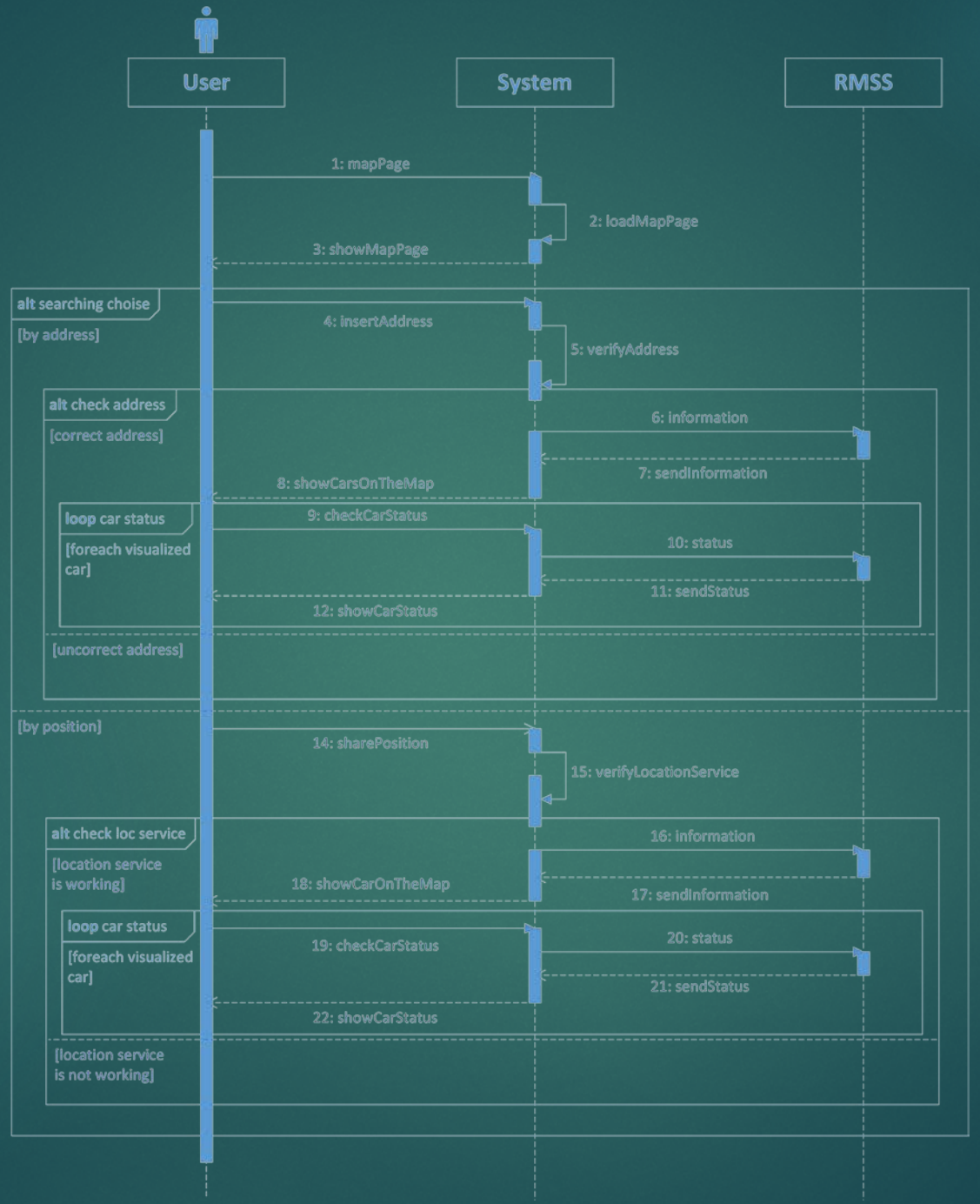

User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
[by position]

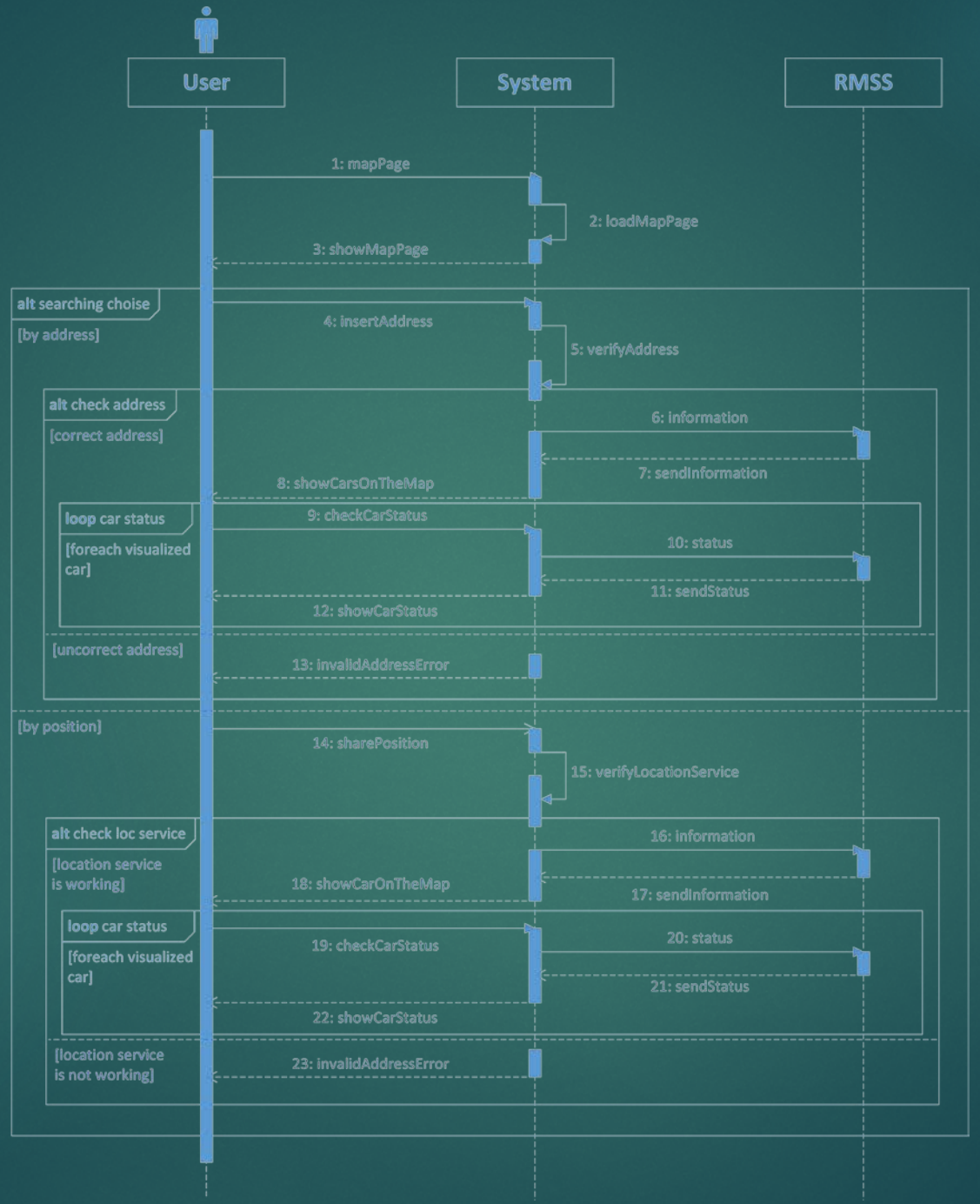

User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
4: insertAddress
5: verifyAddress
[by position]
14: sharePosition
15: verifyLocationService


User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
4: insertAddress
5: verifyAddress
alt check address
[correct address]
[uncorrect address]
[by position]
14: sharePosition
15: verifyLocationService
alt check loc service
[location service
is working]
[location service
is not working]


User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
4: insertAddress
5: verifyAddress
alt check address
[correct address]
6: information
[uncorrect address]
[by position]
14: sharePosition
15: verifyLocationService
alt check loc service
[location service is working]
16: information
[location service is not working]


User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
4: insertAddress
5: verifyAddress
alt check address
[correct address]
6: information
7: sendInformation
8: showCarsOnTheMap
[uncorrect address]
[by position]
14: sharePosition
15: verifyLocationService
alt check loc service
[location service is working]
16: information
17: sendInformation
18: showCarOnTheMap
[location service is not working]

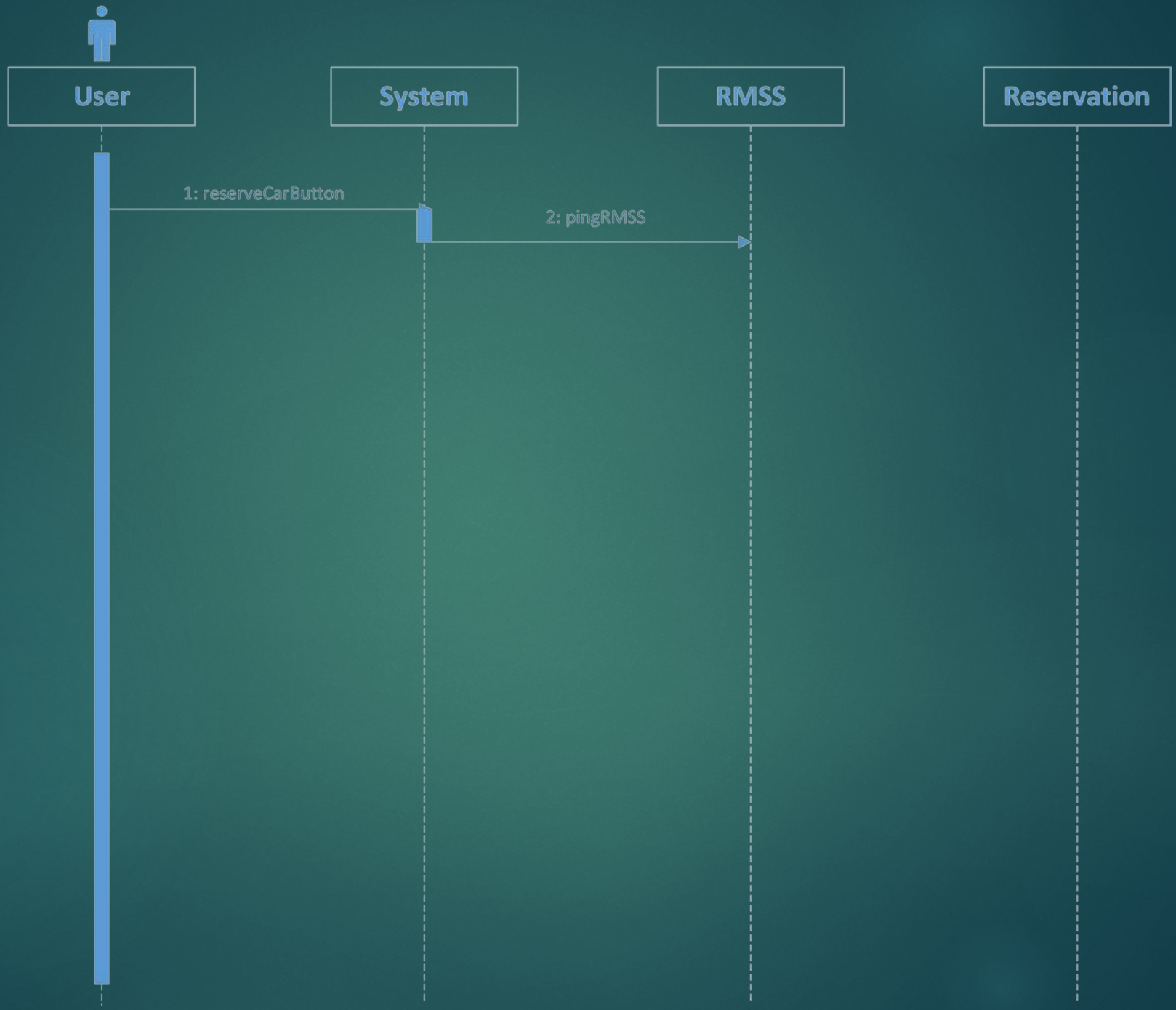

User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
4: insertAddress
5: verifyAddress
alt check address
[correct address]
6: information
7: sendInformation
8: showCarsOnTheMap
loop car status
[foreach visualized car]
9: checkCarStatus
10: status
11: sendStatus
12: showCarStatus
[uncorrect address]
[by position]
14: sharePosition
15: verifyLocationService
alt check loc service
[location service is working]
16: information
17: sendInformation
18: showCarOnTheMap
loop car status
[foreach visualized car]
19: checkCarStatus
20: status
21: sendStatus
22: showCarStatus
[location service is not working]

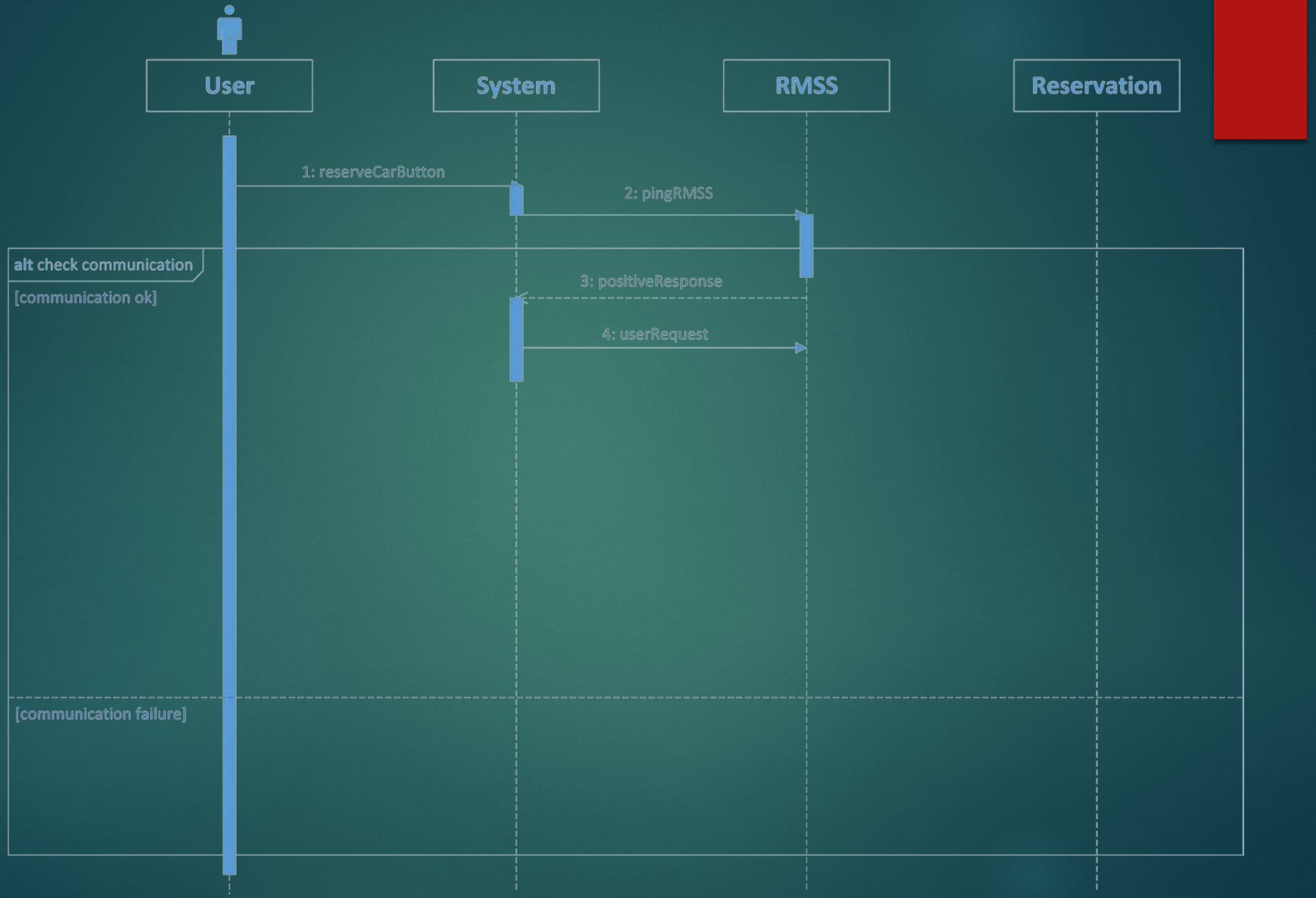

User
System
RMSS
1: mapPage
2: loadMapPage
3: showMapPage
alt searching choise
[by address]
4: insertAddress
5: verifyAddress
alt check address
[correct address]
6: information
7: sendInformation
8: showCarsOnTheMap
loop car status
[foreach visualized car]
9: checkCarStatus
10: status
11: sendStatus
12: showCarStatus
[uncorrect address]
13: invalidAddressError
[by position]
14: sharePosition
15: verifyLocationService
alt check loc service
[location service is working]
16: information
17: sendInformation
18: showCarOnTheMap
loop car status
[foreach visualized car]
19: checkCarStatus
20: status
21: sendStatus
22: showCarStatus
[location service is not working]
23: invalidAddressError

