Software Requirement Specification

For:

Prepared for:

Ardeshir Bagheri

Prepared by:

Greg Allan
Oziel Guerra
Anthony Chao
Mari Shuto
Jeff Schwanebeck
Mike Maclean
Jarrick Pang

Term project for CPSC 2301 - Langara College
Spring 2016

Table of contents

1. Introduction

a. Purpose of the system

The purpose of the software system is to create an online rental management suite for the employees of SuperRent. Additionally, the system will be used by customers of SuperRent to reserve vehicles for rent. The company needs a software solution for managing car rentals and customer information to allow for quicker transactions and quality of life for its employees as well as customers.

This document outlines the requirements of the project, and gives a detailed breakdown of the nonfunctional and functional requirements given by the company and discovered during the Requirement Elicitation stage in the project. Along with a general description of how the system will be used by specific types of users, it will also describe a few of the basic functionalities of the system. At the end of the document, we have also placed a tentative idea of what the GUI of the system will look like for each user interacting with the system.

b. Project Scope

The Software requested by the company is a car rental management system and it will be able to show all vehicles in fleet for the company, as well as many rental functions in an easy to access and step by step process. The interface of the system will be menu based for all employees, with tabs separating the different functionalities that the system has to offer. The customers interface is very limited as it is only connected to an online form for online reservations, as well as a way to track or check on reservations that have been made.

The system will also be sized to be able to work properly regardless of the hardware used by the car rental company or the browser used to access the SuperRent site by the customers. The system will be as self sufficient as possible, reminders will be sent to customers automatically, back ups will be daily during the closed hours of stores.

2. Current system

Current systems vary between web applications to desktop applications, but all seem to be fitted with some basic functionalities. These functionalities include indexing of all cars in the fleet, giving details whether they are currently rented out, being cleaned, require maintenance etc. The system also handles cost calculations and invoicing between the branch and the customers. These systems all require logins from employees, along with manager logins that have are given special permissions such as discounting or reversing sales. We have also observed a system that was almost primarily text based, almost similar to that of a shell terminal.

3. Proposed System

a. Overview

This proposed system will be able to perform all common rental functions at any car rental store, as well as display sales for the day, and the week. This system will also be able to store information of all customers allowing for easier transactions later on if the customer is recurring. Vehicles will be kept in a database holding all relevant information required on the car, and will also be able to send out reminders for certain repairs needed for cars after a certain amount of kilometers has been achieved. Overall, the system will help keep track of all of the company's bookkeeping needs, as well as provide reminders for employees and customers of certain events.

The following document has been broken into 2 main sections, the function requirements table, scenarios and use case models help explain the very basic behaviours that the users of the system will be seeing or using from it. The Object, class, and dynamic models give a general and abstract overview of how the system will be structured.

b. Functional requirements

#	Functional Requirements	Priority
	Reservations	
1	Customers will be able to filter by location and vehicle category	High
1.1	If the vehicle location is left blank, vehicles from all locations will be shown, grouped by location	High
1.2	If the vehicle category is left blank, vehicles from all categories will be shown, grouped by location	High
2	Customers will be able to view their reservations	High
2.1	Customers will be able to modify or cancel their reservation	High
3	When a customer is reserving a vehicle for rent, they will be able to specify a date range for which they will be using the vehicle	High
4	A customer can return the vehicle after office hours, and it will be assumed that the car was returned on time	High
5	Customers must provide a valid driver's license and credit card in order to rent a vehicle	High
6	If promotional codes exist, a customer will have the option of entering a code when reserving a vehicle	Medium
7	1000 SuperRent membership points will be redeemable for a 24 hour rental period	High
8	Secondary drivers for rental vehicles can be added by an employee of SuperRent	Medium
9	Reservations are limited to 2 drivers per vehicle	High
10	Reservations will be assigned a unique number to use by employees and customers to access it	High
11	In the event that a vehicle is unavailable for the time it was reserved, branch managers will be able to promote or change the reserved vehicle	High
12	Customers will be able to specify the location from which to rent, as well as the location for which to return the vehicle	High
13	Customers will have the option to receive reminders for vehicle rental dates	Medium
14	Vehicle and equipment reservations will be limited so they will not conflict	High
15	Each non-corporate customer will have only one reservation at any given time	High

16	SuperRent employees will be able to reserve vehicles for the customer if they come into the shop	High
16.1	Employees will be able to filter by everything that the customer can	High
Fleet Management - Managers Only		
17	Branch managers will be able to move cars from a "for rent" list to a "for sale" list	High
18	Vehicles on the "for sale" list will be able to be sold to customers	High
19	Vehicles on the "for sale" list will be able to be sold to corporate entities	High
20	Vehicle identification numbers will be stored in the system, and only viewable by the branch managers	High
21	Prices of vehicles on the "for sale" list are to be controlled by the branch manager	High
22	Vehicle rental rates will be able to be modified by the branch manager	High
23	Vehicle returns will be processed by SuperRent employees	High
24	Branch managers will be able to add new vehicles to the fleet	High
25	Branch managers will receive notifications about overdue vehicles	High
25.1	Branch Managers will receive notifications about car status and maintenance upon logging in	High
Store Management		
26	Store promotions are to be controlled by the branch manager	High
27	Incident reports will generated by store employees in event of an issue with a returned or rented vehicle	High
28	Incident reports will be sent to a branch manager to sign before sending vehicles to third party services	High
29	Reports of the state of the Branch's inventory will be generated nightly before closing	High
30	The System will be able to generate daily and weekly sales reports	High
System Administration		
31	Employee accounts are to be added and removed by the System Administrator	High
32	The system administrator will be able to backup the system as needed	High
33	The system will provide nightly backups while the office is closed	High
34	Employee account passwords will be changeable by the System Administrator	High
35	Any errors generated through use of the system will be written to an error file for the System Administrator to review	Medium
36	All database interactions are to be logged to a file	Medium
36.1	Employee account actions (log in/ log out) will be logged to the same file	Medium
37	The database interaction log will only be viewable by the System Administrator and Branch Managers	High
System Security		
38	Employee passwords will be limited to a character count greater than eight(8)	High
39	Employee accounts will be logged out after 20 minutes of inactivity	High
40	Account login requests will be limited to 5 attempts before locking the account temporarily	High
41	Customer credit details will be deleted after the contract has closed	High
42	During a contract's period, only the Branch Manager will be able to view the Customer's credit details	High
43	Any information inputted to the system will be formatted in such a way as that the system shall remain safe	High
44	Desktop application for SuperRent Employees and Managers	Low
Accounts and Interface		
45	Customer accounts will keep a record of all rental contracts	High
46	Contracts will be locked after vehicle return and any charges and fees have been paid	High
47	Customers can opt to sign up for a SuperRent membership	High
47.1	Customers with SuperRent accounts will be given membership points proportionate to rental cost	High
48	Corporate entities will be able to register for corporate accounts	High

49	Employees will be able to add notes to Customer accounts	High
50	Users will be given a choice to choose between French and English for the interface	Low
51	Customer accounts will be prompted to give feedback on their first visit, then every 10 to 15	Medium
52	Customers will be able to register and delete accounts from the interface	High
System Constraints		
53	The system must be implemented in Java version 7	High
54	The system must be able to run on Microsoft Windows Operating System	High

c. Non-Functional requirements

#	Non-Functional Requirements	Priority
Functionality		
1	The System will have an API that enforces current security standards	High
2	The system will require users to create a new password every 90 days	Medium
3	The communication between the API and the interface must be secure	High
3.1	The system will use TLS/SSL technology to enforce a secure connection by using HTTPS	High
3.2	The system will not expose important transaction to the customer in the website URL, to avoid manipulation by malicious actors	High
3.3	The communication between the API and the interface must account for SQL Injection and take precautions against it	High
4	Passwords will require a length of 8 characters, numbers, or symbols	High
Useability		
5	The interface for the application must be intuitive and easy to use	High
6	The application must be completely documented, including basic and advanced operations and definitions of errors	High
7	First log in for employee should have step by step instructions on use of system	Low
8	The system must respond to user input in less than thirty(30) seconds	High
9	The customer interface for the system will accommodate different screen sizes so it can be used by any internet connected device	High
Reliability		
10	The system will provide complete, restorable backups	High
11	The system will guarantee uptime, unless the event of hardware failure	High
Performance		
12	The System will support up to 30 transactions per minute	High
Supportability		
11	The API for the system will be straightforward, as to provide a way for further extension	High

d. System model

i. Scenarios

The following scenarios depict some of the most common situations that customers and employees of SuperRent, the users of TurboRent software will encounter and the flow of events for them.

Scenario Name	rentATruck
Participating Actor Instances	bob:Employee, alice:Customer db:CentralDatabase
Flow of Events	1. Alice enters SuperRent and is then attended by Bob at his computer which is running the TurboRent software 2. Alice would like to rent out a car and is provided the details required to rent a car such as: dateOfPickup, dateOfReturn, pickupLocation, returnLocation, what type of vehicle (truck), or a specific vehicle if she had in mind, any additional accessories for the vehicle such as equipment. 3. Alice makes payment for the reservation 4. Bob creates a new reservation and enters the information that Alice provided, creating a new reservation 5. Bob prints a confirmation paper that Alice would bring in on the day she picks up her rental car.