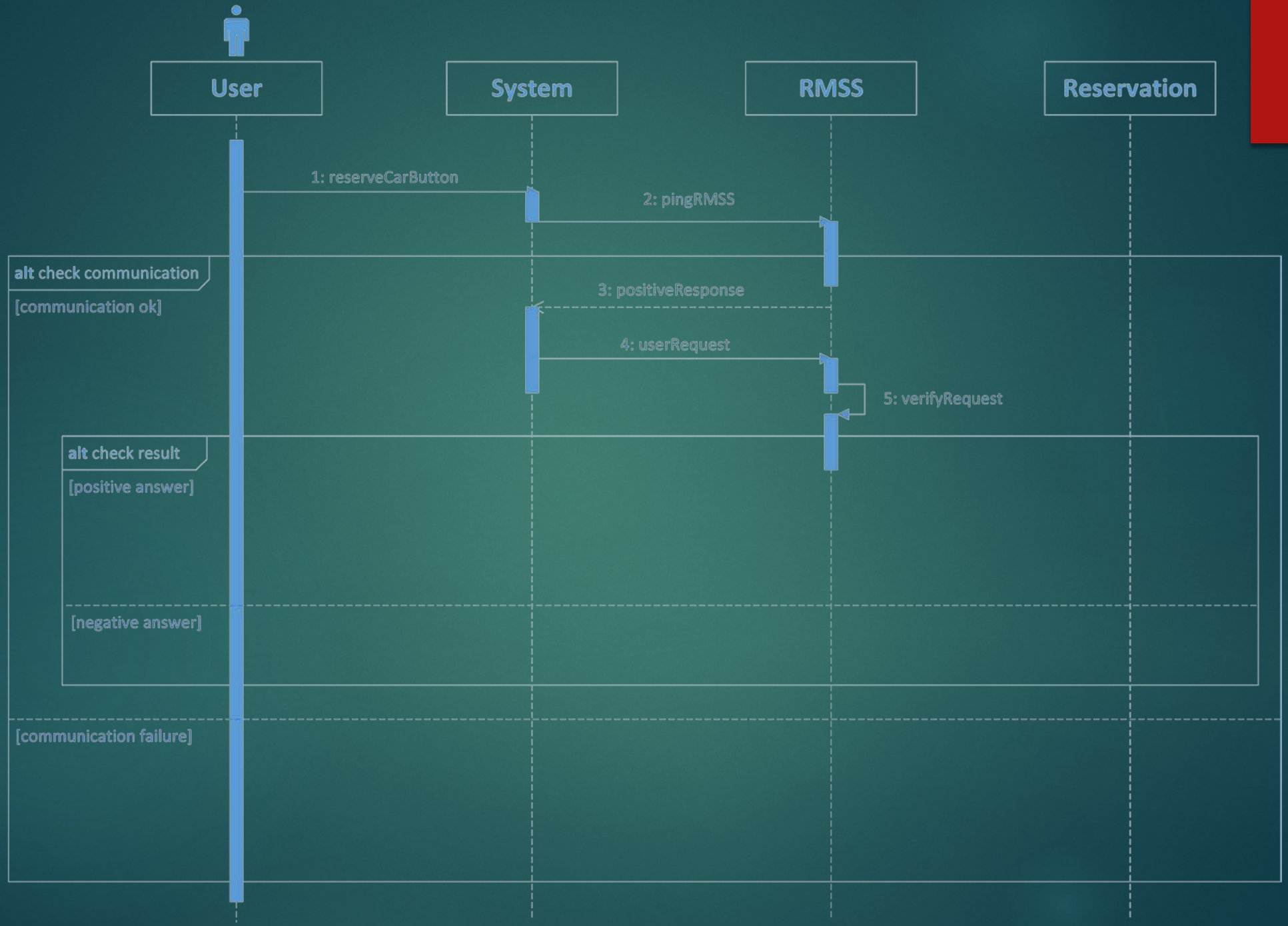
5.1.6 Reserve a car

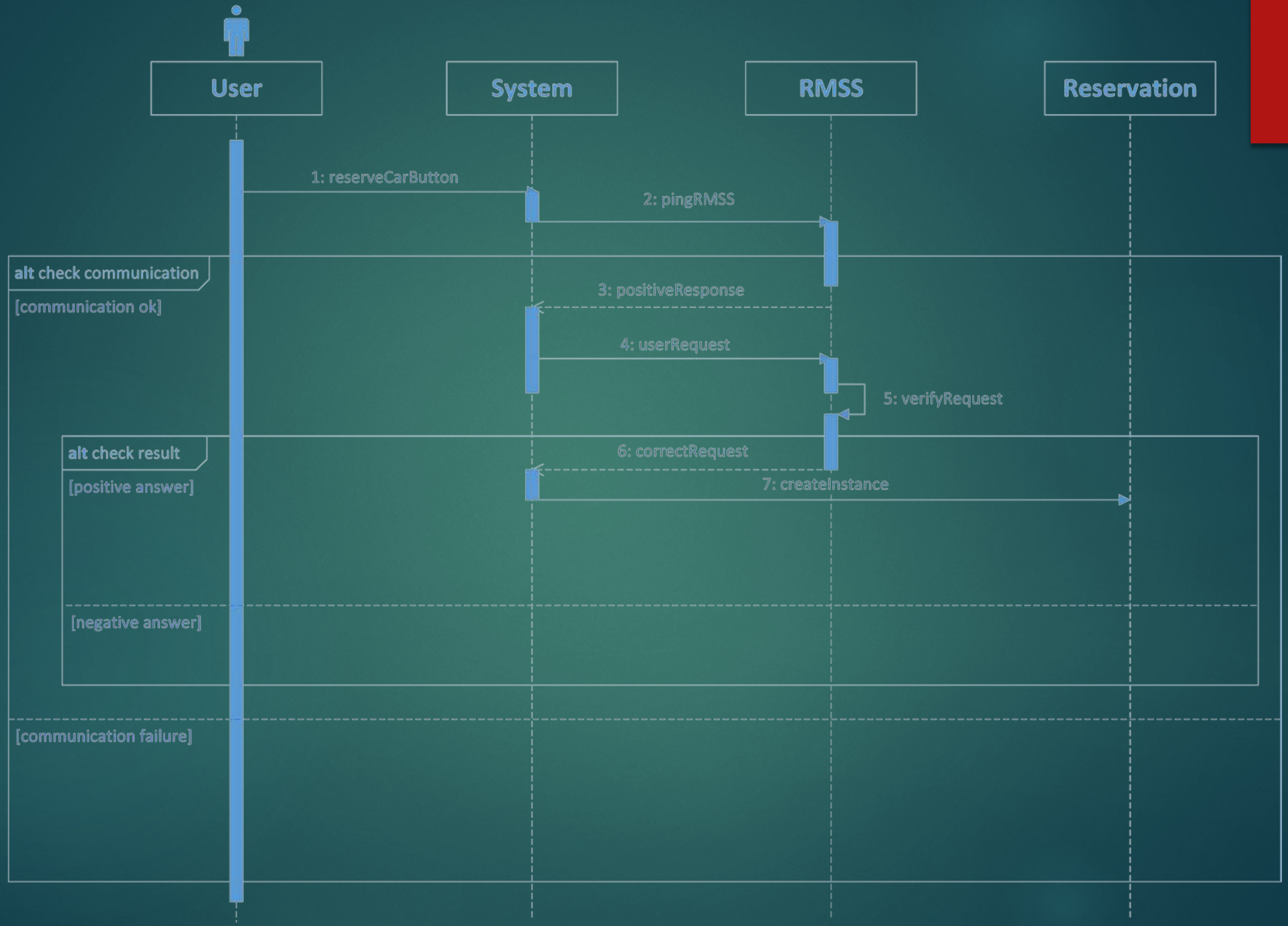
Actors	User
Preconditions	The user has a working Internet connection, he has already checked the position and the status of an available car and he is logged into the system for the whole reservation.
Events	1. The user clicks on “Reserve car” button 2. The system send to the RMSS the user position and his reservation request 3. The system receives an affirmative answer from the RMSS 4. The system creates a new instance of the reservation 5. The system notifies the user the success of the reservation
Postconditions	The user reserved successfully a car.
Exceptions	The communication with the RMSS failed. The system says to the user that the service is temporarily not available. The localization service does not work. The system receives a negative answer from the RMSS. In these cases the system notifies the error to the user and he cannot complete the reservation.

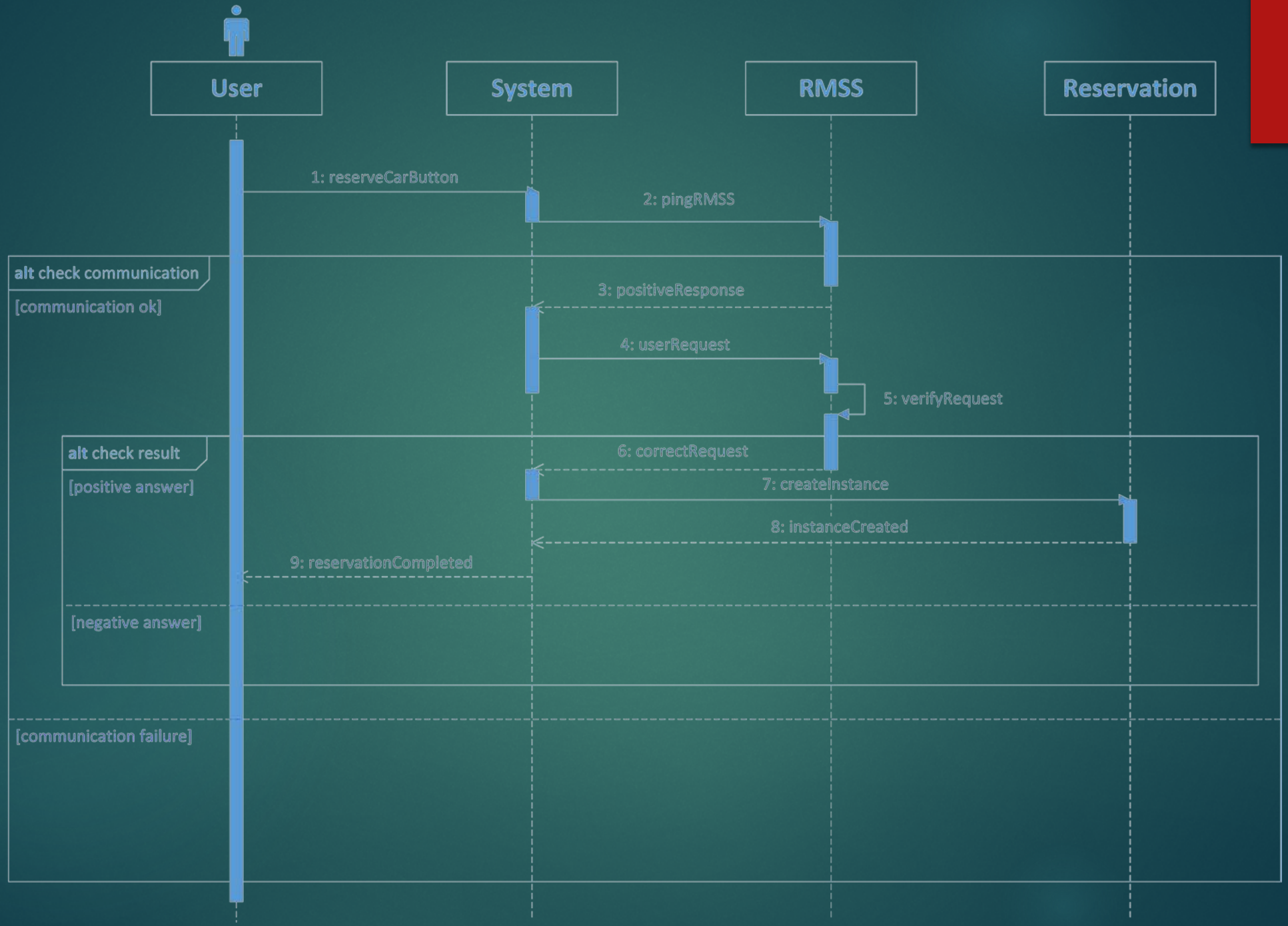

User
System
RMSS
Reservation

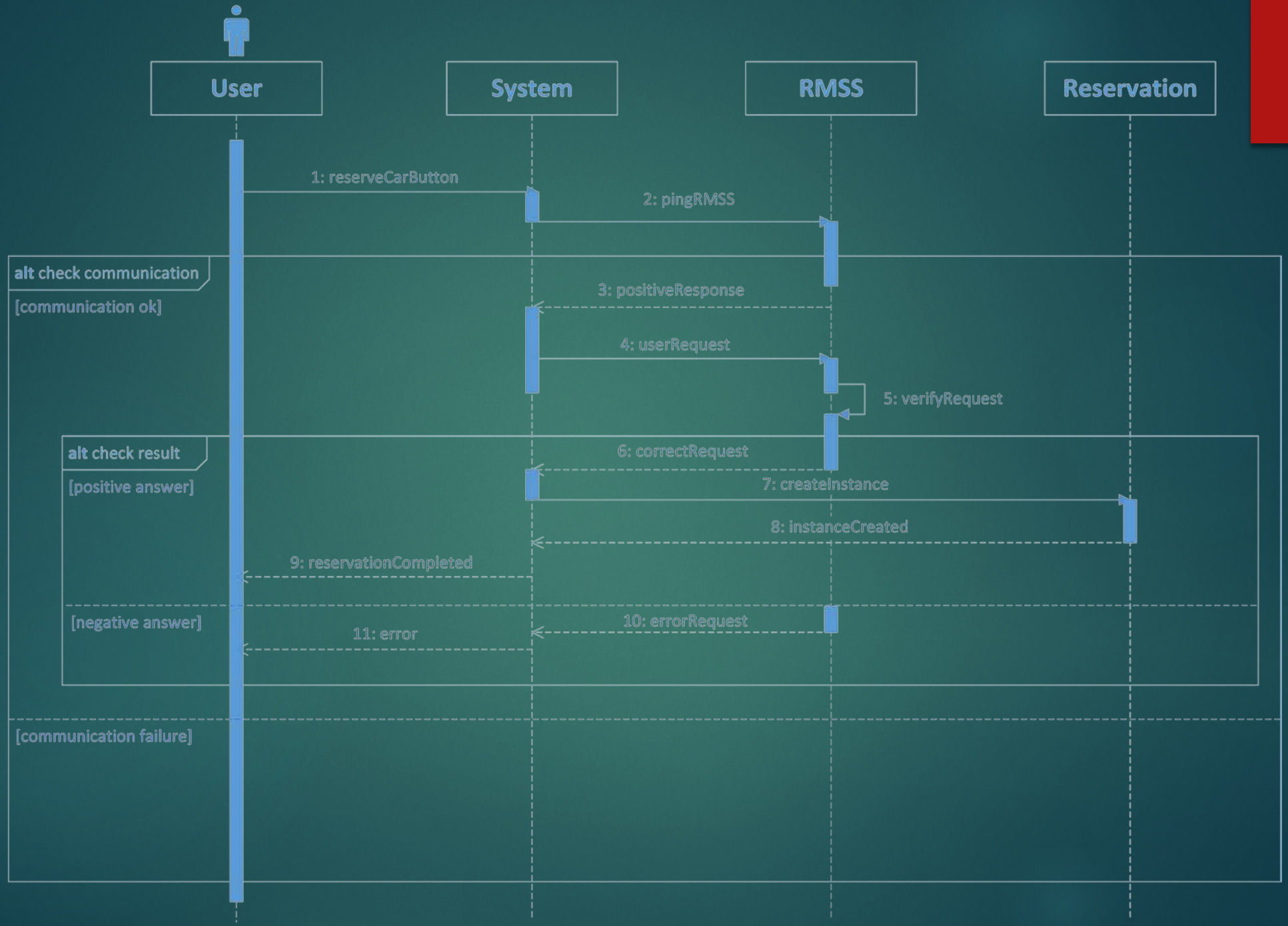

User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS

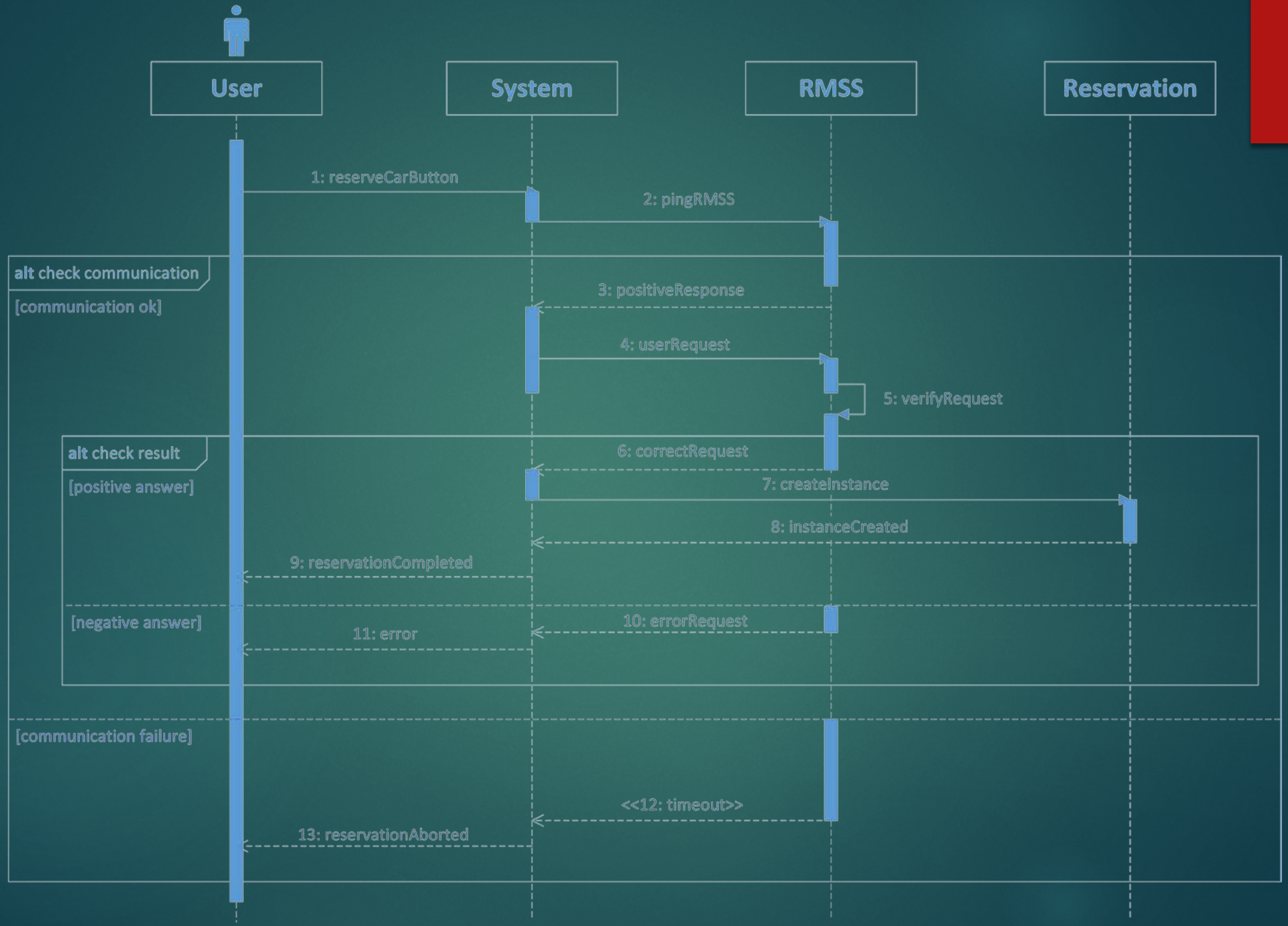

User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS
alt check communication
[communication ok]
3: positiveResponse
4: userRequest
[communication failure]


User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS
alt check communication
[communication ok]
3: positiveResponse
4: userRequest
5: verifyRequest
alt check result
[positive answer]
[negative answer]
[communication failure]


User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS
alt check communication
[communication ok]
3: positiveResponse
4: userRequest
5: verifyRequest
alt check result
[positive answer]
6: correctRequest
7: createInstance
[negative answer]
[communication failure]

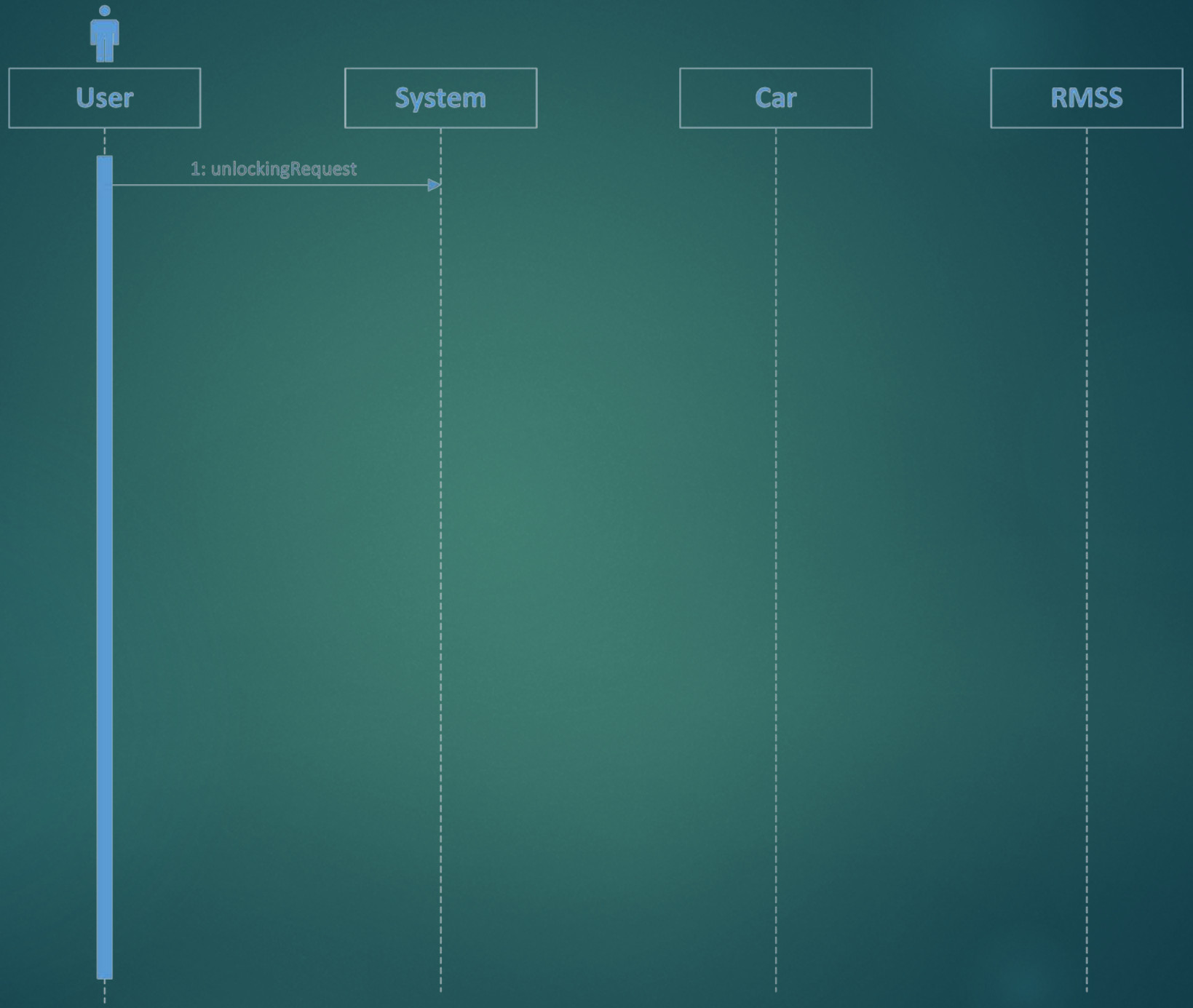

User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS
alt check communication
[communication ok]
3: positiveResponse
4: userRequest
5: verifyRequest
alt check result
[positive answer]
6: correctRequest
7: createInstance
8: instanceCreated
9: reservationCompleted
[negative answer]
[communication failure]

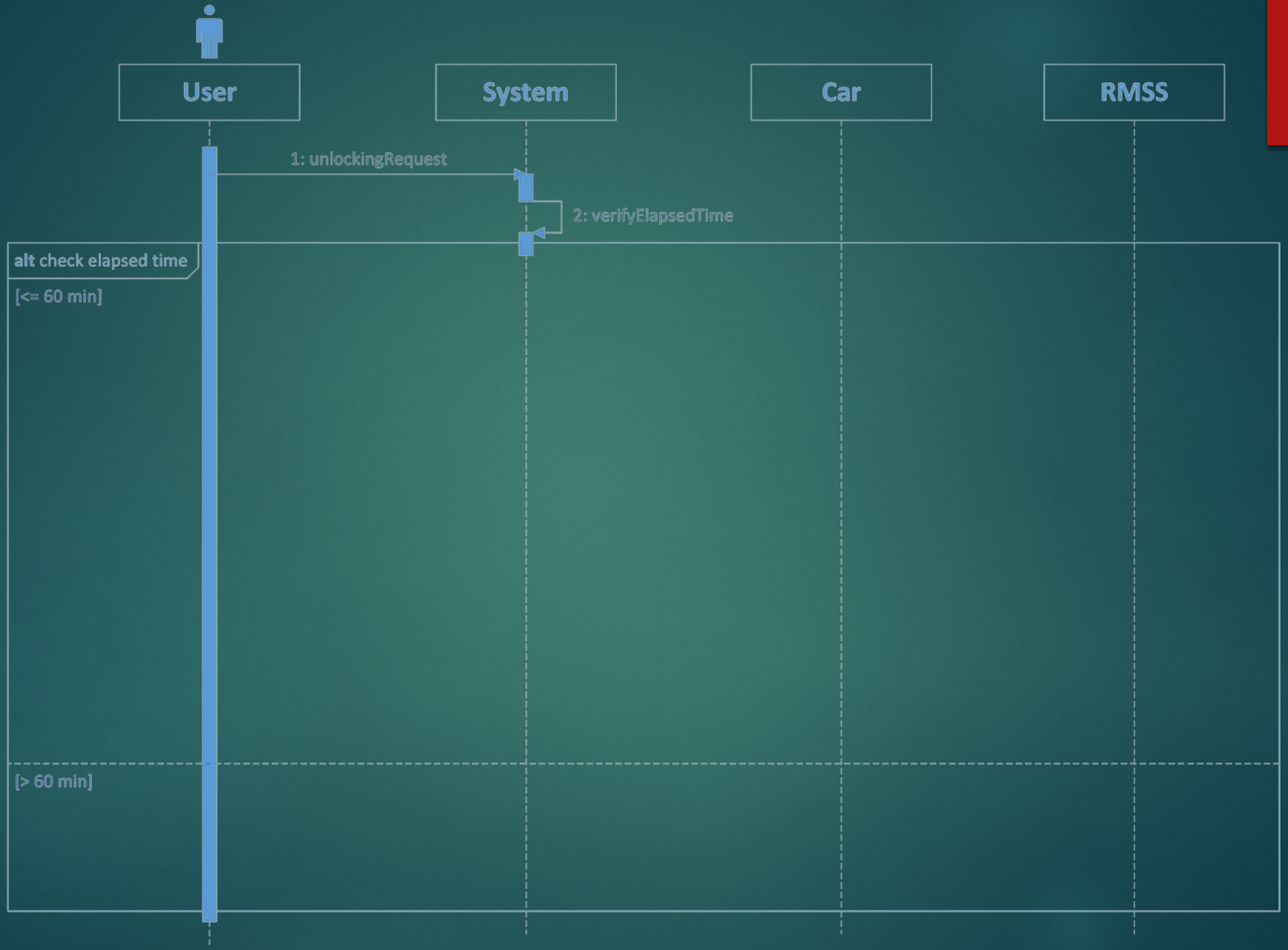

User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS
alt check communication
[communication ok]
3: positiveResponse
4: userRequest
5: verifyRequest
alt check result
[positive answer]
6: correctRequest
7: createInstance
8: instanceCreated
9: reservationCompleted
[negative answer]
10: errorRequest
11: error
[communication failure]

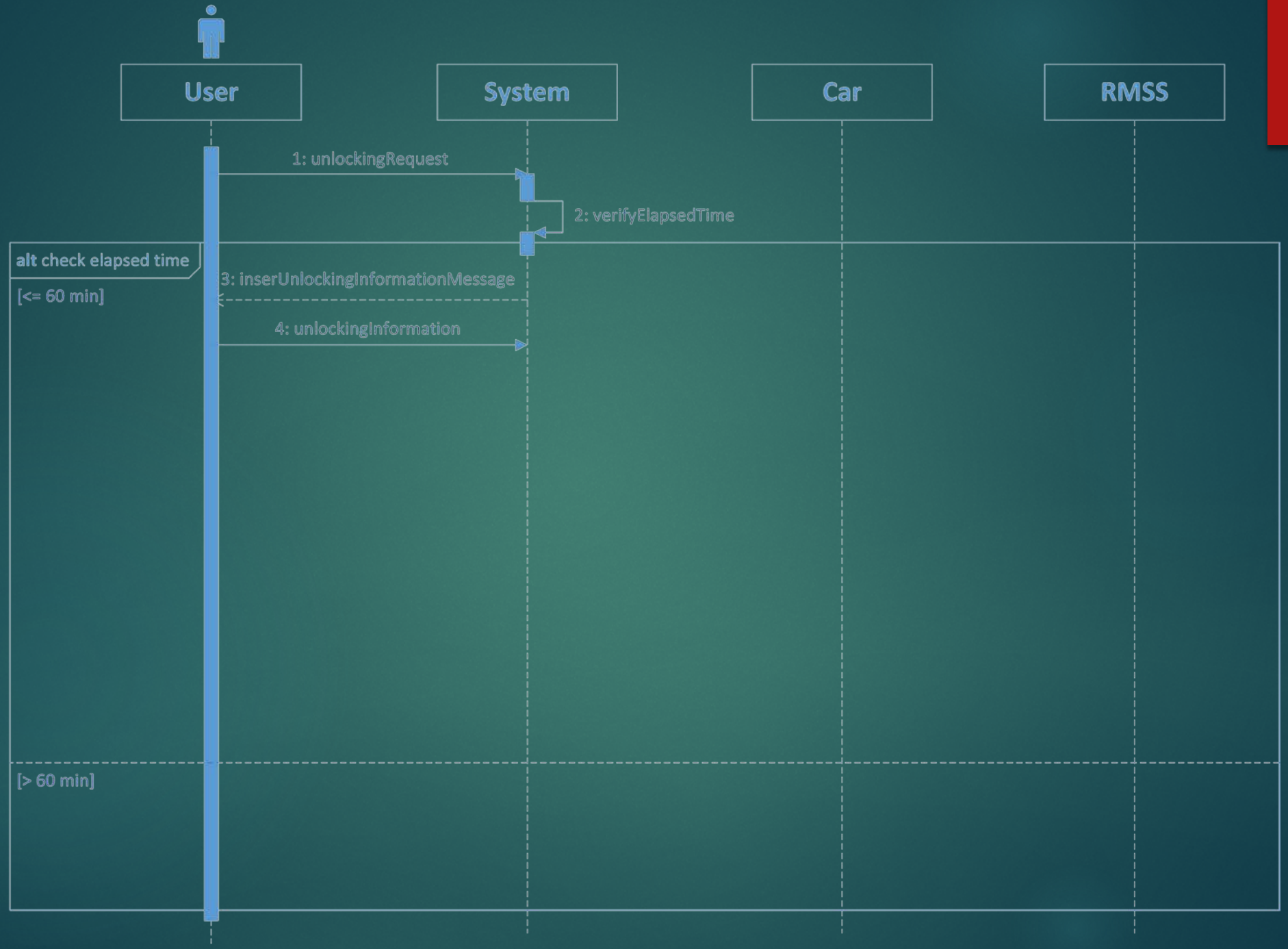

User
System
RMSS
Reservation
1: reserveCarButton
2: pingRMSS
alt check communication
[communication ok]
3: positiveResponse
4: userRequest
5: verifyRequest
alt check result
[positive answer]
6: correctRequest
7: createInstance
8: instanceCreated
9: reservationCompleted
[negative answer]
10: errorRequest
11: error
[communication failure]
<<12: timeout>>
13: reservationAborted