Scenario Name	pickupVehicle
Participating Actor Instances	bob:Employee, alice:Customer, db:CentralDatabase
Flow of Events	1. Alice enters SuperRent and brings her confirmation paper to Bob 2. Bob confirms through the TurboRent software that all the details provided by the confirmation paper match the details on the db. 3. Bob then brings Alice to her vehicle with keys and return papers for return. 4. Alice drives off happy

Scenario Name	returnVehicleNoDamages
Participating Actor Instances	john:Employee, alice:Customer, db:CentralDatabase
Flow of Events	1. Alice drives back to SuperRent and has it checked by John, who handles the vehicle inspections and returns. He assesses and checks to make sure that there was no damage from Alice's use of the vehicle 2. After inspection, John marks the vehicle as returned in the db with no new damages. 3. Alice returns the keys to the employee and then leaves SuperRent

Scenario Name	returnVehicleWithDamages
Participating Actor Instances	john:Employee, alice:Customer, db:CentralDatabase
Flow of Events	1. Alice drives back to SuperRent and has it checked by John, who handles the vehicle inspections and returns. He notices some damage to the front bumper 2. John creates a damage report for new damages to the vehicle and files it to the db 3. John then prints an invoice of the damages and the charges for Alice. 4. Alice returns to the SuperRent building to pay for the additional charges for vehicle damage 5. Alice then leaves SuperRent

Scenario Name	closeMembership
Participating Actor instances	alice:Customer, bob:Employee, db:CentralDatabase
Flow of Events	1. Alice comes into SuperRent to close her membership account

	2. Bob deletes Alice's record from the db

Scenario Name	editReservation
Participating Actor instances	alice:Customer, bob:Employee, db:CentralDatabase
Flow of Events	1. alice the customer enters SuperRent and asks to extend her her reservation, wishing to change her return date by 4 days. 2. bob the Employee asks for alice's reservation number or drivers licsence, opening up her account with this information and opening the specific reservation. 3. bob selects the edit reservation function on his terminal 4. bob changes the return date by adding 4 extra days on to the reservation 5. TURBORENT sends a queries to the central database, db checking if the changes can be made, that is, if the currently reserved vehicle (or same model) is available and to get the new total for the reservation 6. TURBORENT displays that the changes are possible and displays the new total for the reservation and the new remaining balance due. 7. bob tells alice the new balance due and asks if she wishes to make the changes to her reservation 8. alice says yes and bob confirms the changes on his terminal 9. TURBORENT communicates with db to update the reservation 10. alice pays the remaining balance and leaves the SuperRent location

Scenario Name	openNewMembership
Participating Actor instances	alice:Customer, bob:Employee, db:CentralDatabase
Flow of Events	1. Alice comes into SuperRent and is looking to set up a new club membership at SuperRent

	2. Bob helps Alice set up a new club membership with Alice providing the necessary details such as fullName, phoneNumber, homeAddress, email 3. Bob inputs a new clubMember record into the db.

Scenario Name	addNewEmployee
Participating Actor instances	alice:SysAdmin, bob:Employee, db:CentralDatabase
Flow of Events	1. alice, the SysAdmin, needs to create a TurboRent standard employee account for bob, a new employee in a SuperRent branch. 2. alice activates the add new employee account function on her terminal 3. alice inputs bob's information including first and last name, and email address, branch location 4. alice selects the account level for bob, a standard employee account 5. alice selects create account 6. TurboRent prompts to confirm the information 7. alice confirms and TurboRent generates password and username for bob, communicating with db, the central database to store the account information securely 8. TurboRent then sends the credentials to bob's SuperRent email address and displays to alice the account was successfully created and credentials were sent.

Scenario Name	customerDisplayTrucks
Participating Actor instances	alice:Customer, db:CentralDatabase
Flow of Events	1. alice, a potential customer opens a browser and navigates to SuperRent home page. 2. alice enters the pickup location and the dates she wants to see the available vehicles for. 3. alice selects all Trucks as the vehicle type and selects view vehicles. 4. TurboRent communicates with db, the central database, to get a list of all available trucks at the provided location for the date range. 5. the list is displayed in alice's browser window on a new web page along with an estimation of the rates for each type of truck.

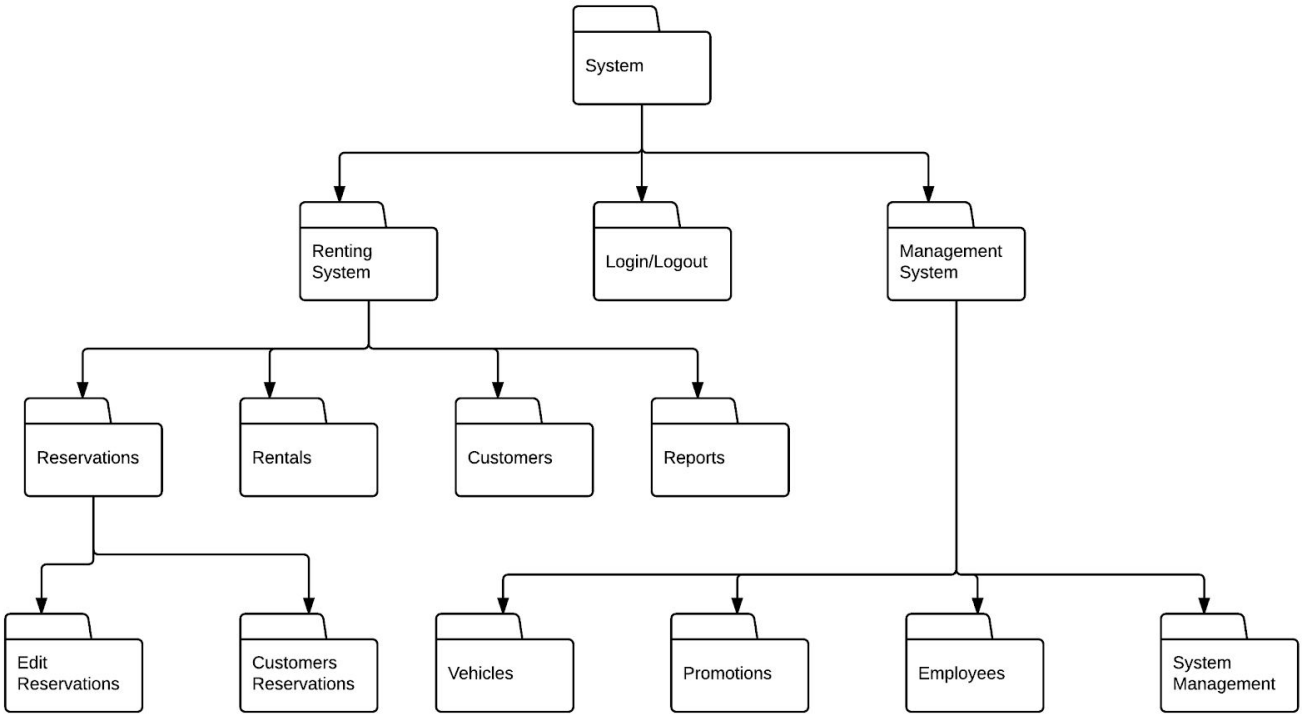
Scenario Name	failedLoginWithLockout
Participating Actor instances	bob:Employee, db:CentralDatabase
Flow of Events	1. bob, an employee is at a terminal at work in a SuperRent Branch and needs to login 2. bob enters his username and password but miss-types his username 3. TurboRent displays that username or password are invalid 4. bob realizes the error in his username and corrects this, re-entering his password. 5. TurboRent checks the username and password the db, the central database but they do not match 6. Turbo Rent displays that the password is invalid and the bob has 4 more attempts to login before being locked out of TurboRent for 2 minutes 7. bob proceeds to re-enter his password but incorrectly enters it 4 more times 8. after the last attempt, TurboRent displays the the user is locked out of their account and displays the amount of time remaining on the lockout.

ii. Use Case Models

The use cases shown by the following models and text documentation describe how users of TurboRent will utilize the required functionalities of the software system.

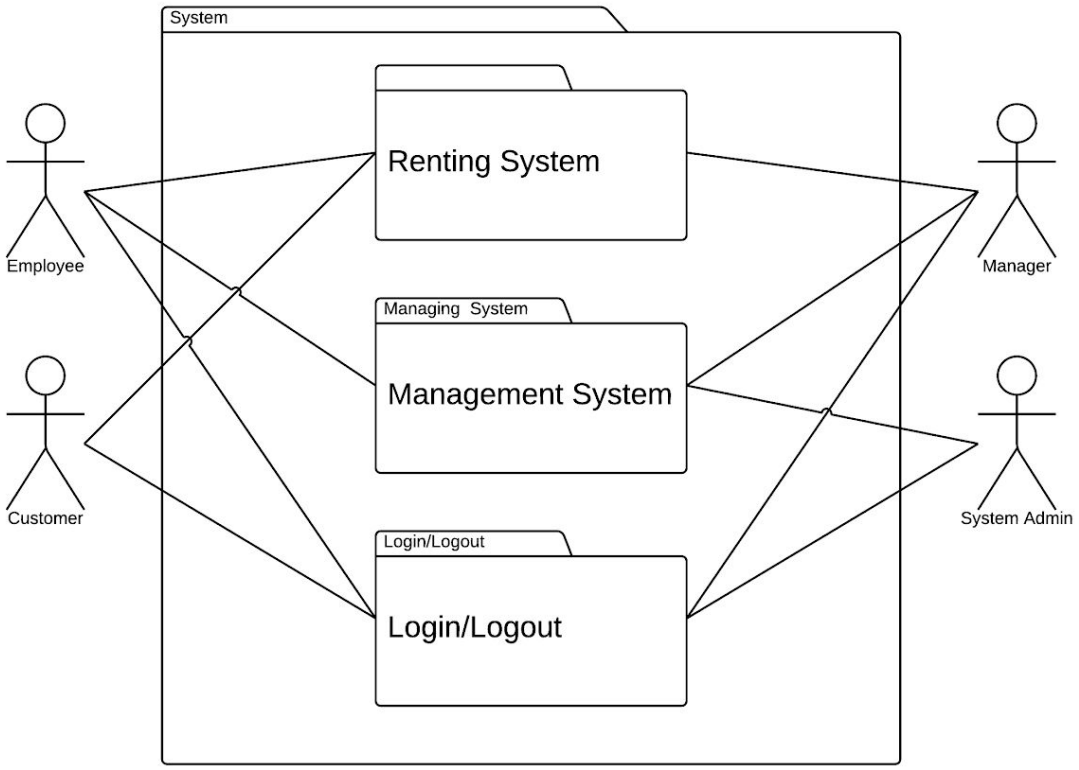
The Use Case Hierarchy:

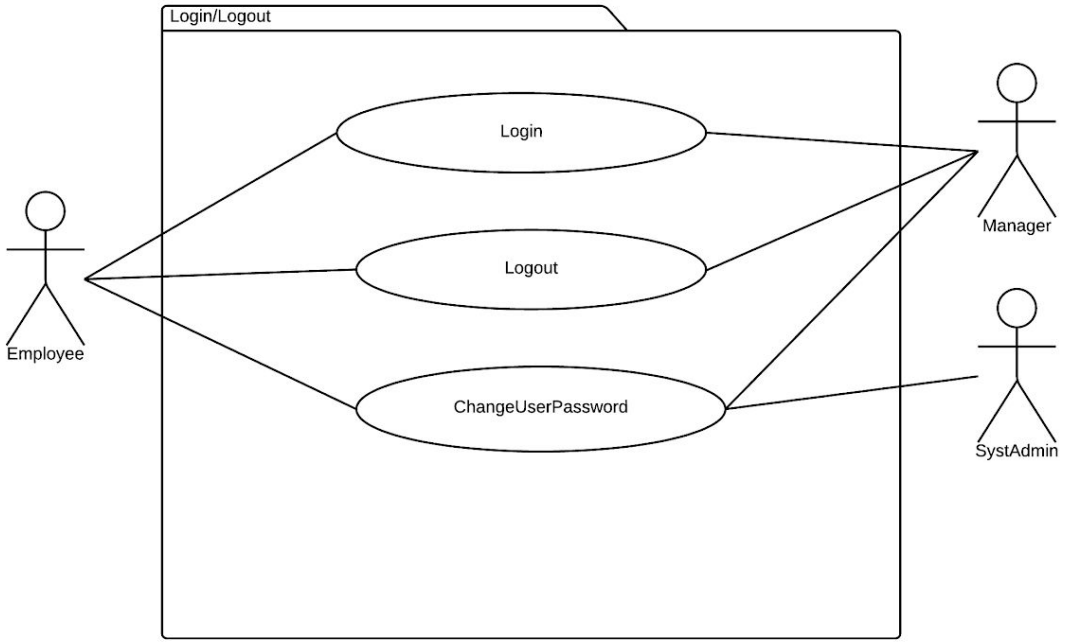
The diagram below shows how the use cases have been divided into their functionalities

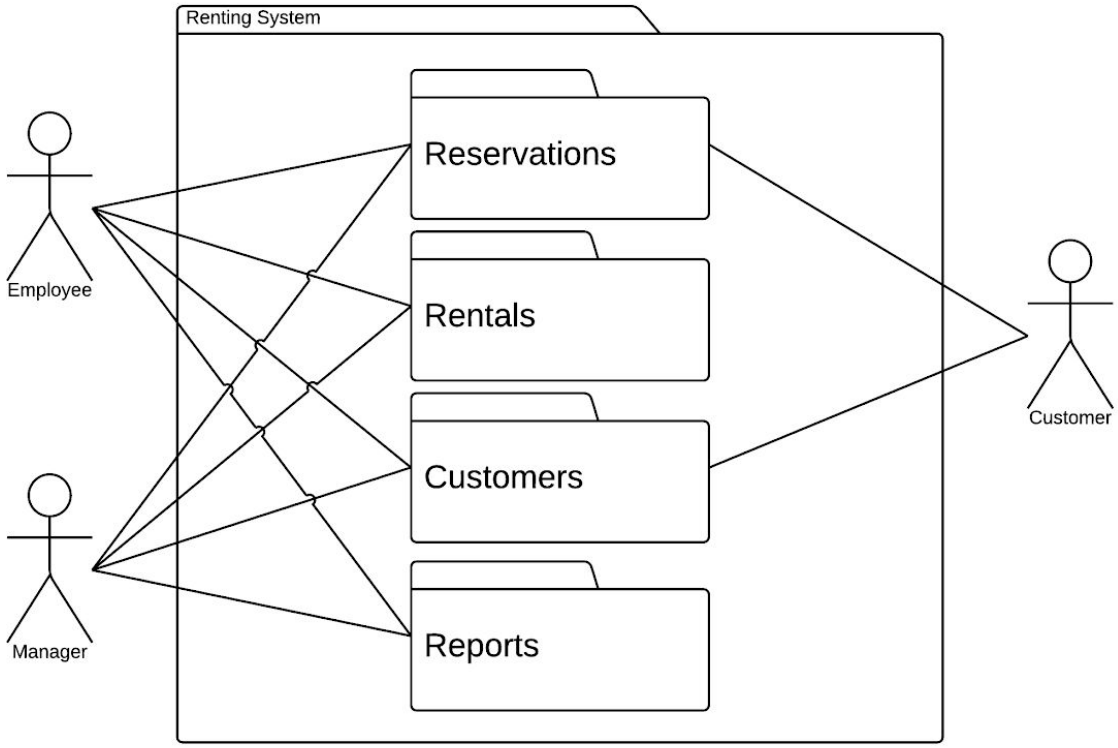


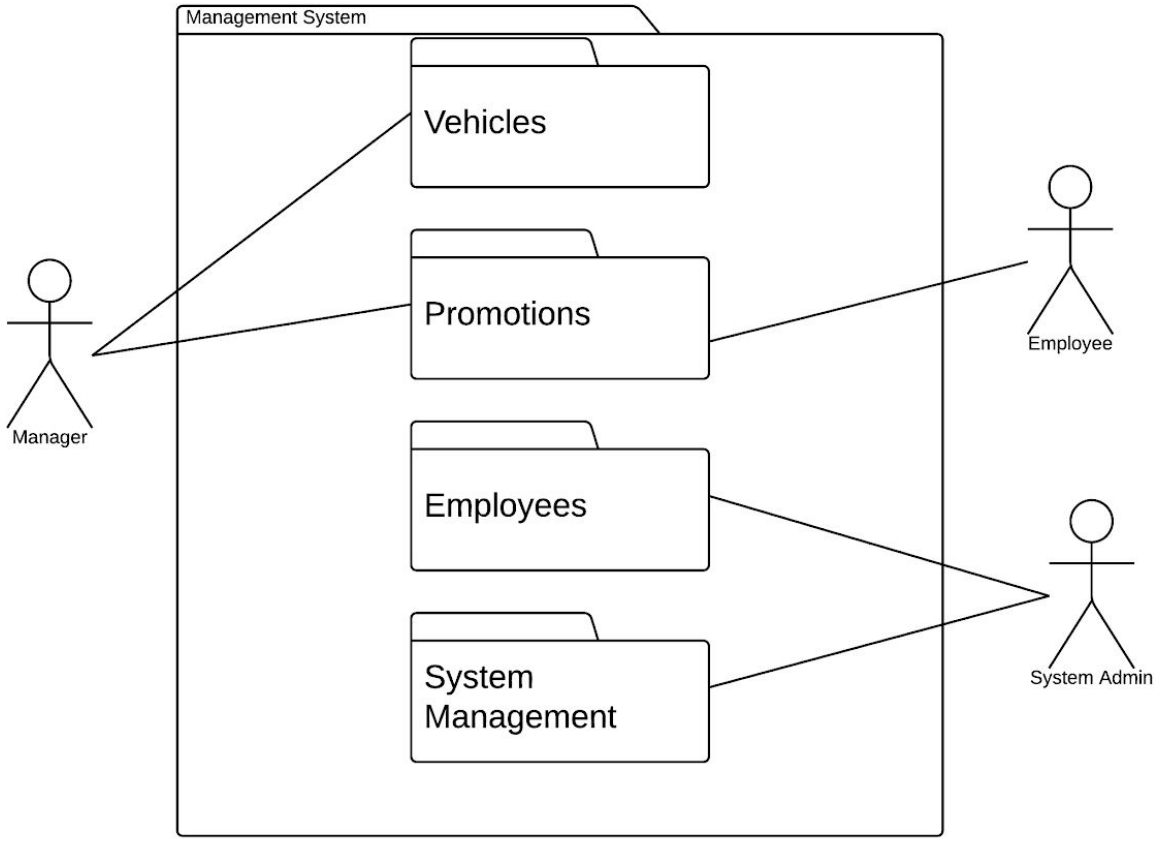
Use Case Diagrams:

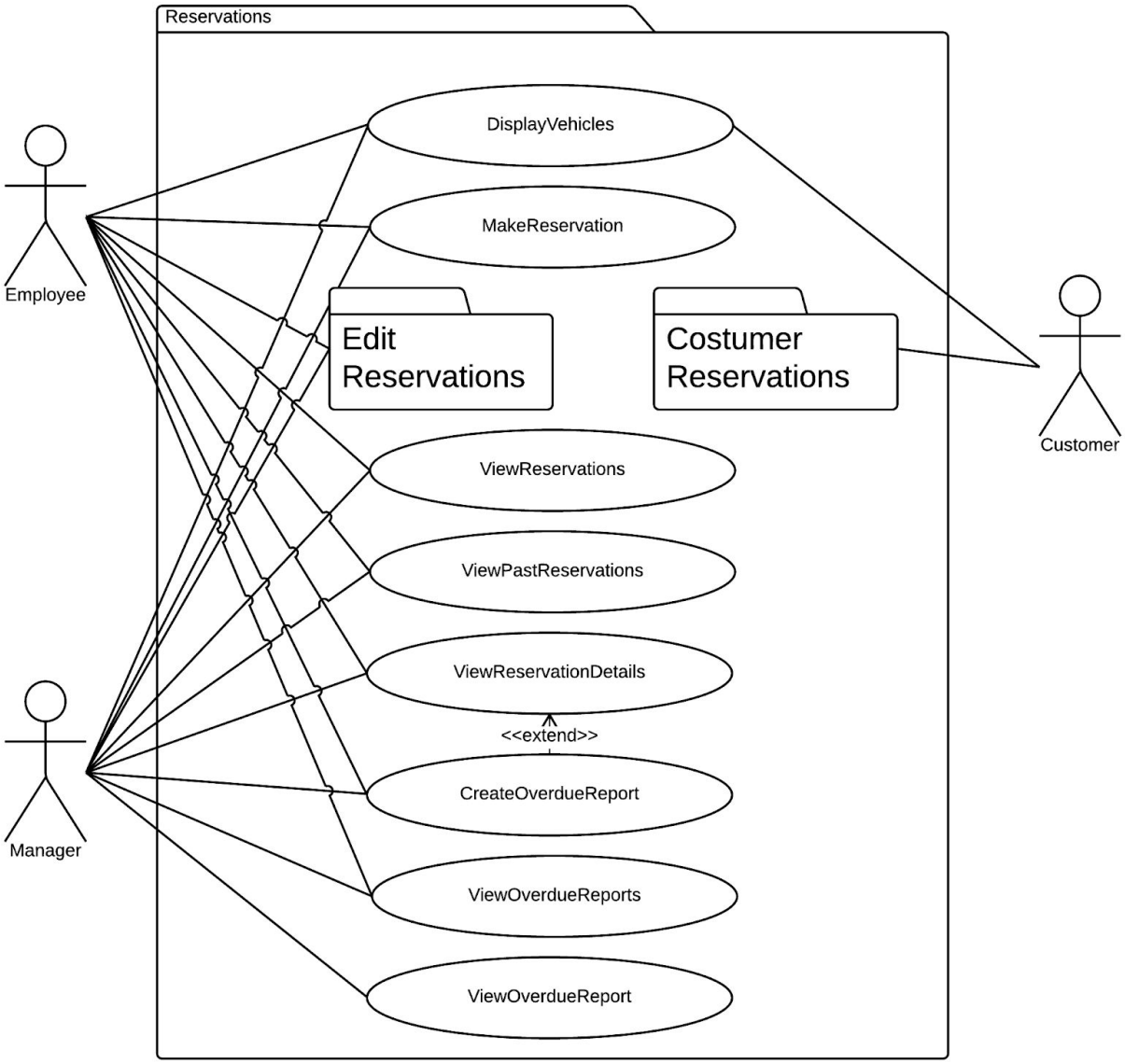
A visual representation of what uses each actor will have with the system.