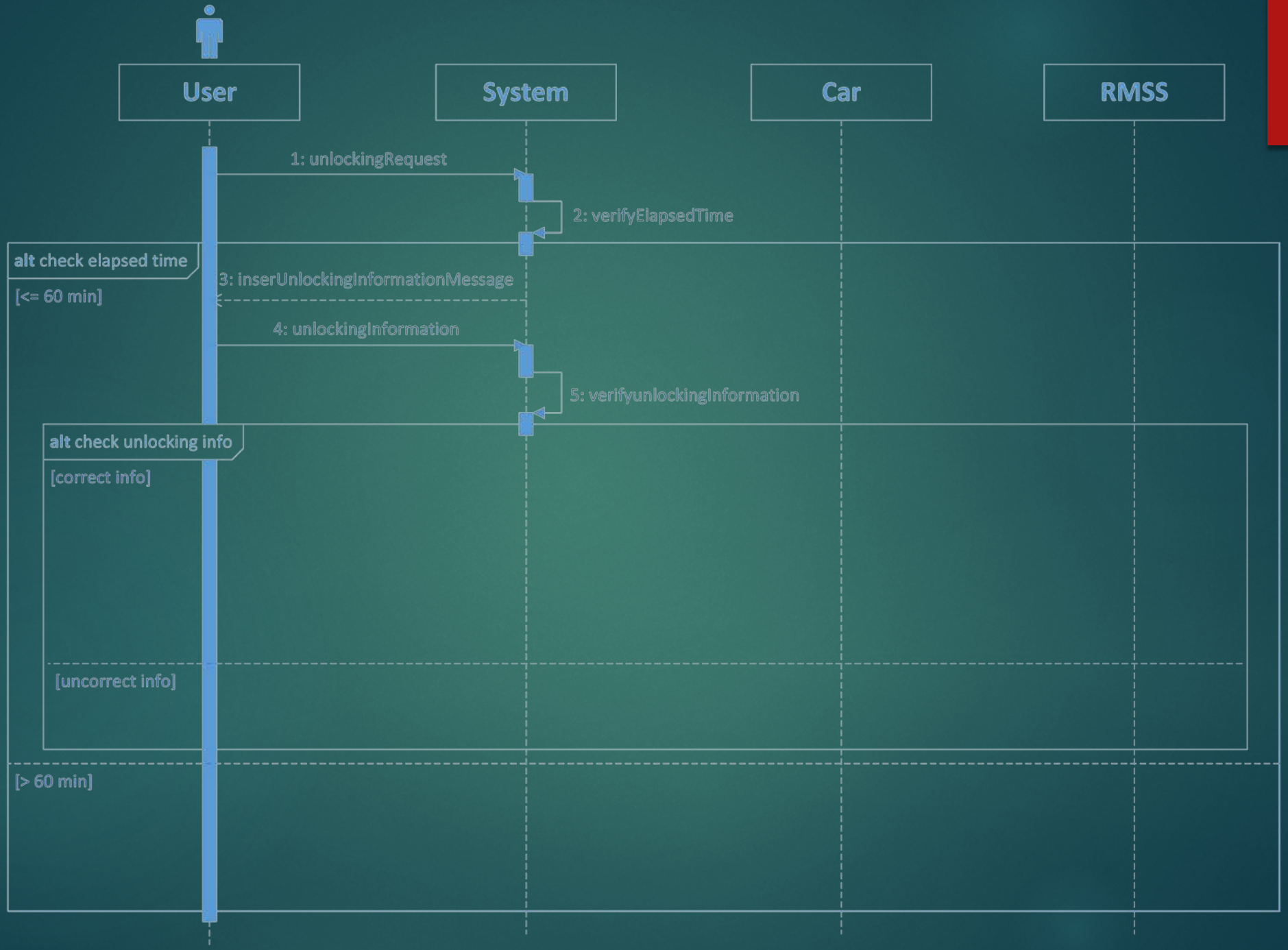
5.1.7 Open the reserved car

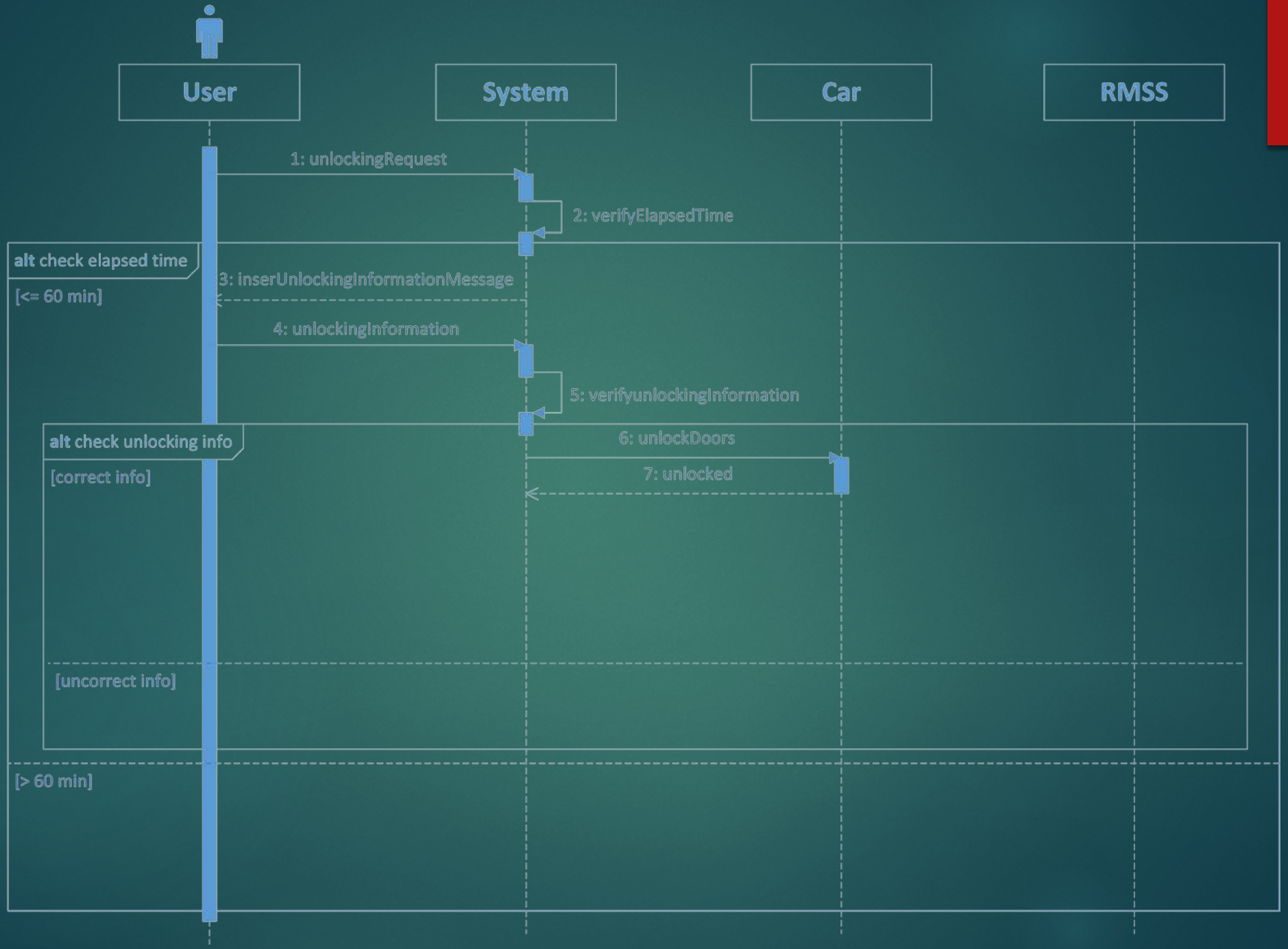
Actors	User
Preconditions	The user has a working Internet connection, has already reserved a car and he is logged into the system.
Events	1. The user reaches the car he reserved. 2. The system checks user position or requires him to enter the code on the windshield 3. If necessary, the user enters the code he can read on the windshield 4. The system requires the user enter his verification code 5. The user types his verification code 6. The system verifies the correctness of the verification code and the status of the reservation 7. The system unlocks the car doors, terminates the reservation and creates a rent instance on the RMSS 8. The user opens the car doors
Postconditions	The user can get in the car.
Exceptions	The user failed to unlock the car in less than one hour: the system notifies the end of the reservation to the user, declares the reserved car as available again and applies 1€ of fee to the user. | The system is not able to check the user position. The user fails to enter the windshield code. The verification code is not correct. In these cases the system doesn't unlock the doors.

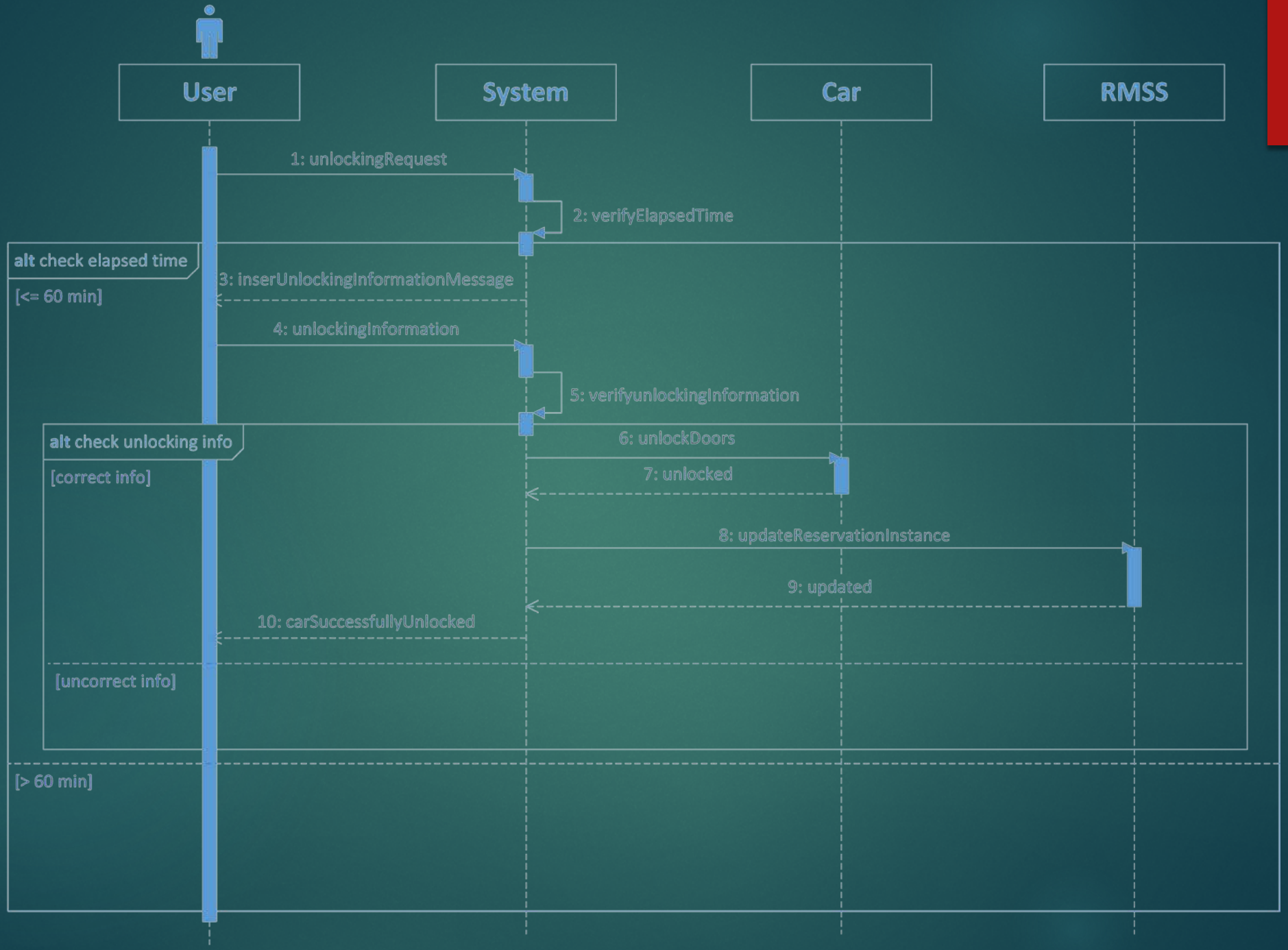

User
System
Car
RMSS
1: unlockingRequest

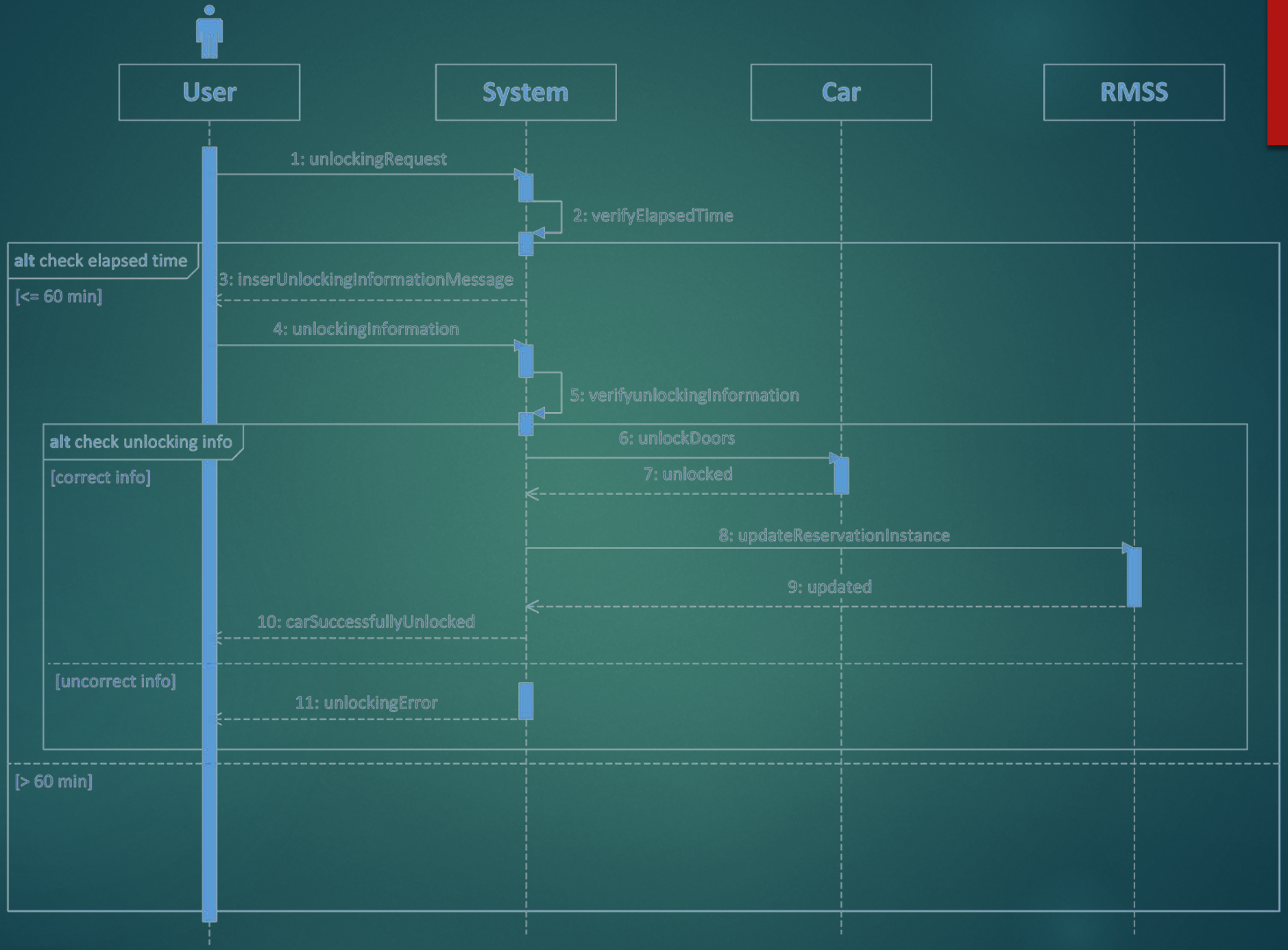

User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
[> 60 min]

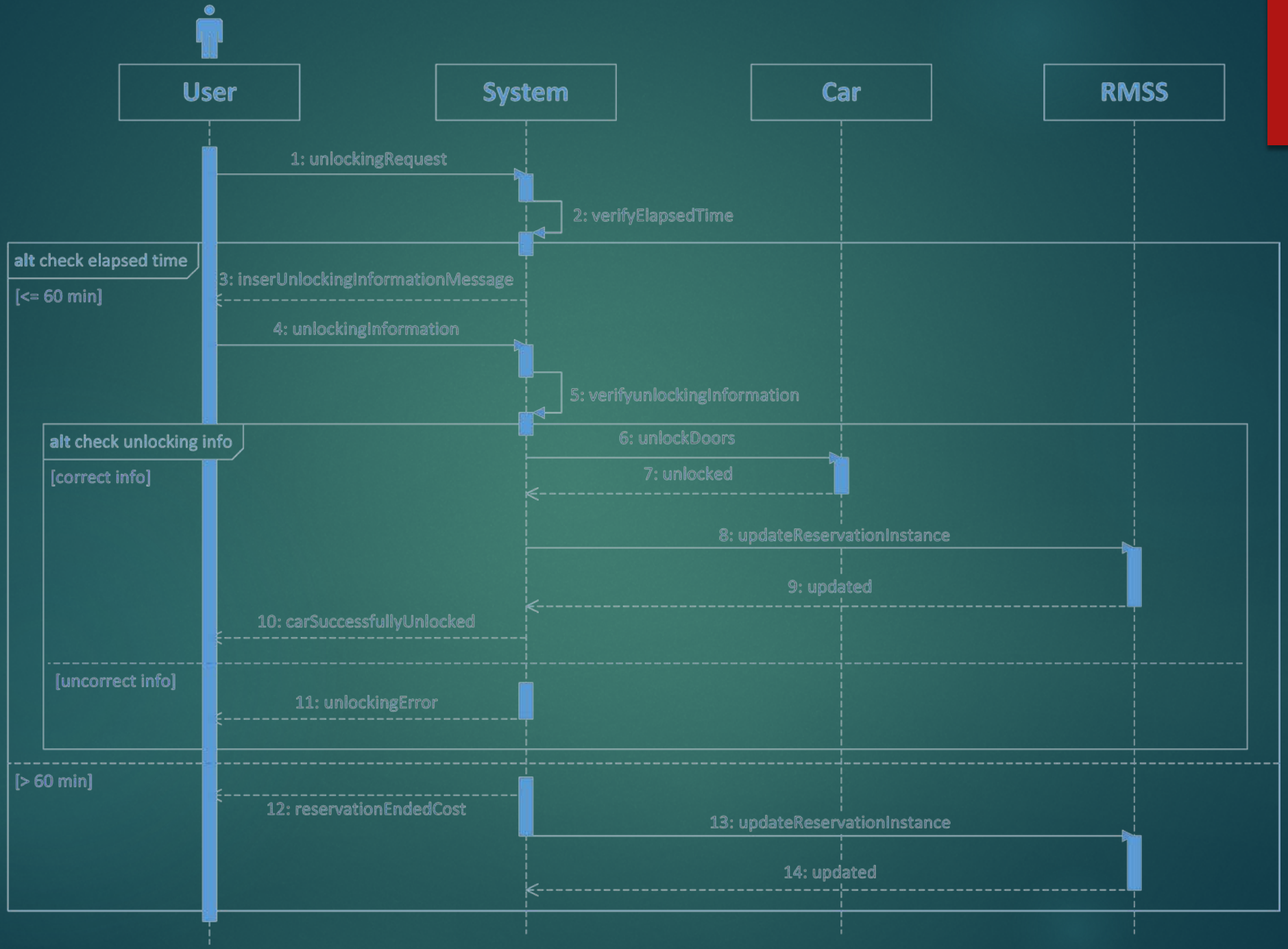

User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
3: inserUnlockingInformationMessage
4: unlockingInformation
[> 60 min]