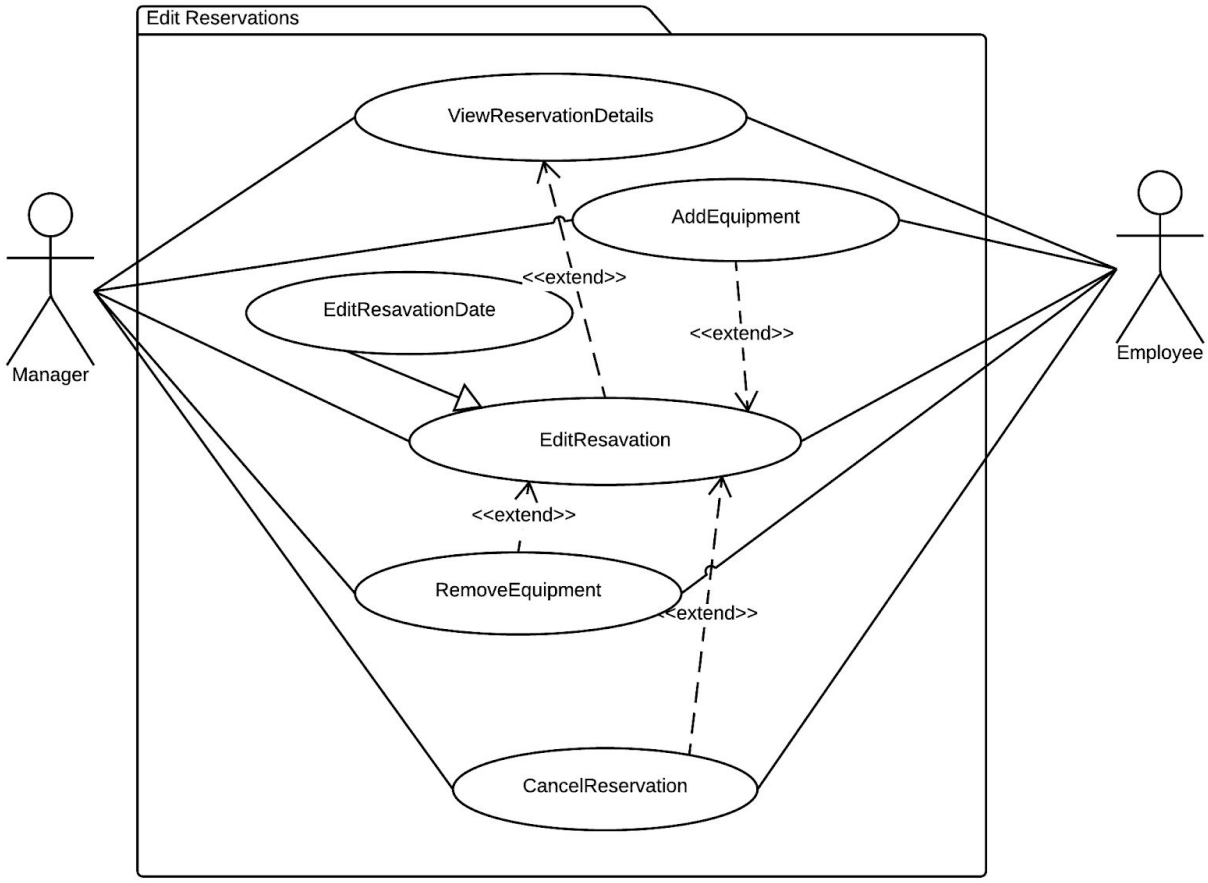

System
Renting System
Managing System
Management System
Login/Logout
Login/Logout
Employee
Customer
Manager
System Admin