User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
3: inserUnlockingInformationMessage
4: unlockingInformation
5: verifyunlockingInformation
alt check unlocking info
[correct info]
[uncorrect info]
[> 60 min]


User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
3: inserUnlockingInformationMessage
4: unlockingInformation
5: verifyunlockingInformation
alt check unlocking info
[correct info]
6: unlockDoors
7: unlocked
[uncorrect info]
[> 60 min]

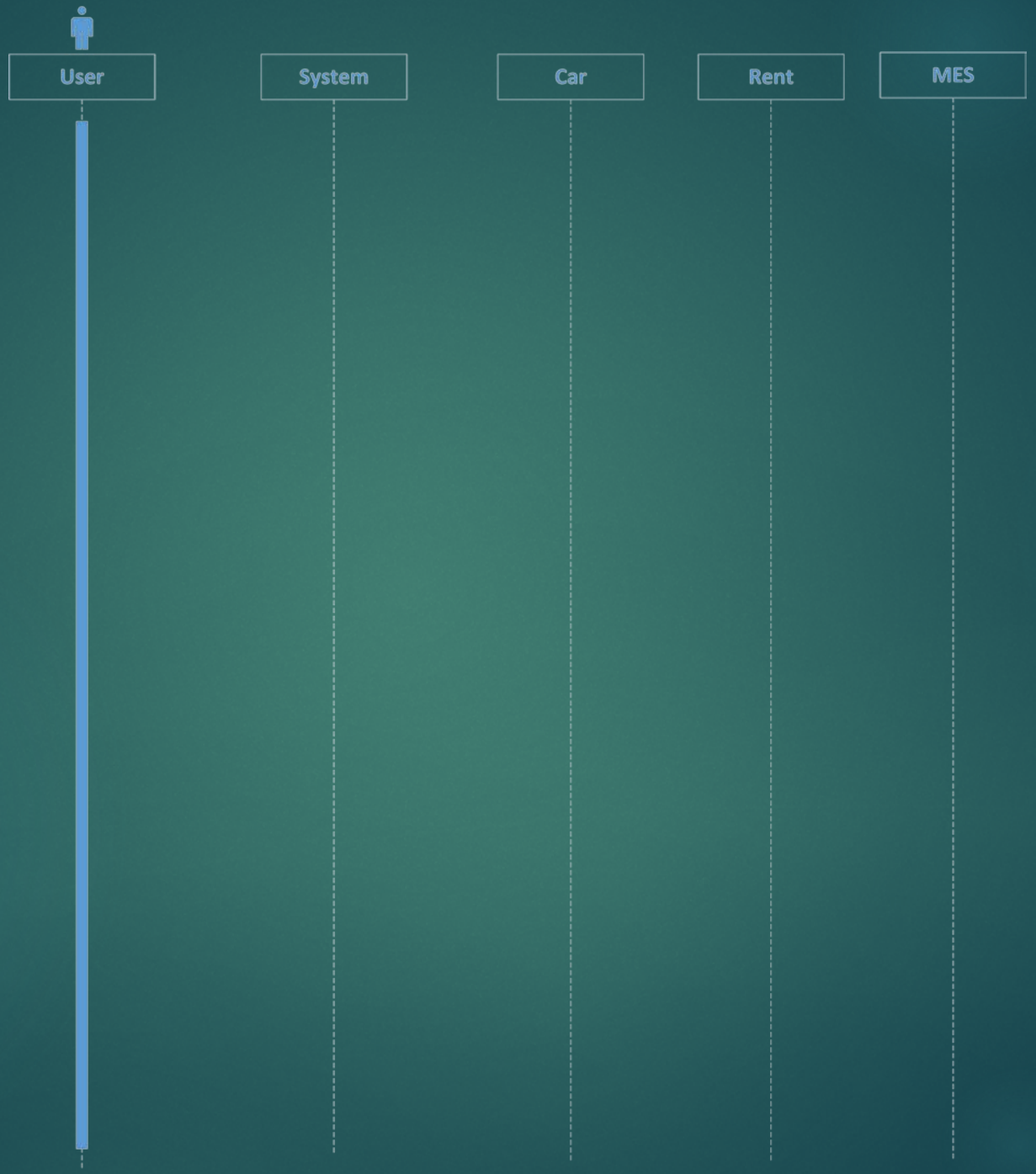

User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
3: inserUnlockingInformationMessage
4: unlockingInformation
5: verifyunlockingInformation
alt check unlocking info
[correct info]
6: unlockDoors
7: unlocked
8: updateReservationInstance
9: updated
10: carSuccessfullyUnlocked
[uncorrect info]
[> 60 min]


User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
3: inserUnlockingInformationMessage
4: unlockingInformation
5: verifyunlockingInformation
alt check unlocking info
[correct info]
6: unlockDoors
7: unlocked
8: updateReservationInstance
9: updated
10: carSuccessfullyUnlocked
[uncorrect info]
11: unlockingError
[> 60 min]

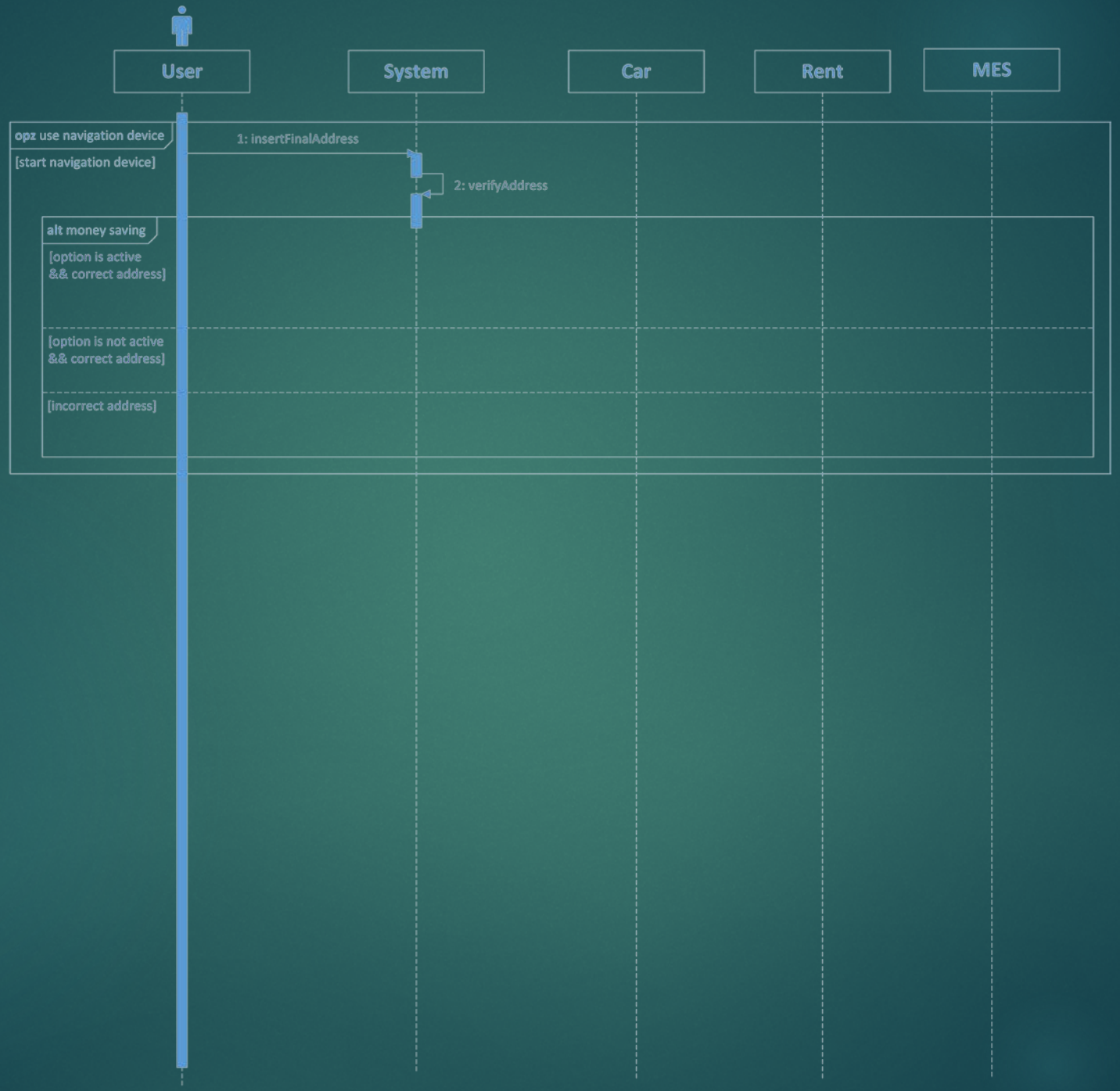

User
System
Car
RMSS
1: unlockingRequest
2: verifyElapsedTime
alt check elapsed time
[<= 60 min]
3: inserUnlockingInformationMessage
4: unlockingInformation
5: verifyunlockingInformation
alt check unlocking info
[correct info]
6: unlockDoors
7: unlocked
8: updateReservationInstance
9: updated
10: carSuccessfullyUnlocked
[uncorrect info]
11: unlockingError
[> 60 min]
12: reservationEndedCost
13: updateReservationInstance
14: updated

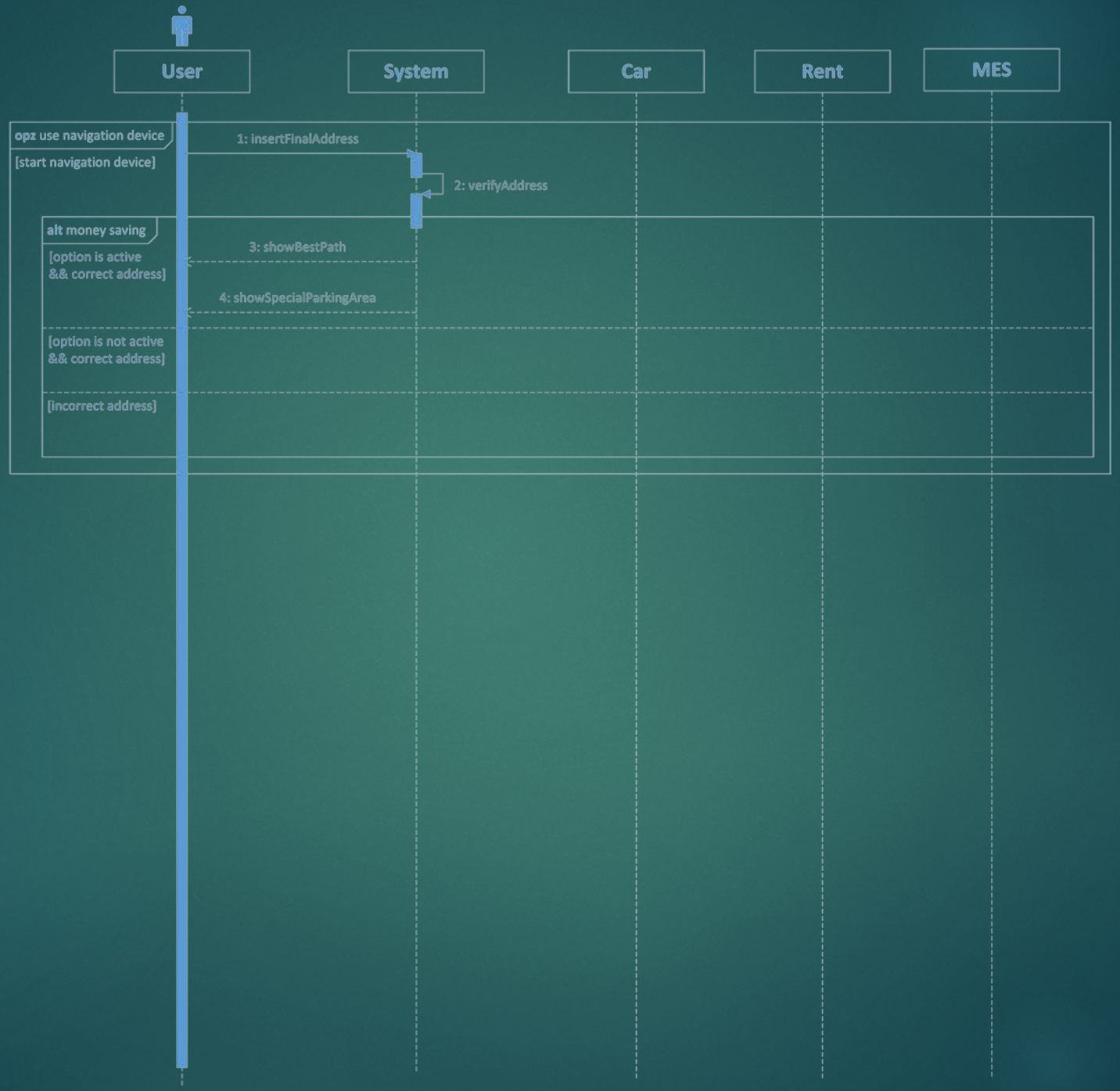
5.1.8 Use the reserved car and terminate the rent

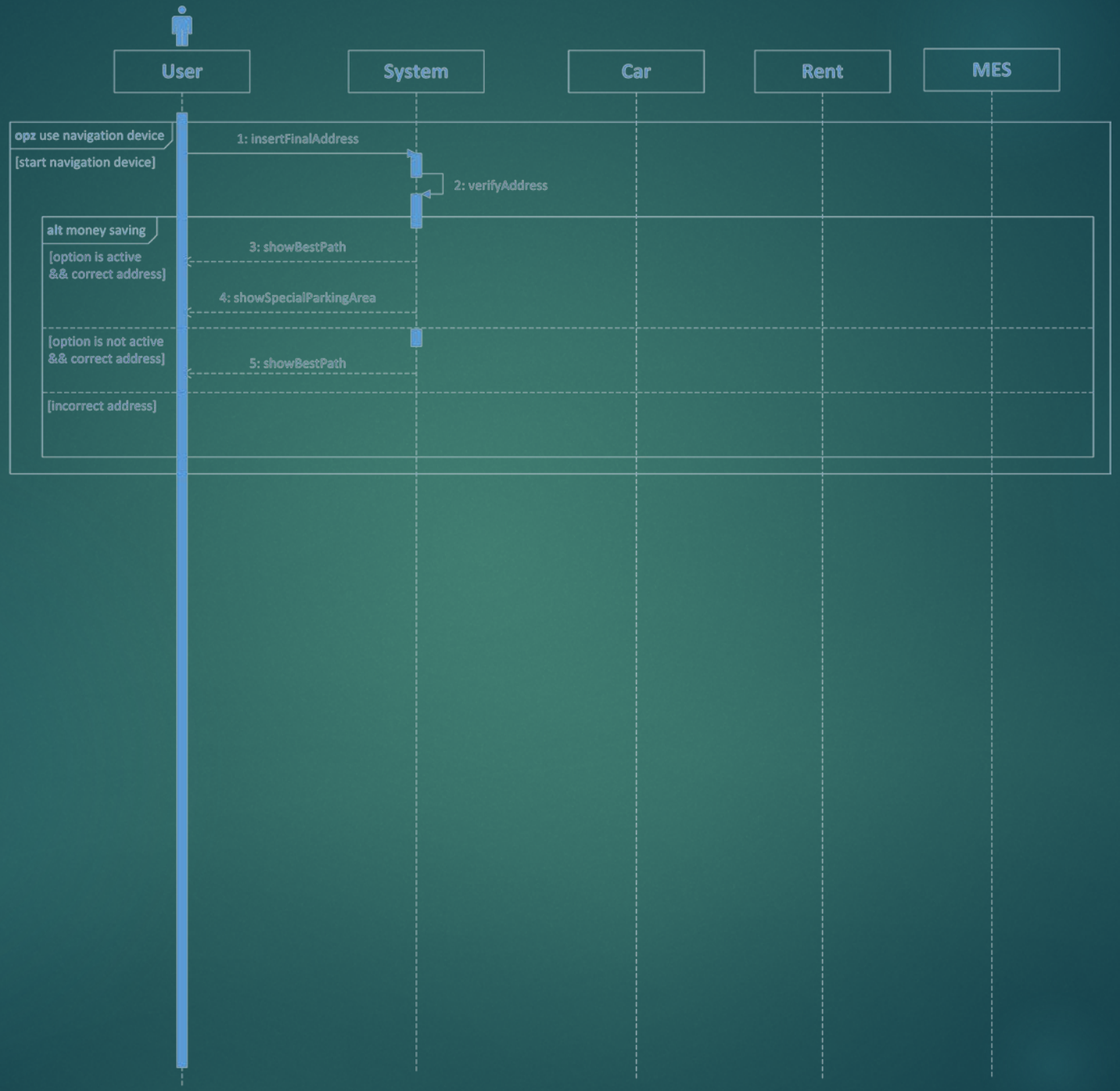
Actors	User
Preconditions	The user has a working Internet connection, has already opened his reserved car and he is logged into the system.
Events	1. If the money saving option is actived, the user can enter his final address on the touchscreen display in the car 2. If the user did it, the system shows a special parking area near his destination 3. The user starts the engine and uses the car 4. As a matter of choice, the user takes care of plugging the car into the power grid 5. The user clicks "Terminate rent" 6. The system verifies nobody is in the car and that it is in a safe parking area 7. The system locks the car doors again, verifies the car's status and updates the rent instance 8. If necessary, the system contacts the MES for an ordinary (or extraordinary) car maintenance. 9. The system calculates the total cost and notifies the end of the rental to the user along with the total cost
Postconditions	The user successfully rented a car.
Exceptions	The address is not correct or it is not found. The system notifies the error to the user and requires him to insert it again. | The car is not in a safe parking area. The car is not empty. In these cases the system doesn't allow to terminate the rent. | The user can't pay the total cost: the system deactivate the user.


User
System
Car
Rent
MES

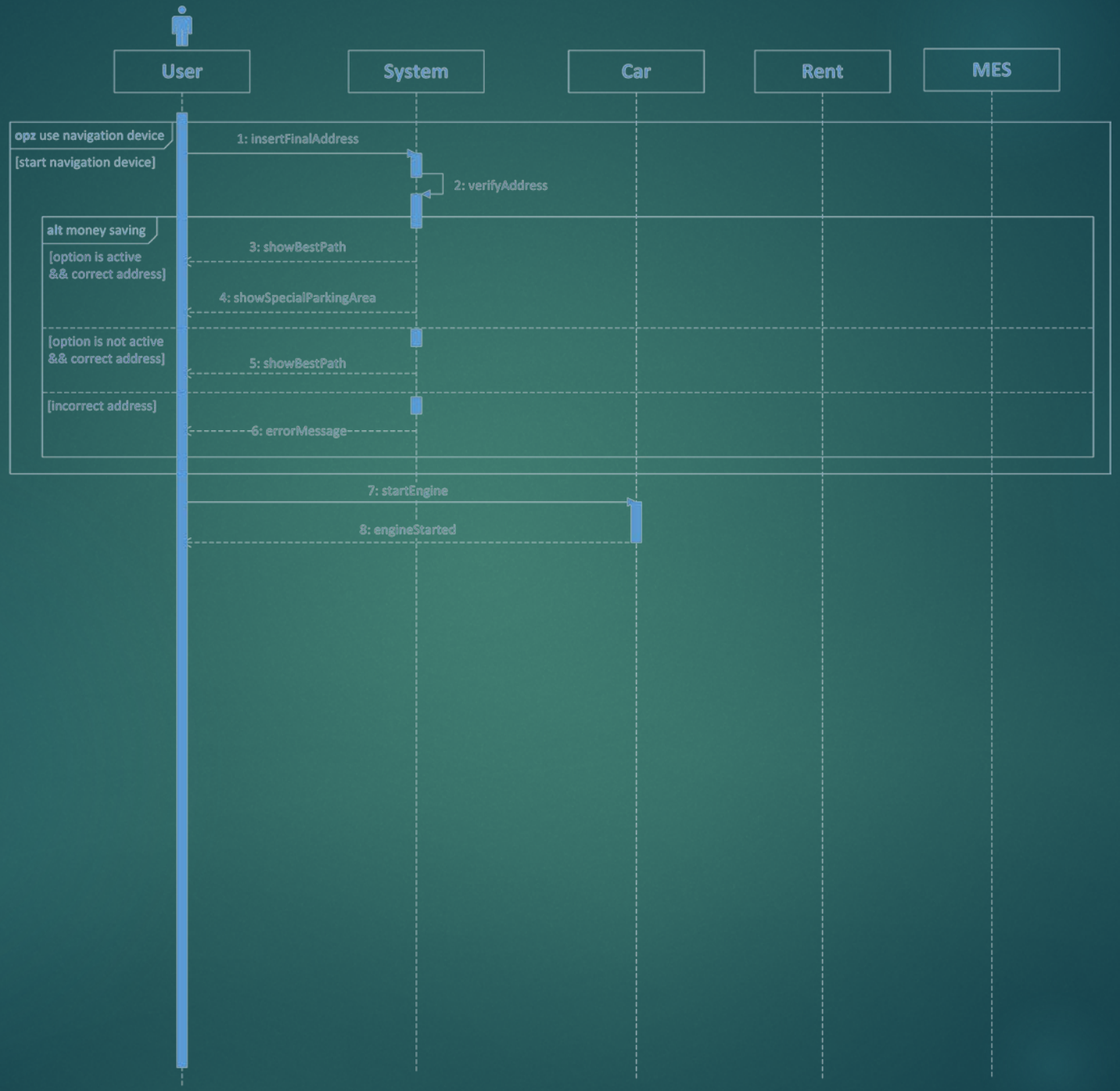

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress

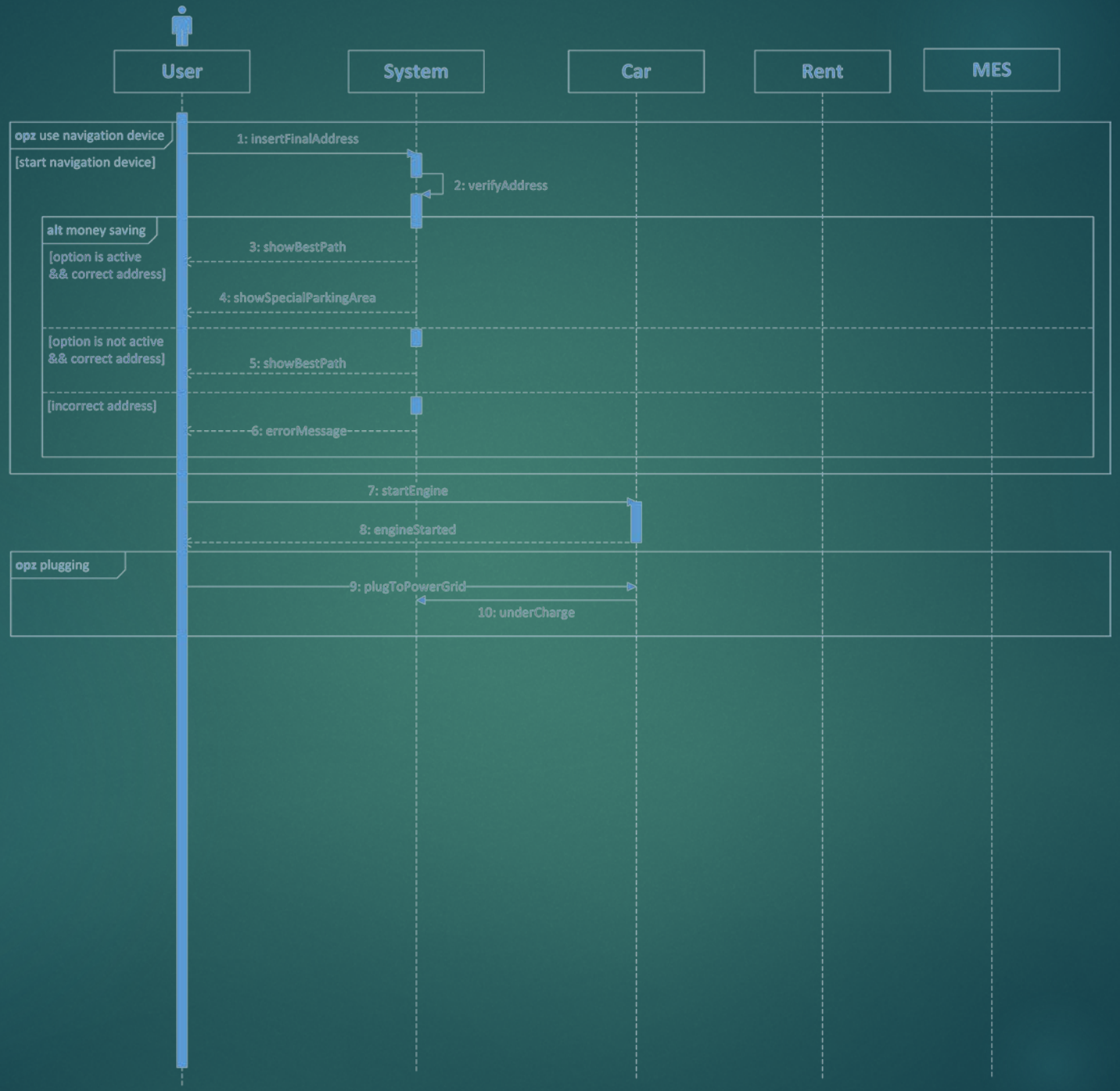

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
[option is not active
&& correct address]
[incorrect address]

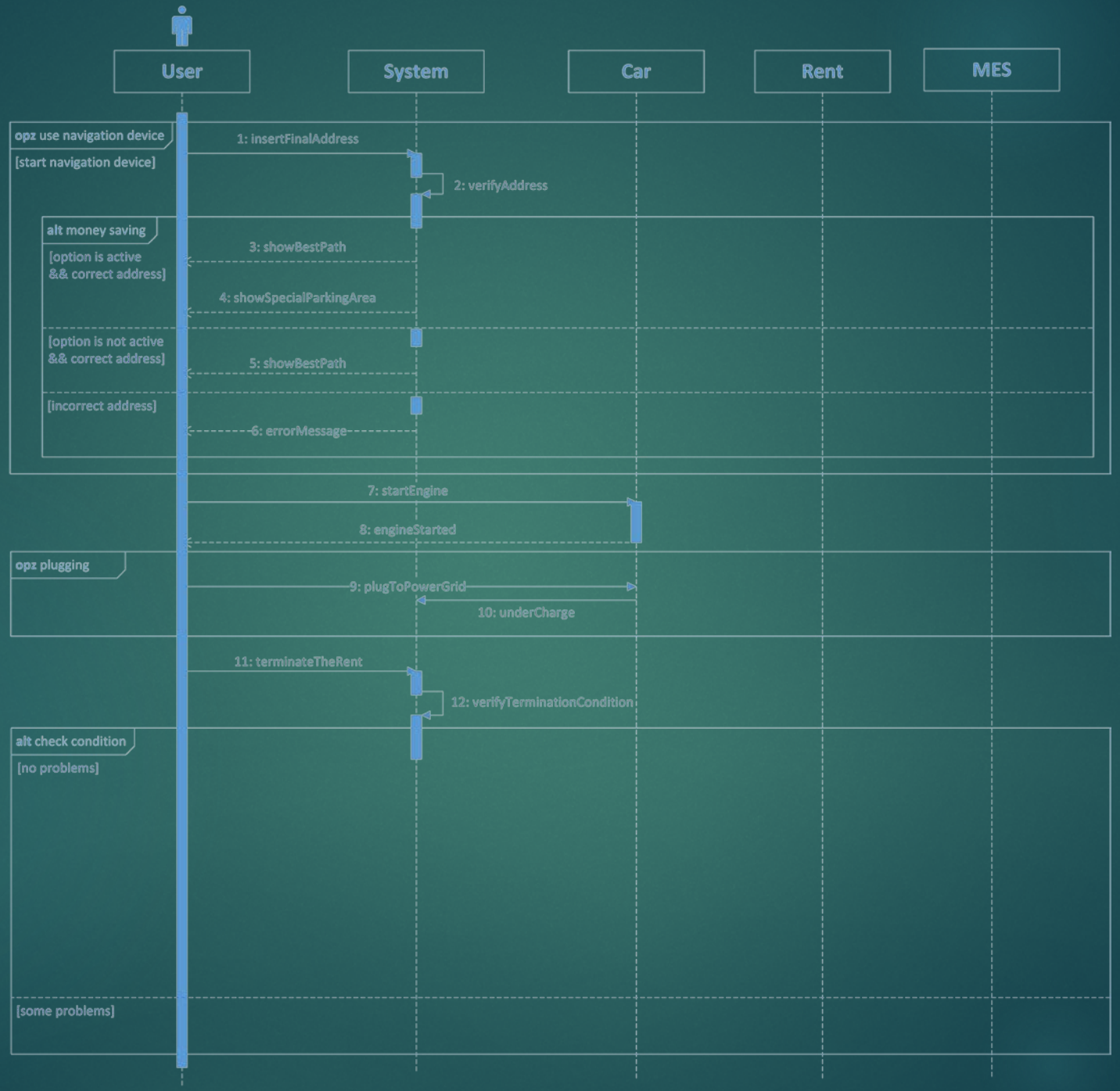

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
[incorrect address]

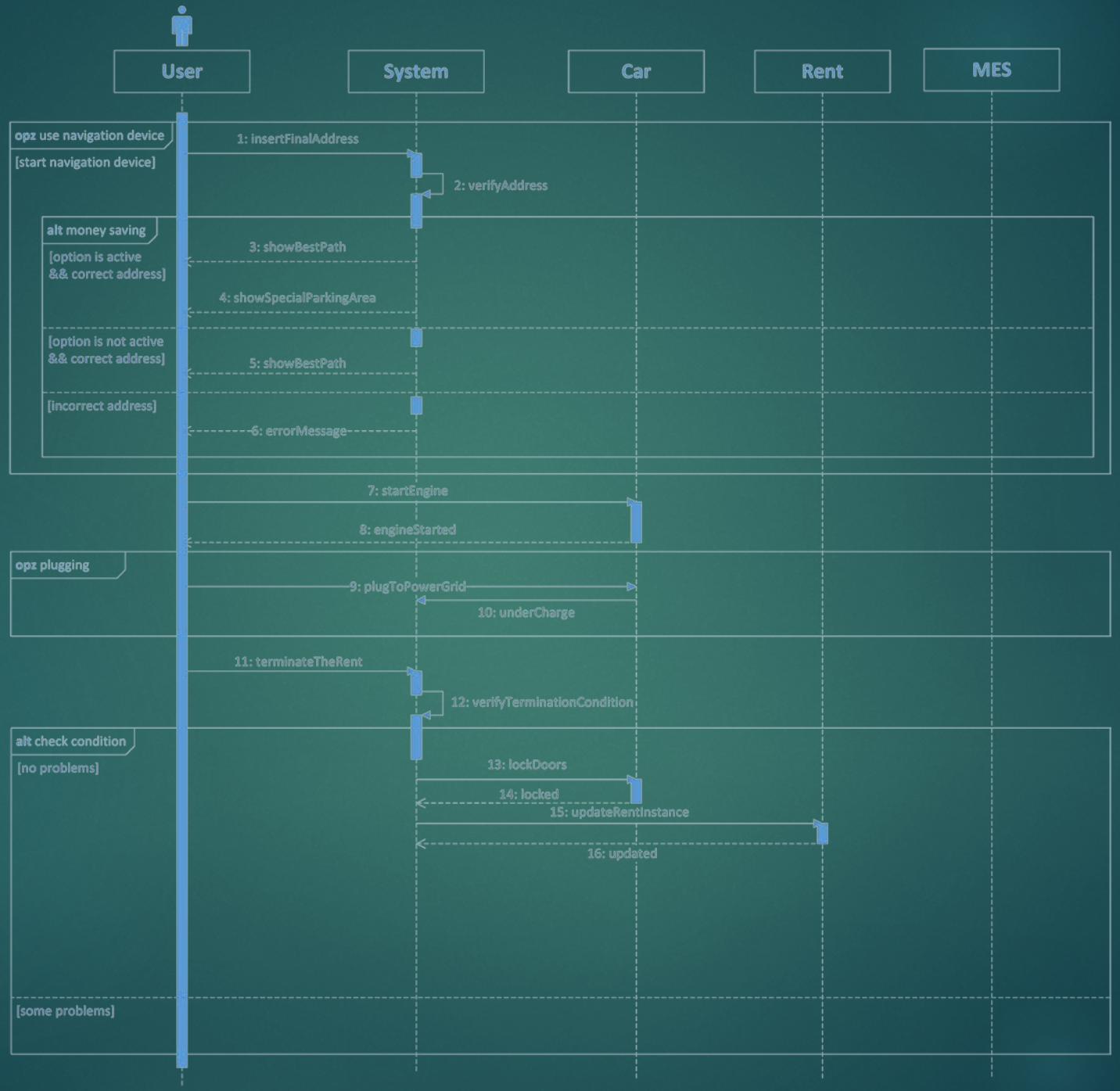

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]