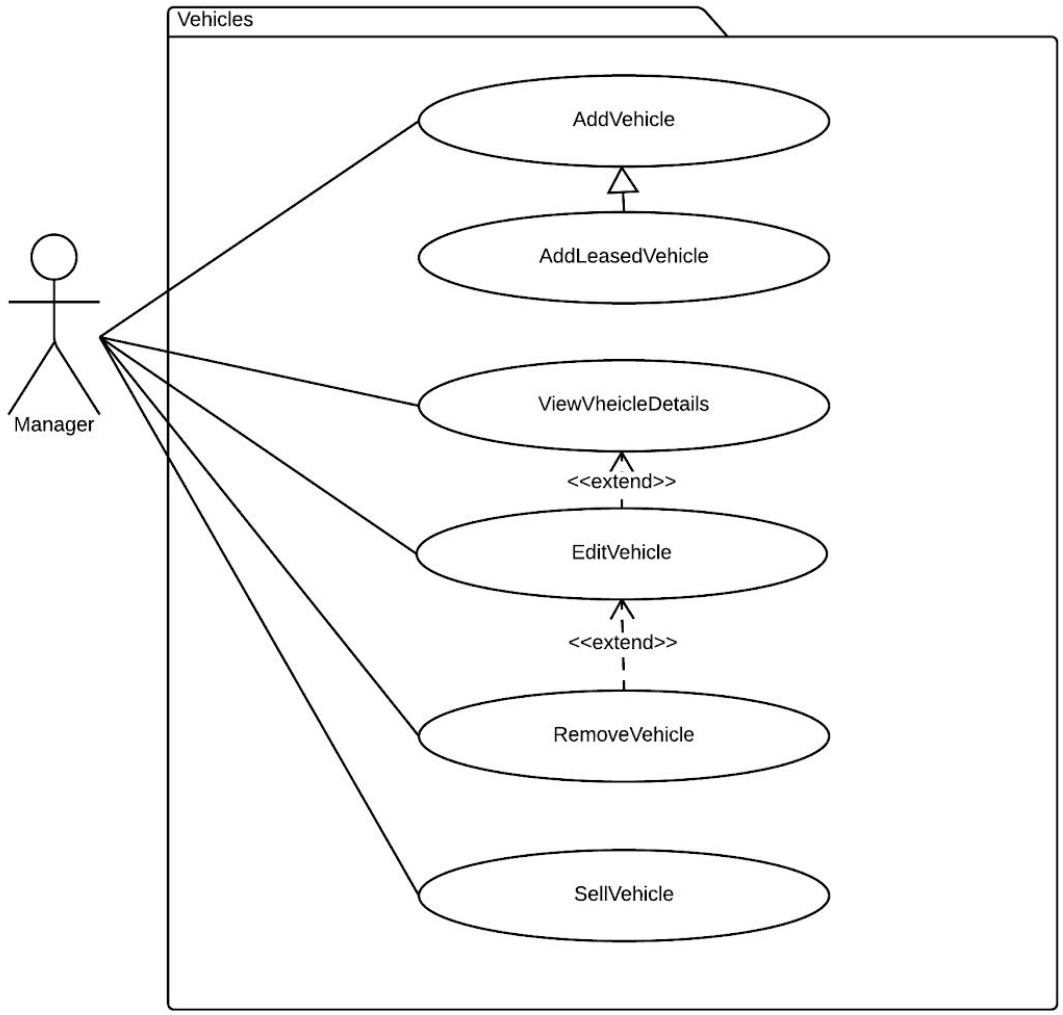

Login/Logout
Login
Logout
ChangeUserPassword
Employee
Manager
SystAdmin

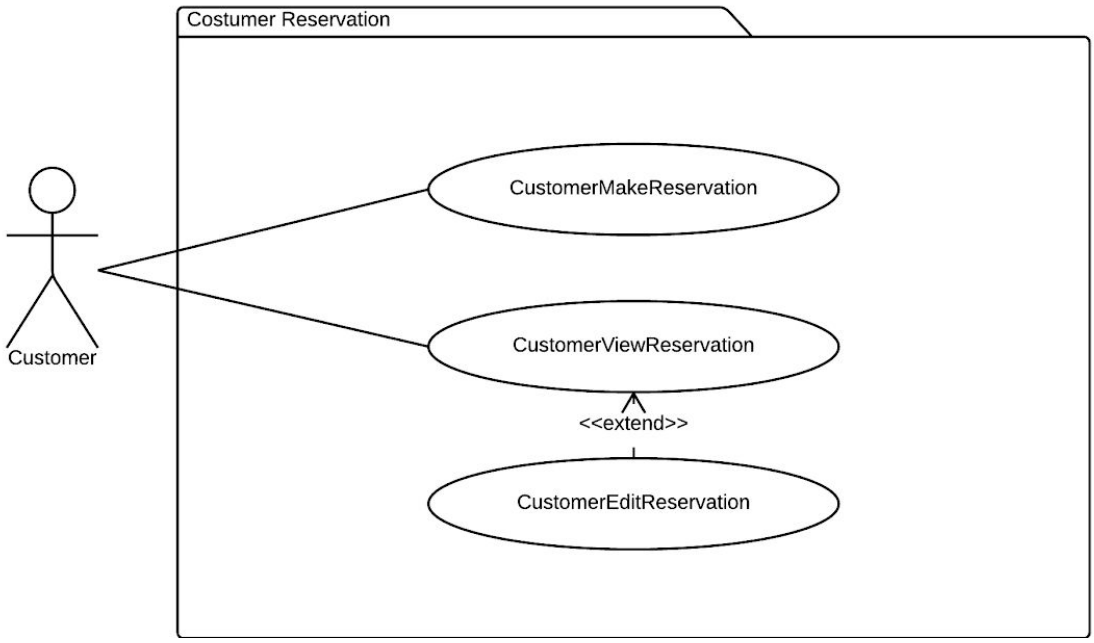

Renting System
Reservations
Rentals
Customers
Reports
Employee
Manager
Customer

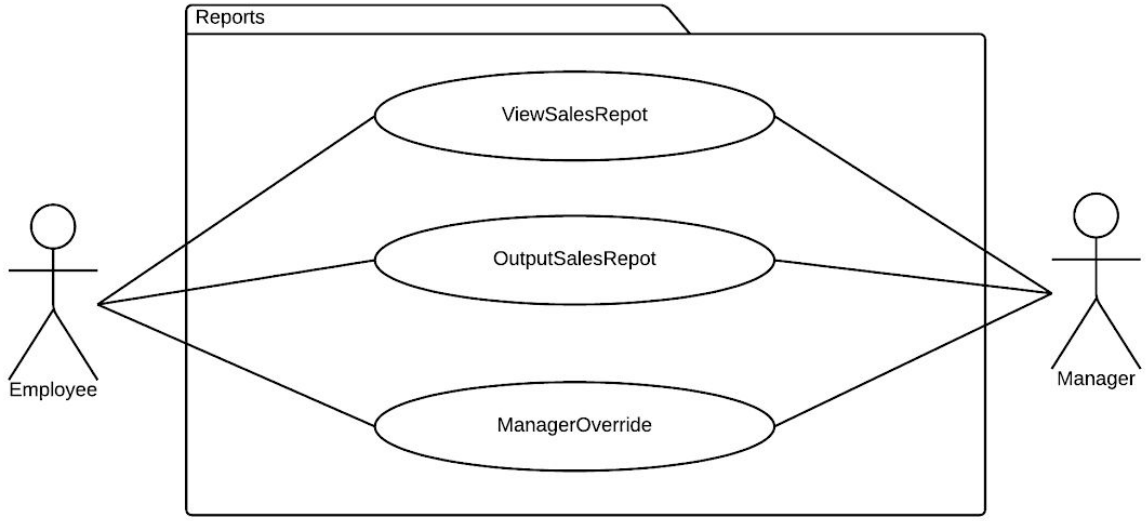

Management System
Vehicles
Promotions
Employees
System Management
Manager
Employee
System Admin

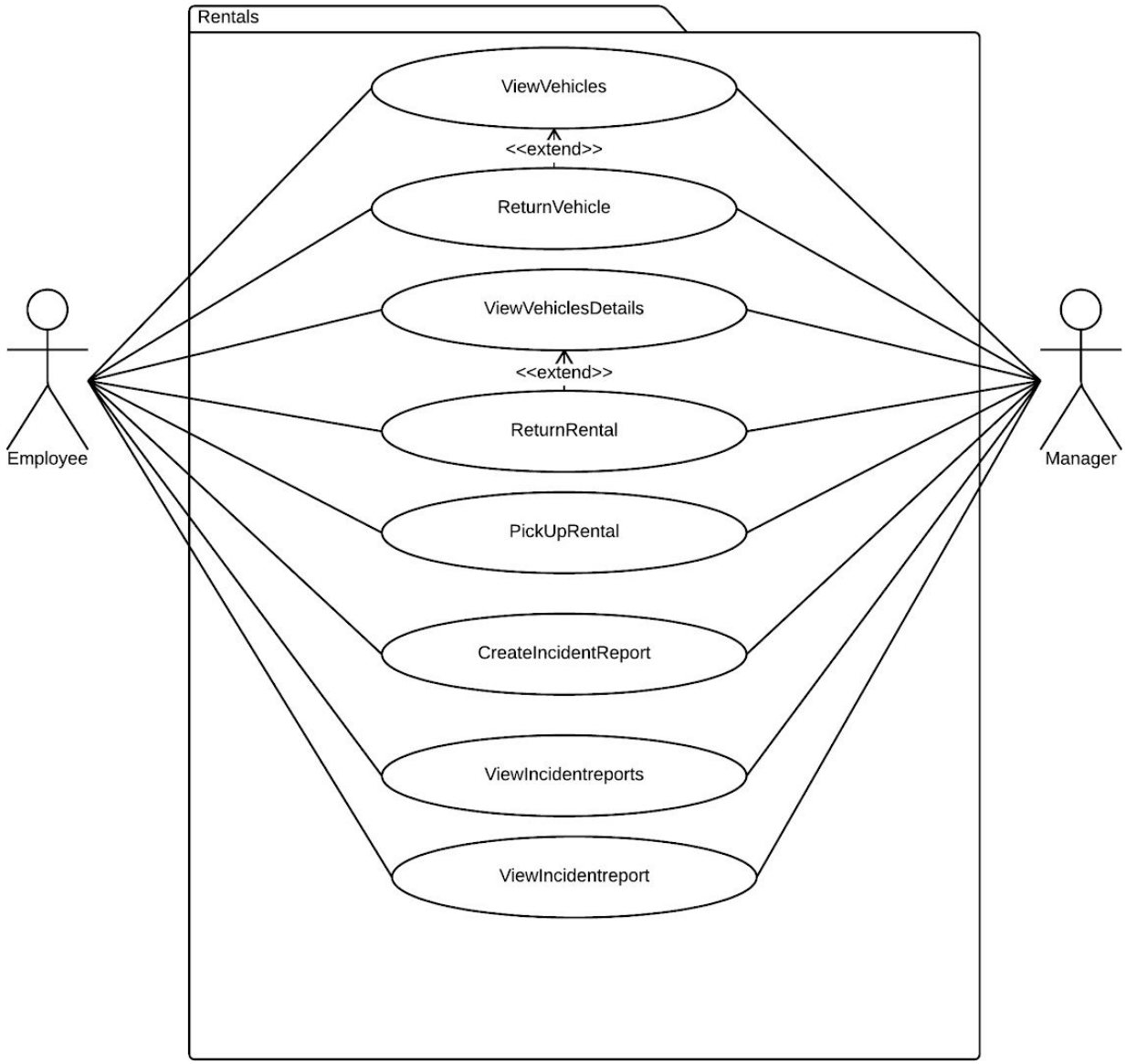

Reservations
DisplayVehicles
MakeReservation
Edit Reservations
Costumer Reservations
ViewReservations
ViewPastReservations
ViewReservationDetails
<<extend>>
CreateOverdueReport
ViewOverdueReports
ViewOverdueReport
Employee
Manager
Customer

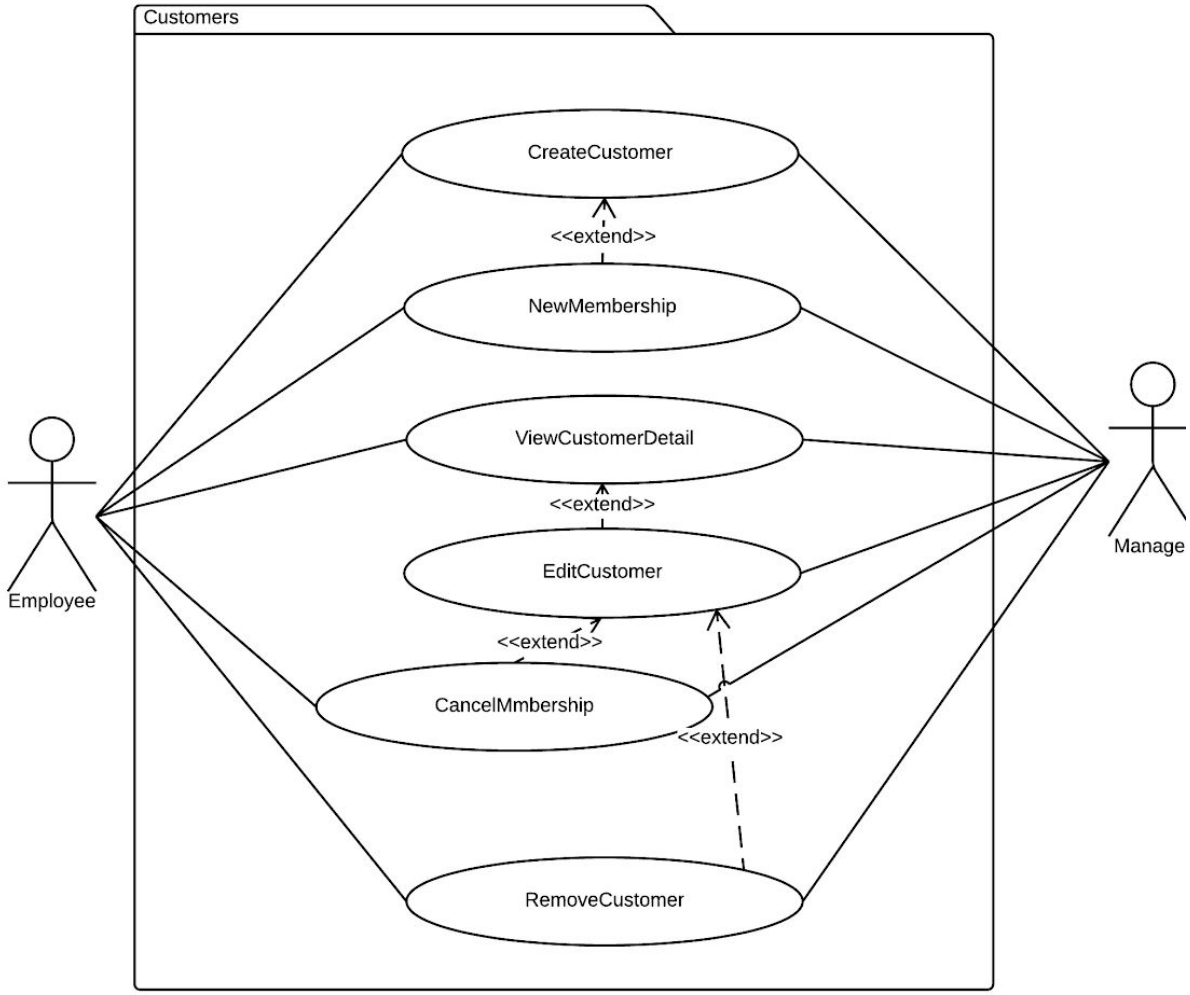

Edit Reservations
ViewReservationDetails
AddEquipment
<<extend>>
EditResavationDate
<<extend>>
Manager
Employee
EditResavation
<<extend>>
RemoveEquipment
<<extend>>
CancelReservation