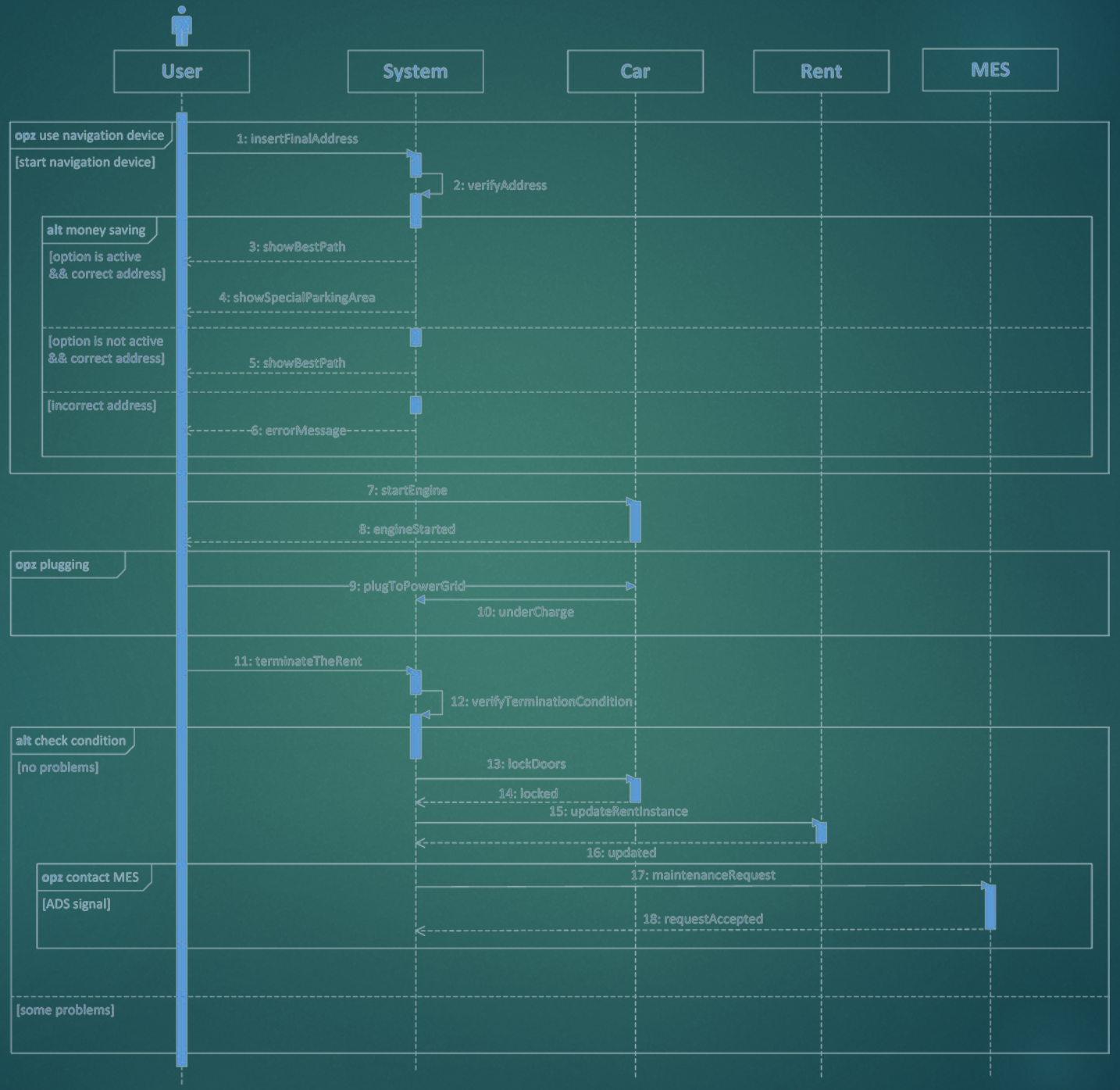

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage

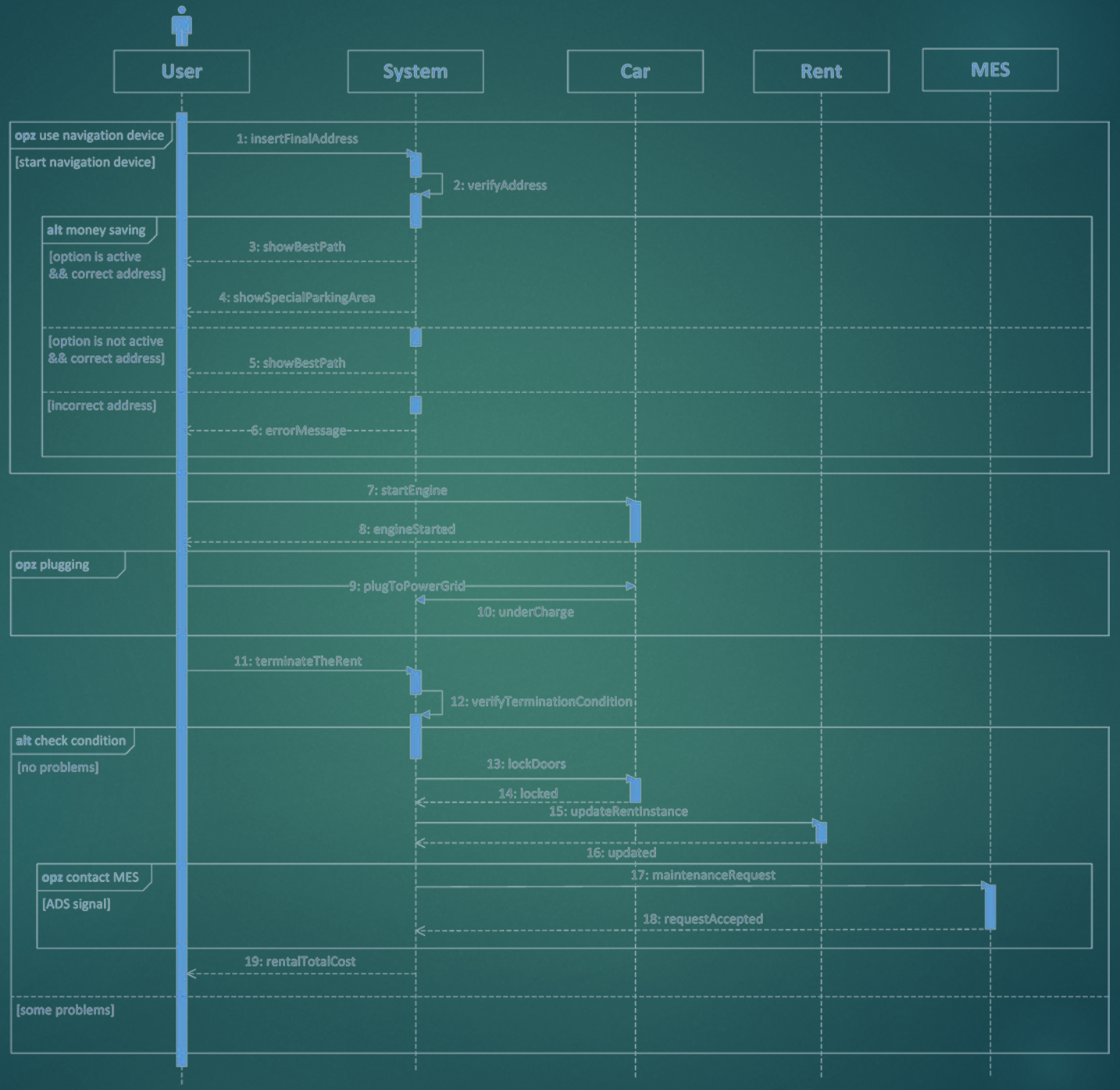

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted

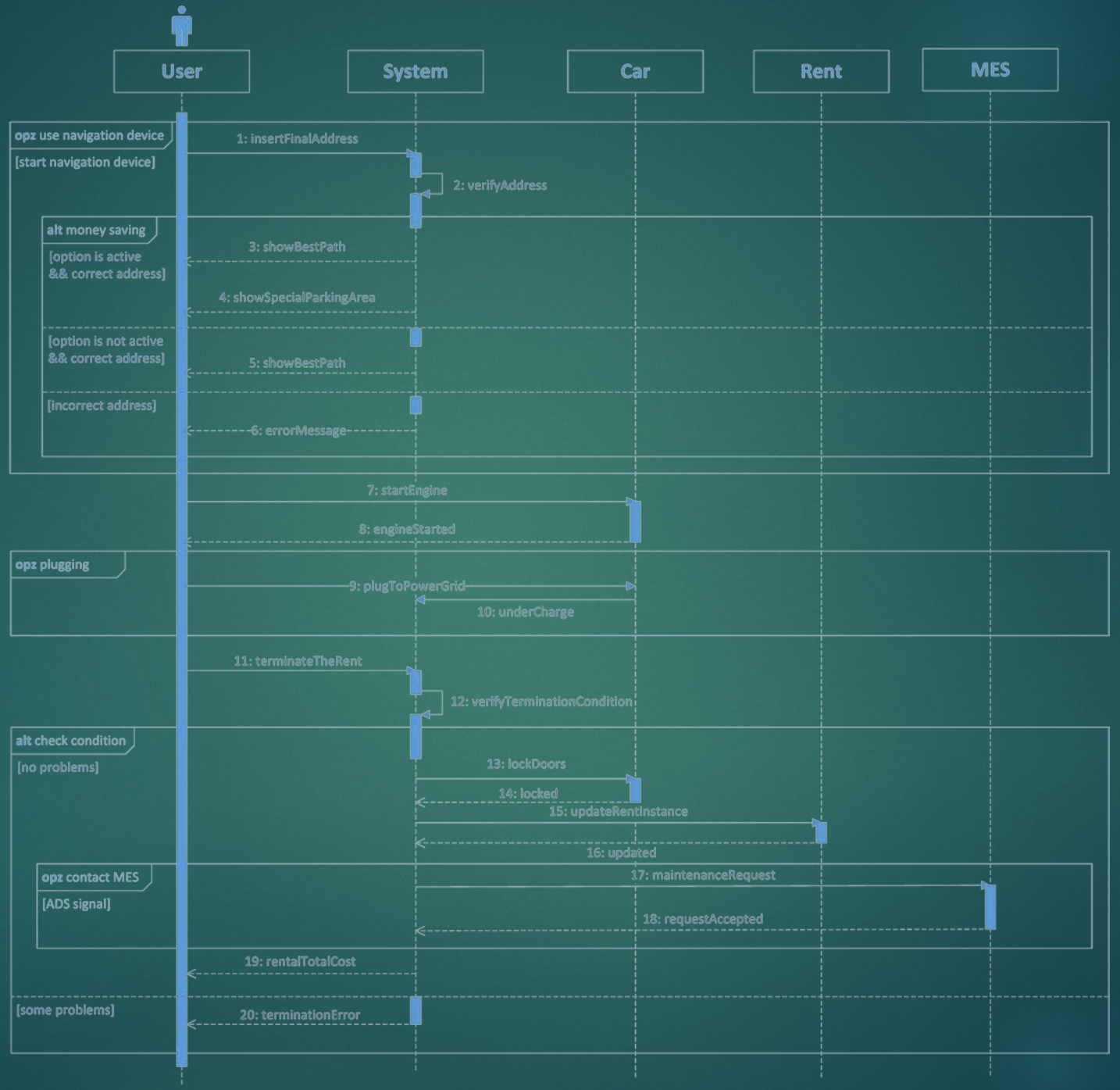

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted
opz plugging
9: plugToPowerGrid
10: underCharge


User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted
opz plugging
9: plugToPowerGrid
10: underCharge
11: terminateTheRent
12: verifyTerminationCondition
alt check condition
[no problems]
[some problems]

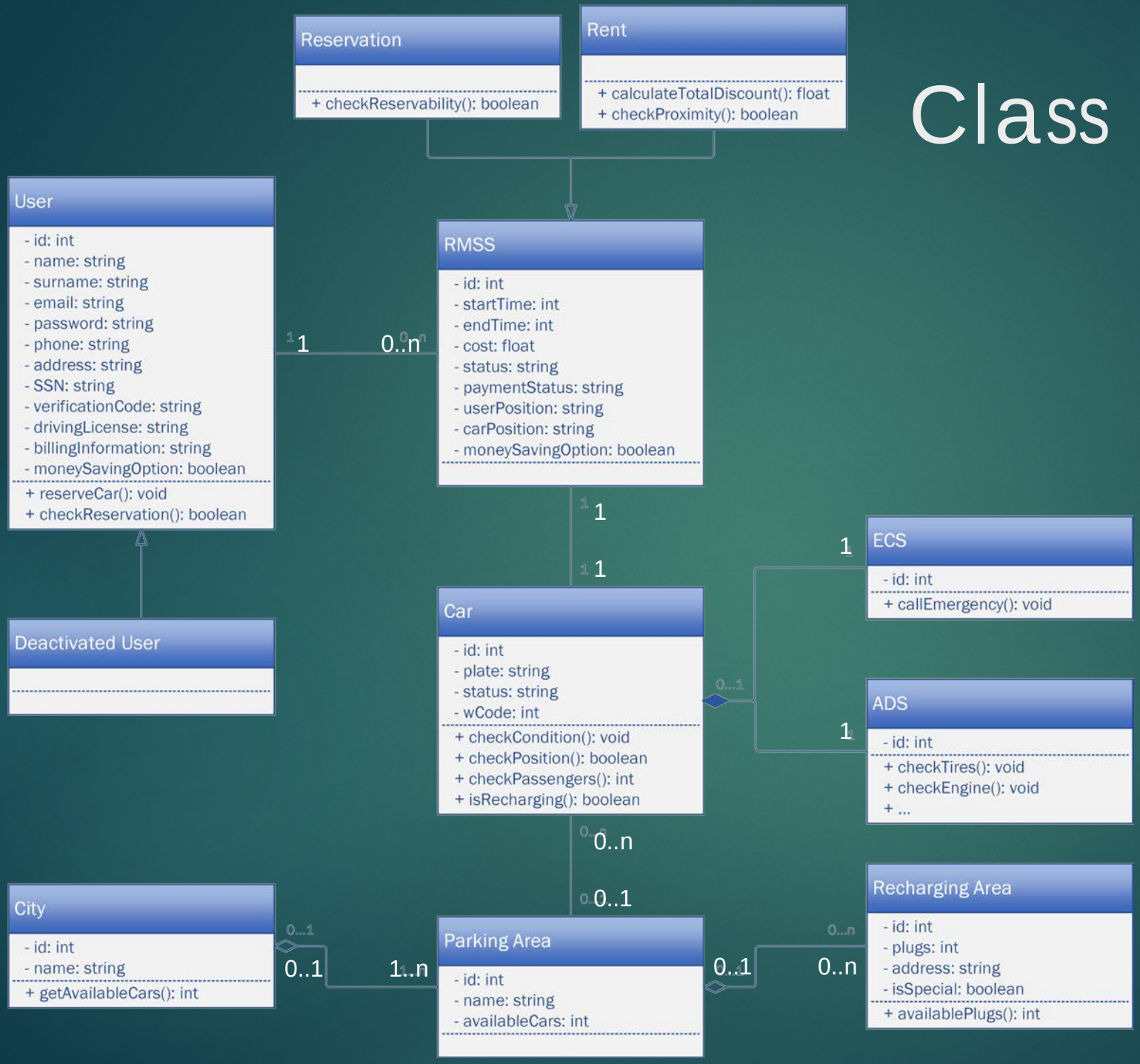

User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted
opz plugging
9: plugToPowerGrid
10: underCharge
11: terminateTheRent
12: verifyTerminationCondition
alt check condition
[no problems]
13: lockDoors
14: locked
15: updateRentInstance
16: updated
[some problems]


User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted
opz plugging
9: plugToPowerGrid
10: underCharge
11: terminateTheRent
12: verifyTerminationCondition
alt check condition
[no problems]
13: lockDoors
14: locked
15: updateRentInstance
16: updated
opz contact MES
[ADS signal]
17: maintenanceRequest
18: requestAccepted
[some problems]


User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted
opz plugging
9: plugToPowerGrid
10: underCharge
11: terminateTheRent
12: verifyTerminationCondition
alt check condition
[no problems]
13: lockDoors
14: locked
15: updateRentInstance
16: updated
opz contact MES
[ADS signal]
17: maintenanceRequest
18: requestAccepted
19: rentalTotalCost
[some problems]


User
System
Car
Rent
MES
opz use navigation device
[start navigation device]
1: insertFinalAddress
2: verifyAddress
alt money saving
[option is active
&& correct address]
3: showBestPath
4: showSpecialParkingArea
[option is not active
&& correct address]
5: showBestPath
[incorrect address]
6: errorMessage
7: startEngine
8: engineStarted
opz plugging
9: plugToPowerGrid
10: underCharge
11: terminateTheRent
12: verifyTerminationCondition
alt check condition
[no problems]
13: lockDoors
14: locked
15: updateRentInstance
16: updated
opz contact MES
[ADS signal]
17: maintenanceRequest
18: requestAccepted
19: rentalTotalCost
[some problems]
20: terminationError

Class Diagram



Non-Functional Requirements

The **non-functional requirements** are those related to the quality of the system to be implemented (Quality of Service, QoS), regardless of the application domain.