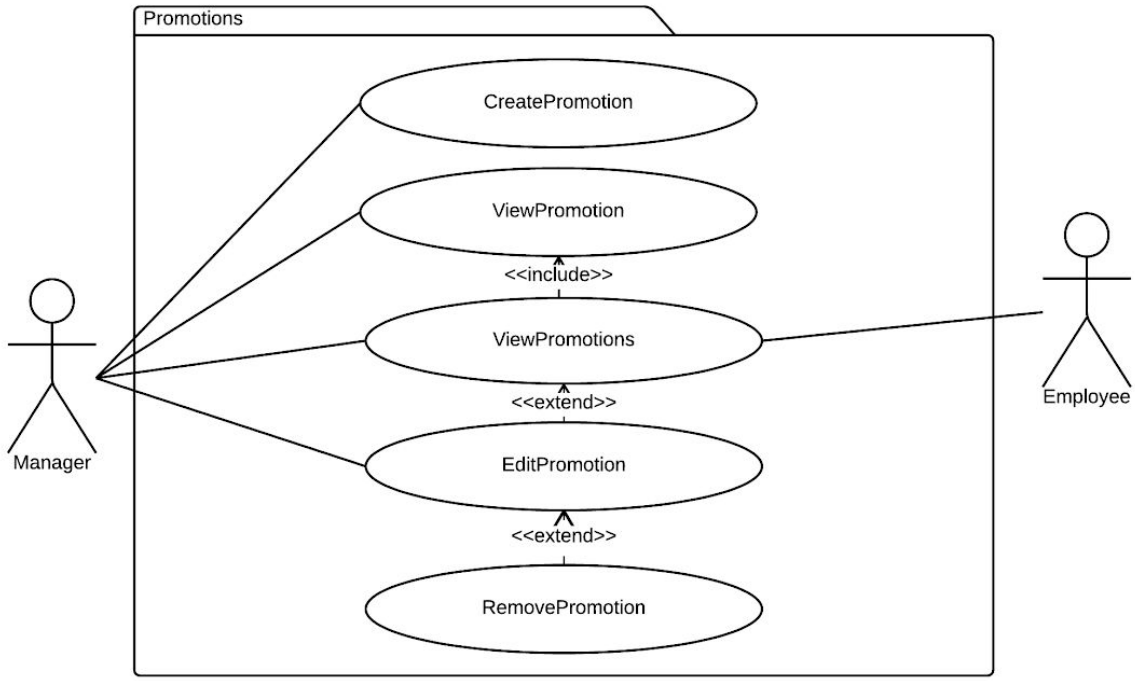

Vehicles
AddVehicle
AddLeasedVehicle
Manager
ViewVheicleDetails
<<extend>>
EditVehicle
<<extend>>
RemoveVehicle
SellVehicle

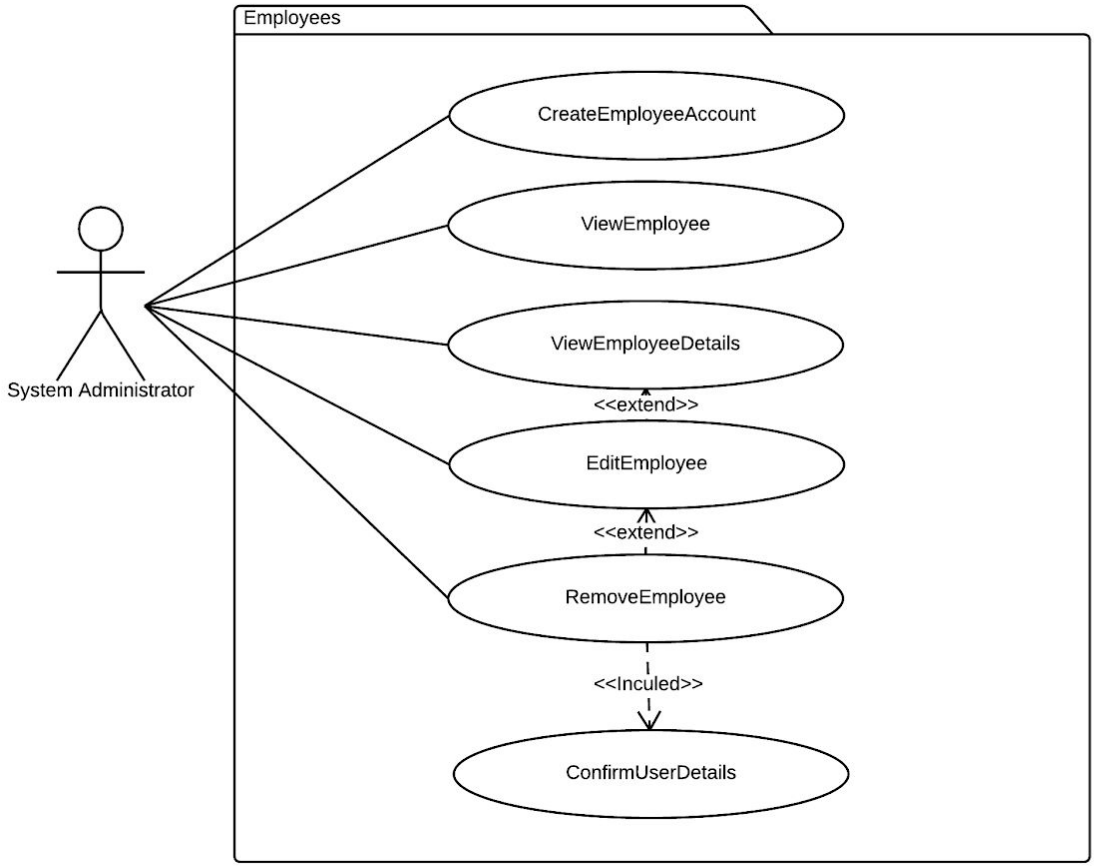

Costumer Reservation
CustomerMakeReservation
Customer
CustomerViewReservation
<<extend>>
CustomerEditReservation

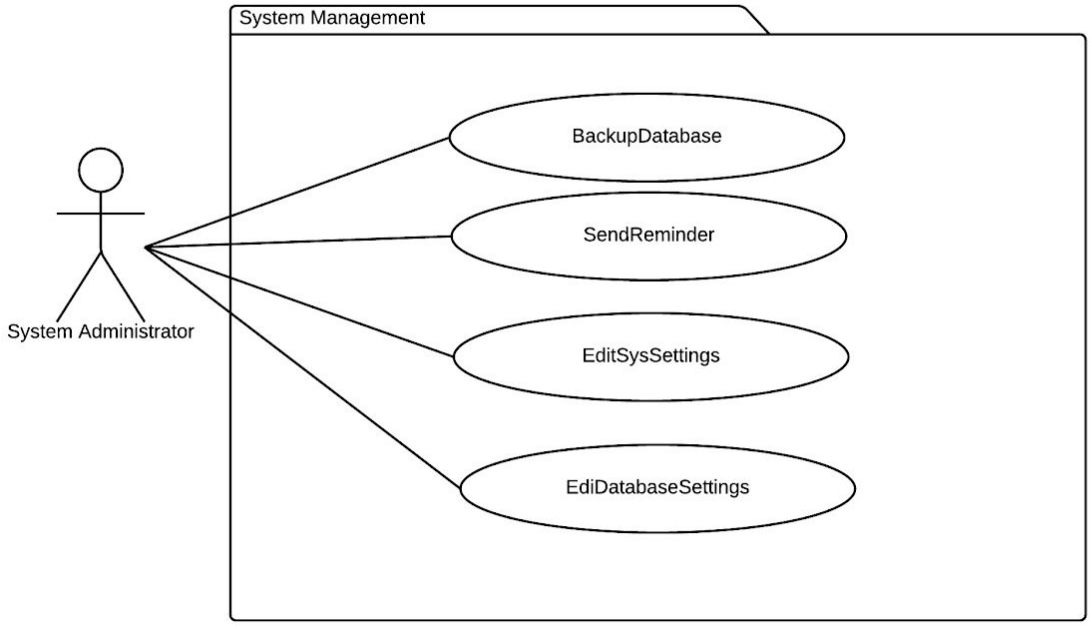

Reports
ViewSalesRepot
OutputSalesRepot
ManagerOverride
Employee
Manager

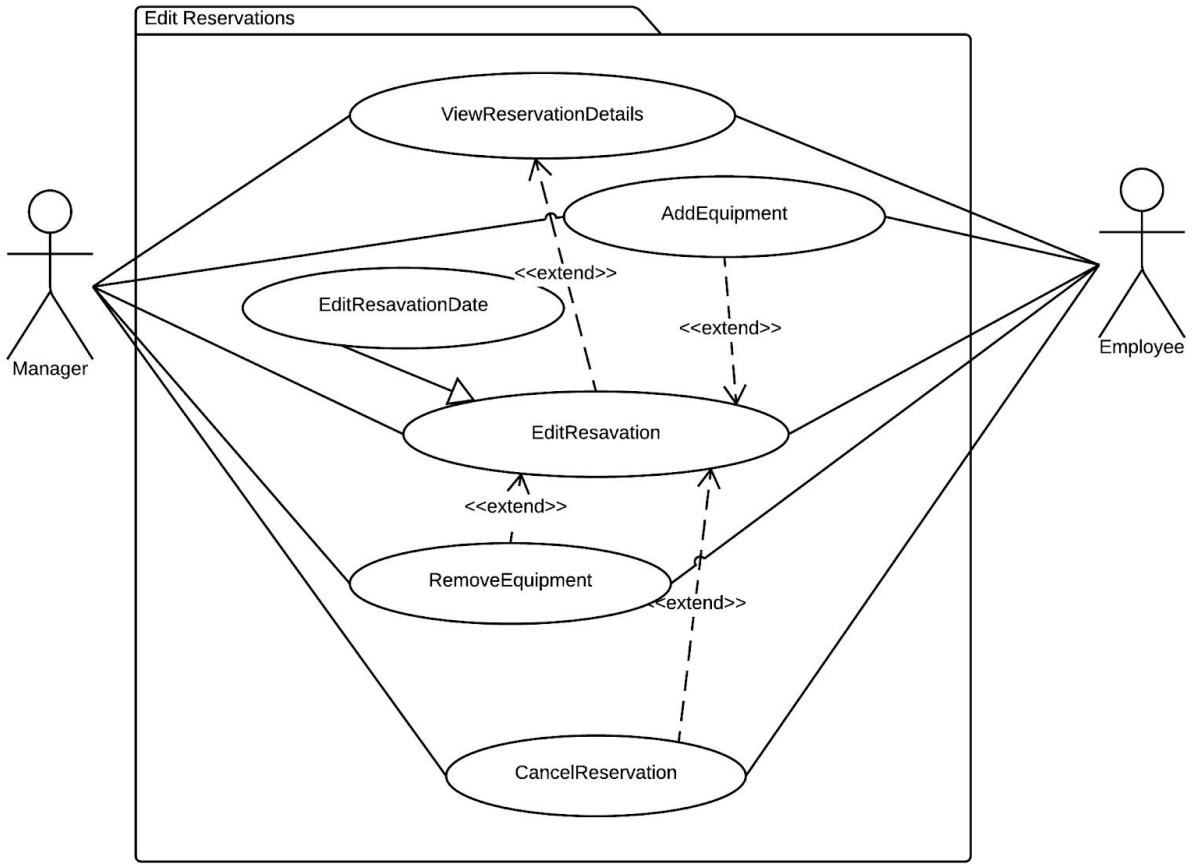

Rentals
ViewVehicles
<<extend>>
ReturnVehicle
ViewVehiclesDetails
<<extend>>
ReturnRental
PickUpRental
CreateIncidentReport
ViewIncidentreports
ViewIncidentreport
Employee
Manager


Customers
CreateCustomer
<<extend>>
NewMembership
ViewCustomerDetail
<<extend>>
EditCustomer
<<extend>>
CancelMmbership
<<extend>>
RemoveCustomer
Employee
Manage


Promotions
CreatePromotion
ViewPromotion
<<include>>
ViewPromotions
<<extend>>
EditPromotion
<<extend>>
RemovePromotion
Manager
Employee


Employees
CreateEmployeeAccount
ViewEmployee
ViewEmployeeDetails
<<extend>>
EditEmployee
<<extend>>
RemoveEmployee
<<Inculed>>
ConfirmUserDetails
System Administrator


System Management
BackupDatabase
SendReminder
EditSysSettings
EdiDatabaseSettings
System Administrator


Edit Reservations
ViewReservationDetails
AddEquipment
<<extend>>
EditResavationDate
<<extend>>
Manager
Employee
EditResavation
<<extend>>
RemoveEquipment
<<extend>>
CancelReservation

Use Case Descriptions:

Use case name	`Login`
Participating actors	Initiated by `Employee, Customer, or Manager`
Flow of events	1. `Employee` or `Customer` enters login details to login to the system 2. `Employee` or `Customer` presses the "Login" button 3. `TURBORENT` checks login details against database information corresponding to user 4. `TURBORENT` logs user into system and displays the current Reservations
Entry condition	• Participant is logged out of TURBORENT
Exit condition	• TURBORENT logs user into system • TURBORENT specifies reason as to why Participant could not login
Quality requirements	• `TURBORENT` logs the user into the system in less than five seconds

Use case name	`Logout`
Participating actors	Initiated by `Employee or ,Customer, or Manager`
Flow of events	1. `Employee or Customer` presses the "Logout" button displayed 2. `TURBORENT` logs user out of the system, logging any necessary details
Entry condition	• Participant is logged into `TURBORENT`
Exit condition	• Participant is logged out of `TURBORENT`
Quality requirements	• `TURBORENT` logs the user out of the system in less than three seconds

Use case name	`DisplayVehicles`
Participating actors	Initiated by `Employee`, `Manager,` or `Customer`
Flow of events	1. The `Employee, Manager, or Customer` would like to see all vehicles with a specified filter, or with no filter at all 2. The `Employee, Manager or Customer` then applies the filters to `TURBORENT` or none if none are specified 3. `TURBORENT` displays the vehicles that are put through the filter.
Entry condition	● The `Employee` is logged into `TURBORENT`
Exit condition	● `TURBORENT` displays all the vehicles in the specified filter, OR ● `TURBORENT` displays an explanation as to why the vehicles could not be displayed
Quality requirements	● `TURBORENT` applies the filters within 5 seconds
Applies to	● List vehicles, show overdue cars in categories and location, show vehicles that are for sale by price, category and location, list all vehicles sorted by branch

Use case name	`EditVehicle`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. Use case extends `ViewVehicleDetails` use case 2. `Manager` selects a `Vehicle` he/she would like to edit. 3. `Manager` can then change necessary information such as `rentalPricing` and `salePrice` 4. `Manager` presses the “Submit” button 5. `TURBORENT` validates and confirms changes
Entry condition	● `Manager` is in `ViewVehicleDetails` use case
Exit condition	● `TURBORENT` confirms that changes were made to a specified vehicle ● `TURBORENT` notifies the manager the reason as to why changes could not be processed.
Quality requirements	

Use case name	`AddLeasedVehicle`
Participating actors	Inherited from `AddVehicle`
Flow of events	1. `Manager` activates "Add Lease Vehicle" function on terminal from within `AddVehicle`. 2. `TURBORENT` produces form for additional leasing information required 3. `Manager` fills in all required information 4. `TURBORENT` validates and sends form information to Database 5. `TURBORENT` notifies `Manager` that vehicle addition successful 6. `TURBORENT` returns terminal to regular screen
Entry conditions	Inherited from `AddVehicle`
Exit conditions	Inherited from `AddVehicle`
Quality requirements	

Use case name	`AddVehicle`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` activates add new vehicle function on terminal 2. `TURBORENT` responds by displaying an add vehicle form 3. `Manager` fills in form 4. `TURBORENT` validates and sends form information to `Database` 5. `TURBORENT` indicates new vehicle form submission successful 6. `TURBORENT` returns terminal to regular screen
Entry conditions	• `Manager` is logged into `TURBORENT`
Exit conditions	• `Manager` receives notification that vehicle added to fleet • `TURBORENT` indicates why this addition was not successful
Quality requirements	

Use case name	`ViewVehicles`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` presses the "Vehicles" tab 2. `TURBORENT` displays all currently `Vehicles` that are in the "available" state 3. `Employee` or `Manager` can then filter for a certain type of `Vehicle` with the available filter functions 4. `TURBORENT` displays the `Vehicles` corresponding to the filter
Entry conditions	• `Employee` or `Manager` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` displays all `Vehicles` corresponding to the applied filter
Quality requirements	

Use case name	`ViewVehicleDetails`
Participating actors	Initiated by `Manager` or `Employee` Communicates with `Database`
Flow of events	1. `Employee` selects a `Vehicle` and presses the "View Vehicle" function 2. `TURBORENT` displays the information about the `Vehicle`
Entry conditions	• `Employee` or `Manager` is in the `ViewVehicles` use case
Exit conditions	• `TURBORENT` displays information about the `Vehicle`
Quality requirements	

Use case name	`ReturnVehicle`
Participating actors	Initiated by `Employee or Manager` Communicates with `Database`
Flow of events	1. Extends the `ViewVehicleDetails` use case 2. `Employee or Manager` presses the "Vehicle Return" button 3. `TURBORENT` changes the state of the Vehicle to "available" 4. `TURBORENT` confirms the change with a dialog
Entry conditions	• `Employee or Manager` is in the `ViewVehicle` use case
Exit conditions	• `TURBORENT` confirms that the `Vehicle` state has been changed • `TURBORENT` opens a dialog stating the reason why changes could not be made
Quality requirements	

Use case name	`RemoveVehicle`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. Use case extends `EditVehicle` use case 2. `Manager` activates "Remove Vehicle" function on terminal 3. `TURBORENT` displays remove vehicle form 4. `Manager` fills in form with correct information 5. `TURBORENT` validates the form and communicates with `Database` to ensure vehicle is in fleet 6. `TURBORENT` displays prompt for manager to confirm removal of vehicle from fleet 7. `Manager` selects yes or no 8. If yes `TURBORENT` communicates with `Database` to set vehicle as out of service 9. `TURBORENT` displays confirmation on terminal that vehicle removal was successful and returns terminal to regular secreen
Entry conditions	• `Manager` is logged into `TURBORENT` • `Vehicle` needs to be removed from fleet
Exit conditions	• `TURBORENT` confirms that the `Vehicle` was removed from the fleet
Quality requirements	

Use case name	`MakeReservation`
Participating actors	Initiated by `Employee` or `Manager`
Flow of events	1. `Employee` or `Manager` initiates the "New Car Rental" function on `TURBORENT` 2. The `Customer` provides the necessary details for car rental such as: `pickupLocation`, `returnLocation`, `pickupDate`, `returnDate`, `pickupAfterTime`, vehicle `make`, and, `model` 3. `Employee` or `Manager` enters the data provided by the `Customer` 4. `TURBORENT` calculates and displays the total cost of that `Reservation`. 5. `TURBORENT` processes payment (for deposit) 6. `Employee` or `Manager` then presses the "Submit" button to add the `Reservation` to the database 7. `TURBORENT` validates and confirms that the `Reservation` has been created and prints a pickup slip for the `Customer` 8. `Employee` or `Manager` gives pickup slip to the `Customer`
Entry condition	• The `Employee` is logged into `TURBORENT` • A `Customer` would like to make a new `Reservation` for a `Vehicle`
Exit condition	• A new `Reservation` record is shown in the database • A pickup slip is printed and given to the customer
Quality requirements	• `Reservation` record should not take more than 30 seconds to show up in database • Pickup slip should print within 60 seconds

Use case name	`ViewReservations`
Participating actors	Initiated by `Employee or Manager` Communicates with `Database`
Flow of events	1. `Employee or Manager` presses the "Reservations" tab 2. `TURBORENT` displays all current `Reservations` from the database
Entry condition	• `Employee or Manager` is logged into `TURBORENT`
Exit condition	• Current `Reservations` are displayed

Quality requirements	• `Employee or Manager` can specify a filter to view different `Reservation` categories

Use case name	`ViewReservationDetails`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` selects a `Reservation` and presses the "View Details" button 2. `TURBORENT` opens a dialog displaying the information of the selected `Reservation`
Entry condition	• `Employee` or `Manager` is viewing the `Reservations` table
Exit condition	• `TURBORENT` opens a dialog displaying `Reservation` information
Quality requirements	

Use case name	`EditReservation`
Participating actors	Initiated by `Employee` or `Manager`
Flow of events	1. This use case extends the `ViewReservationDetails` use case 2. `Employee` or `Manager` presses the "Edit Reservation" button 3. `Employee` or `Manager` edits Reservation details and presses the "Save Changes" button 4. `TURBORENT` opens a confirmation dialog to confirm changes 5. `Employee` or `Manager` confirms 6. `TURBORENT` makes the changes to the database
Entry condition	• `Employee or Manager` is logged into `TURBORENT` • A `Customer` would like to edit their `Reservation` for a `Vehicle`
Exit condition	• The Reservation record is updated with the new information • A pickup slip is printed and given to the customer
Quality requirements	• This option should ONLY be accessed by the employees or managers