- **Security**: each user will only be able to access functionality that competes to his category. It is, therefore, necessary to provide an **authentication** method: it will be used the symmetric **username/password** one. Furthermore, each credential will not be stored in clear text, and client/server communications will be protected by asymmetric encryption.
- **Portability**: the system must be **compatible** with all hardware and software platforms to reach as many users as possible. This requirement could be satisfied, for example, by realizing the system as a web application: in this way the only assumptions that must be met by users are the availability of a web browser and an Internet connection.
- **Stability and Reliability**: The system must notify the user the result of each his/her transaction request while maintaining an optimal level of reliability. Therefore it will be able to face with the possible loss of connection between clients and the central server by ensuring **atomicity** of all operations.
- **Performance**: the timing of getting the information must be *acceptable* for the users.
- **Concurrency Management**: the system must ensure **data consistency** while dealing with multiple concurrent accesses to the same resource by **multiple users** authenticated to the system.
- **Graphical User Interfaces** (**GUIs**): The system has to include several graphical user interfaces: **intuitive**, complete and exceptionally **easy to use**, that will give the user access to the system's features.

Functional Requirements

The **functional requirements** are the functionalities that the system must necessarily have to work properly.

They describe the interactions between the system developed and the external environment independently from the implementation.

- **Registration** of a guest to the system:

  (a) The system has to guarantee the registration to a new user by creating a new account.

- **Login** of a user into the system:

  (a) The system has to allow the login to an already registered user when he types the correct username and the password in the login form.

- Find the location of **available cars** in a specified area:

  (a) The system has to allow both guests and users to specify the address where they want to locate an available car or to use their position;

  (b) The system has to guarantee that a car is showed on the map if and only if it is available (that means that every available car is showed and that every showed car is available);

  (c) The system has to guarantee that the car's position is regularly showed up to date.

Functional Requirements

- **Book** a car with the possibility to cancel the reservation:

  (a) The system has to guarantee that only a user (i.e. not a guest) can reserve a car;

  (b) The system must be able to check the position of the user;

  (c) The system has to ensure that the same car cannot be reserved at the same time by different users;

  (d) The system has to guarantee that any user can have at most an active reservation;

  (e) When a reservation is activated, the system starts a countdown at the end of which the reservation expires if the user did not pick-up the reserved car;

  (f) The system has to guarantee that every reservable car is available;

  (g) The system has to ensure that only a reservable car can be reserved;

  (h) The system has to guarantee that when a user reserves a car, the latter is no more available;

  (i) The system has to ensure that only an active reservation can expire or be canceled;

  (j) The system has to make sure that when a reservation expires or is revoked, the car becomes available again;

Functional Requirements

- **Open** his/her reserved car;

  (a) The system must be able to check the proximity of the user to the reserved car;

  (b) The system must be able to check if the money saving option is enabled for the current reservation;

  (c) The system has to allow the user to insert the verification code;

  (d) The system has to unlock the reserved car only if the entered verification code is correct.

- **End** the rental:

  (a) The system has to allow the user to communicate that he/she wants to terminate the rental;

  (b) The system has to allow the user to end the rental only if he is in a safe parking area;

  (c) The system has to guarantee that when a reservation is ended, the car becomes available again.

Alloy

```
module PowerEnjoy

//SIG

sig Stringa {}

sig Float {
    leftPart : one Int,
    rightPart : one Int
} {
    leftPart > 0
    rightPart > 0
}

sig User {
    id : one Int,
    name : one Stringa,
    surname : one Stringa,
    email : one Stringa,
    password : one Stringa,
    phone : one Stringa,
    address : one Stringa,
    SSN : one Stringa,
    verificationCode : one Stringa,
    drivingLicence : one Stringa,
    billingInformation:  one BillingInformation,
    moneySavingOption : one Bool,
    request : set RMSS
} {
    id > 0
}

sig Car {
    id : one Int,
    plate : one Stringa,
    wCode : one Int,
    ecs : one ECS,
    ads : one ADS,
    status : one CarStatus
} {
    id > 0
    wCode > 0
}
```

```
abstract sig RMSS {
    id : one Int,
    startTime : one Int,
    endTime : lone Int,
    cost : one Float,
    status : one RequestStatus,
    paymentStatus : one PaymentStatu
    userID : one Int,
    userPosition : one Stringa,
    carPosition : one Stringa,
    mSavingOption : one Bool,
    car : one Car,
    user : one User
} {
    id > 0
    startTime > 0
    endTime > 0
    endTime = none or endTime > 0
    userID > 0
    endTime > startTime
}

sig Reservation extends RMSS {}

sig Rent extends RMSS {}

sig DeactivatedUser extends User {}

sig ECS {
    id : one Int,
} {
    id > 0
}

sig ADS {
    id : one Int
} {
    id > 0
}
```

```
    id : one Int,
    name : one Stringa,
    availableCars : one Int,
    car : set Car,
    rechargingArea : set Rechar
} {
    id > 0
    availableCars > 0
}

sig City {
    id : one Int,
    name : one Stringa,
    parkingArea : set ParkingAr
} {
    id > 0
}

sig RechargingArea {
    id : one Int,
    plugs : one Int,
    address : one Stringa,
    isSpecial : one Bool
} {
    id > 0
    plugs > 0
}
```

```
// ENUMS

enum Bool {
    TRUE,
    FALSE
}

enum BillingInformation {
    CONFIRMED,
    NOTCONFIRMED
}

enum PaymentStatus {
    ACCEPTED,
    PENDING,
    DENIED
}

enum CarStatus {
    AVAILABLE,
    RESERVED,
    UNAVAILABLE,
    INUSE
}

enum RequestStatus {
    ACTIVE,
    EXPIRED
}
```

Alloy

```
// FACTS

// In any city there is at least a parking area
fact atLeastAParkingArea {
    #ParkingArea ≥ 1
}

// In any parking area there could be zero or more recharging area
fact presenceOfRechargingArea {
    #RechargingArea ≥ 0
}

// In any parking area there could be zero or more cars
fact presenceOfCars {
    #Car ≥ 0
}

// No ECS with the same ID
fact noDuplicatedECS {
    (no ecs1 , ecs2 : ECS | ecs1.id = ecs2.id and ecs1 ≠ ecs2)
}

// No ADS with the same ID
fact noDuplicatedADS {
    (no ads1 , ads2 : ADS | ads1.id = ads2.id and ads1 ≠ ads2)
}

// The same ECS cannot be used by two different Cars
fact theSameECSCannotBeUsedByDifferentCars {
    no disj c1, c2 : Car | c1.ecs = c2.ecs
}

// The same ADS cannot be used by two different Cars
fact theSameADSCannotBeUsedByDifferentCars {
    no disj c1, c2 : Car | c1.ads = c2.ads
}
```

```
// No Duplicated Users
fact noDuplicatedUser {
    (no u1 , u2 : User | u1.id = u2.id and u1 ≠ u2) and
    (no u1 , u2 : User | u1.email = u2.email and u1 ≠ u2) and
    (no u1 , u2 : User | u1.SSN = u2.SSN and u1 ≠ u2) and
    (no u1 , u2 : User | u1.drivingLicence = u2.drivingLicence and u1 ≠ u2)
}

// No Users with NOTCONFIRMED Billing Information
fact noUserWithNotConfirmedBilling {
    no u : User | u.billingInformation = NOTCONFIRMED
}

// No Cities with the same ID
fact noDuplicatedCities {
    no c1 , c2 : City | c1.id = c2.id and c1 ≠ c2
}

// No Parking Areas with the same ID
fact noDuplicatedParkingAreas {
    no pa1 , pa2 : ParkingArea | pa1.id = pa2.id and pa1 ≠ pa2
}

// No Recharging Areas with the same ID
fact noDuplicatedRechargingAreas {
    no ra1 , ra2 : RechargingArea | ra1.id = ra2.id and ra1 ≠ ra2
}
```

Alloy

```
// FACTS

// When a car is RENTED the related RENT is ACTIVE and viceversa
fact aRentedCarIsRelatedToAnActiveRent {
    (all c : Car | c.status = INUSE implies
    (one ren : Rent | ren.car = c and ren.status = ACTIVE) and
    (no res : Reservation | res.car = c and res.status = ACTIVE) )
    and
    (all ren : Rent | ren.status = ACTIVE implies
    (one c : Car | ren.car = c and c.status = INUSE)
    )
}

// When a car is RESERVED the related RESERVATION is ACTIVE and viceversa
fact aReservedCarIsRelatedToAnActiveReservation {
    (all c : Car | c.status = RESERVED implies
    (one res : Reservation | res.car = c and res.status = ACTIVE) and
    (no ren : Rent | ren.car = c and ren.status = ACTIVE)
    ) and
    (all res : Reservation | res.status = ACTIVE implies
    (one c : Car | res.car = c and c.status = RESERVED)
    )
}

// When a car is UNAVAILABLE, it cannot be RESERVED nor RENTED
fact noUnavailableReservedCar {
    all c : Car | c.status = UNAVAILABLE implies (
    (no res : Reservation | res.status = ACTIVE and res.car = c) and
    (no ren : Rent | ren.status = ACTIVE and ren.car = c)
  )
}

// When a request is ACTIVE the status of the payment is PENDING
fact pendingPaymentForActiveRequest {
    (all r : RMSS | r.status = ACTIVE implies r.paymentStatus = PENDING)
}

// No Multiple Users for the same Request
fact noMultipleUsersForTheSameRequest {
     no disj u1, u2 : User | u1.request & u2.request ≠ none
}

// The same User cannot have two ACTIVE Requests
fact theSameUserCannotPerformTwoActiveRequests {
    no disj r1, r2 : RMSS | r1.user = r2.user and
          r1.status = ACTIVE and r2.status = ACTIVE
}

// The same User cannot start two Request contemporary
fact noSimultaneousActions {
    no disj r1, r2 : RMSS | r1.user = r2.user and r1.startTime = r2.startTime
}

// Relation between deactivated users and active requests (reservation or rent)
fact noActiveRequestForDeactivatedUser {
    (all dU : DeactivatedUser | no r : dU.request | (r.status = ACTIVE))
      // No deactivated users can have an active request
}

//Consistency of the MoneySavingOption for the ACTIVE Requests
fact consistencyOfMoneySavingOptionForActiveRequests {
    (all u : User | u.request.status = ACTIVE implies
       u.moneySavingOption = u.request.mSavingOption)
}

// A Rent is possible only as a consequence of a Reservation
fact rentIsAPossibleConsequenceOfReservation{
    (all r : Rent | one res : Reservation | res in r.user.request and
              r.startTime = res.endTime and r.car = res.car)
}
```

Alloy

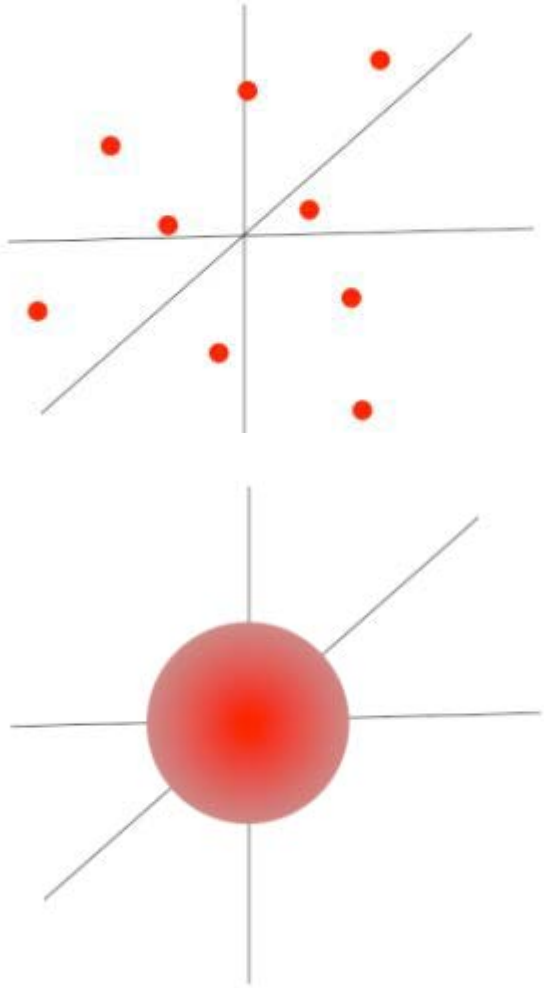
```
// ASSERTIONS

/******* WORKING *******/       //[no counterexamplefound]
// The number of active rents is equal to the number of cars in use
assert equalityOfRentedCarsAndActiveRents {
    #{r : Rent | r.status = ACTIVE} = #{c : Car | c.status = INUSE}
}
// check equalityOfRentedCarsAndActiveRents for 10


/******* WORKING *******/       //[no counterexamplefound]
// The number of active reservations is equal to the number of cars reserved
assert equalityOfReservedCarsAndActiveReservations {
    #{r : Reservation | r.status = ACTIVE} = #{c : Car | c.status = RESERVED}
}
// check equalityOfReservedCarsAndActiveReservations for 10

/******* WORKING *******/       //[no counterexamplefound]
//The number of the Reservation is greater or equal to the number of Rent
assert noRentWithoutReservation {
    all u : User |
    #{res:  Reservation | res.user = u } ≥ #{ren : Rent | ren.user = u}
}
// check noRentWithoutReservation for 10

/******* WORKING *******/       //[no counterexamplefound]
//If there is an end time for a Reservation, that must be after the start
assert requestTimeConsistency {
    all r : RMSS | r.endTime > 0 implies r.endTime > r.startTime
}
// check requestTimeConsistency for 10
```



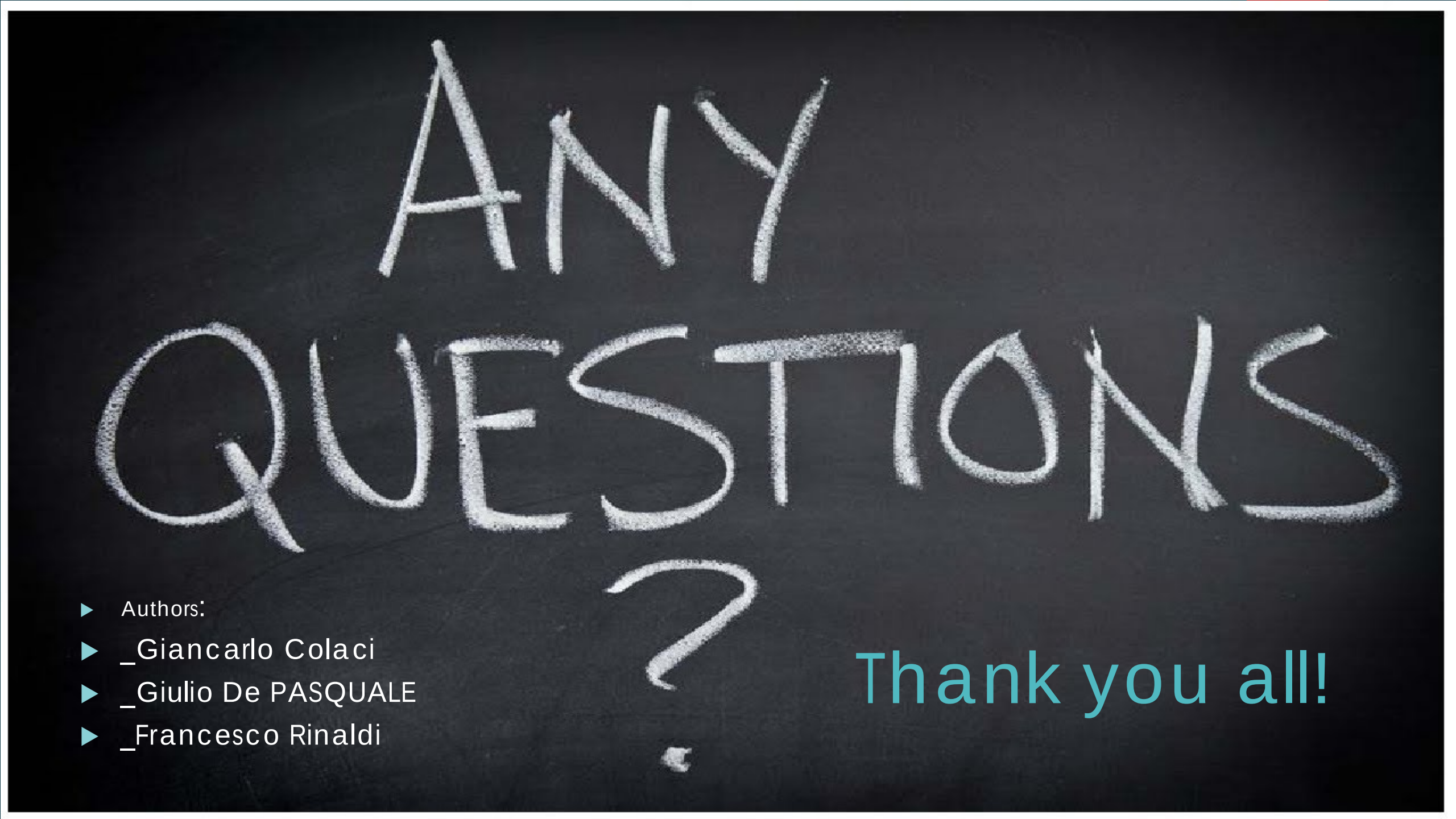
Traditional Code Testing Activities

[random research in a infinite domain]

VS

Small Scope Hypotesys

[exhaustive research in a finite subset of the domain.]



- Authors:
- _Giancarlo Colaci
- _Giulio De PASQUALE
- _Francesco Rinaldi

Thank you all!