Use case name	`EditReservationDate`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. This use case inherits the `EditReservation` use case 2. `Employee` or `Manager` changes the `pickupDate`/`returnDate`, vehicle type, or equipment attributes of the Reservation 3. `TURBORENT` detects the new changes to the `Reservation` that cause a change in rental price 4. `TURBORENT` validates changes and opens confirmation dialog 5. `Employee` or `Manager` confirms `Reservation` date changes 6. `TURBORENT` makes the changes to the reservation
Entry condition	● Inherited from `EditReservation`
Exit condition	● Inherited from `EditReservation`
Quality requirements	

Use case name	`CancelReservation`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. Extends the `EditReservation` use case. 2. The `Employee` or `Manager` presses the “Delete Reservation” button 3. `TURBORENT` opens confirmation dialog 4. `Employee` or `Manager` confirms 5. `TURBORENT` removes `Reservation` from the `Database` and opens confirmation dialog
Entry condition	● There exists a `Reservation` in the `Database` ● A `Customer` would like to cancel their `Reservation`
Exit condition	● `TURBORENT` confirms that the `Reservation` has been removed ● `TURBORENT` displays an error message
Quality requirements	

Use case name	`AddEquipment`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. This use case extends from the `editReservation` and `makeReservation` use cases 2. `Employee` or `Manager` presses the "Add Equipment" button which opens a dialog asking to specify type of `Equipment` and quantity. 3. `TURBORENT` validates and updates/adds the `Equipment` to the `Reservation` and sends it to the `Database`.
Entry condition	• The `Employee` is creating a new `Reservation` or editing an existing one and would like to add an `Equipment`
Exit condition	• Dialog confirming that `Equipment` was added • Dialog specifying the reason why `Equipment` could not be added
Quality requirements	

Use case name	`RemoveEquipment`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. This use case extends from `EditReservation` and `MakeReservation` use cases 2. `Employee` or `Manager` presses the "Remove Equipment" button which opens a dialog asking to specify type of `Equipment` and quantity to remove. 3. `TURBORENT` validates the changes and updates the `Reservation` and writes the changes to the `Database`.
Entry condition	• `Employee` or `Manager` is creating a new `Reservation` or editing a `Reservation` and would like to remove a piece of `Equipment` • There is at least one piece of `Equipment` listed with a given `Reservation`
Exit condition	• Dialog confirming that `Equipment` was removed. • Dialog specifying reason why `Equipment` could not be removed.
Quality requirements	

Use case name	`NewMembership`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. This use case extends from the `CreateCustomer` use case and `EditCustomer` use case 2. `Employee` or `Manager` specifies that this Customer now has Club Membership 3. `Employee` or `Manager` presses the "Submit" button 4. `TURBORENT` confirms that the customer has Club Membership now
Entry condition	• `Employee` or `Manager` is logged into `TURBORENT`
Exit condition	• `TURBORENT` confirms the `Customer` has membership now • `TURBORENT` opens a dialog specifying the reason the `Customer` could not become a member
Quality requirements	

Use case name	`CancelMembership`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. Use case extends `EditCustomer` use case 2. `Employee` or `Manager` presses the "Cancel Membership" button 3. `TURBORENT` confirms with the Participant that he/she would like to cancel this Club Membership 4. `TURBORENT` removes membership from the Customer 5. `TURBORENT` opens a dialog confirming that Club Membership was cancelled
Entry condition	• `Employee` or `Manager` is logged into TURBORENT • `Employee` or `Manager` is in the `EditCustomer` use case
Exit condition	• `Customer` no longer has Club Membership
Quality requirements	

Use case name	`EditCustomer`
Participating actors	Initiated by Employee or Manager Communicates with `Database`
Flow of events	1. Use case extends `ViewCustomerDetails` 2. `Employee` or `Manager` presses the "Edit Customer Details" button 3. `Employee` or `Manager` can then edit the `Customer` details 4. `Employee` or `Manager` then presses the "Save Changes" button which saves the changes 5. `TURBORENT` validates, sends changes to the Database, and confirms that changes were made
Entry condition	• `Employee` or `Manager` is in the `ViewCustomerDetails` use case
Exit condition	• `TURBORENT` opens a dialog confirming changes • `TURBORENT` opens a dialog specifying the reason why changes could not be made
Quality requirements	

Use case name	`ViewCustomerDetails`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` selects a customer and presses the "View Customer Details" button 2. `TURBORENT` opens a new window displaying `Customer` information
Entry condition	• `Employee` or `Manager` is logged into `TURBORENT`
Exit condition	• `Customer` details window is opened
Quality requirements	

Use case name	`CreateCustomer`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` is viewing the customer table and presses the "Create New Customer" button 2. `TURBORENT` opens a new dialog where the `Employee` or `Manager` can enter the details of the new `Customer` 3. `Employee` or `Manager` enters the necessary details such as `name`, `address`, `phone`, `email`, etc. 4. `Employee` or `Manager` presses the "Add" button 5. `TURBORENT` validates information and adds the new `Customer` to the Database 6. `TURBORENT` opens a dialog confirming that the `Customer` has been added
Entry condition	● `Employee` or `Manager` is logged into `TURBORENT`
Exit condition	● A new `Customer` is added to the database ● `TURBORENT` opens a dialog stating the reason why `Customer` could not be added
Quality requirements	

Use case name	`RemoveCustomer`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. Use case extends `EditCustomer` use case 2. `Employee` or `Manager` presses the "Delete Customer" button 3. `TURBORENT` opens a confirm dialog asking the `Employee` or `Manager` to confirm the deletion 4. `Employee` or `Manager` confirms the deletion. 5. `TURBORENT` removes the `Customer` from the `Database`
Entry condition	● `Employee` or `Manager` is in the `EditCustomer` use case
Exit condition	● `Customer` is deleted from the `Database` ● `TURBORENT` states the reason why `Customer` could not be deleted

Quality requirements	

Use case name	`CreateIncidentReport`
Participating actors	Initiated by `Manager` or `Employee` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` presses the "New Incident Report" button on `TURBORENT` 2. `Employee` or `Manager` fills out necessary information such as `reservationID`, `vehicleID`, `customerID`, `date`, `time`, `damageLevel`, `damageDescription`, `cost`. 3. `Employee` or `Manager` submits the new `IncidentReport` to the database 4. `TURBORENT` validates and confirms that the `IncidentReport` has been created
Entry condition	• `Employee or Manager` is logged into TURBORENT
Exit condition	• `TURBORENT` confirms that the `IncidentReport` has been created • `TURBORENT` opens a dialog specifying the reason why the `IncidentReport` could not be made • A form is created and signed by the customer • Damages will be covered by some form of insurance or charged to the customer's credit card
Quality requirements	

Use case name	`CreateOverdueReport`
Participating actors	Initiated by `Employee or Manager` Communicates with `Database`
Flow of events	1. Use case extends `ViewReservationDetails` 2. `Employee or Manager` presses the "Create OverDue Report" button 3. `TURBORENT` displays a preview of the overdue report 4. `Employee or Manager` presses the "Create" button 5. `TURBORENT` creates a new `OverdueReport` and displays it
Entry condition	• `Employee or Manager` is in `ViewReservationDetails` use case
Exit condition	• `TURBORENT` displays the new `OverdueReport`

Quality requirements	

Use case name	`CreatePromotion`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` activates "Add New Promotion" function on terminal 2. `TURBORENT` displays add new promotion form 3. `Manager` fills in form with correct information for specific classes of vehicles, rates of the promotion, requirements for promotion to be met, start and end dates of promotion 4. `TURBORENT` validates form and prompts to confirm promotion and details of promotion 5. `TURBORENT` communicates with `Database` to send form info and add promotion details to `Database` 6. `TURBORENT` displays conformation that promotion added successfully 7. `TURBORENT` returns terminal to regular `Manager` screen
Entry conditions	• `Manager` is logged into `TURBORENT` • `Promotion` end date must be in the future
Exit conditions	
Quality requirements	

Use case name	`EditPromotion`
Participating actors	Initiated by Manager Communicates with `Database`
Flow of events	1. Use case extends `ViewPromotion` 2. `Manager` presses the "Edit" button 3. `TURBORENT` displays a form of all the attributes that the `Manager` can edit 4. `Manager` edits some values such as: `promotionCondition`, `startDate`, `endDate`, etc. 5. `Manager` presses the "Apply Changes" button 6. `TURBORENT` confirms and applies the changes
Entry conditions	• `Manager` is in the `ViewPromotion` use case
Exit conditions	• `TURBORENT` confirms and applies the changes • `TURBORENT` states the reason as to why it could not apply changes
Quality requirements	

Use case name	`RemovePromotion`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. Use case extends `EditPromotion` 2. `Manager` activates remove promotion function on terminal 3. `TURBORENT` communicates with `Database` to get all promotions 4. `TURBORENT` displays remove promotion screen/form 5. `Manager` selects promotion to remove 6. `TURBORENT` prompts to confirm removal of promotion 7. `Manager` selects yes 8. `TURBORENT` communicates with `Database` to end it 9. `TURBORENT` displays confirmation that promotion removed successfully
Entry conditions	• `Manager` is logged into `TURBORENT` • `Promotion` to be removed must exist
Exit conditions	• `Promotion` is removed successfully and rental rates return to regular prices

Quality requirements	

Use case name	`ViewPromotion`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. Includes `ViewPromotions` use case 2. `Manager` selects promotion 3. `SUPPERENT` communicates with database and retrieves information for the selected promotion 4. `TURBORENT` displays information on terminal screen
Entry conditions	• `Manager` is logged into `TURBORENT` • `Manager` has initiated `ViewPromotions` use case and has list of promotions
Exit conditions	• Details of a specific promotion are displayed on screen for `Manager`
Quality requirements	

Use case name	`CreateEmployeeAccount`
Participating actors	Initiated by `SysAdmin` Communicates with `Database`
Flow of events	1. `SysAdmin` activates the add new employee function on the terminal 2. `TURBORENT` responds by displaying a create new employee form 3. `SysAdmin` fills out form with correct information and level of permission for employee (employee, manager, sysadmin) 4. `TURBORENT` validates form and checks `Database` for existing username 5. `TURBORENT` generates password 6. `TURBORENT` securely sends username and password to database 7. `TURBORENT` emails new `Employee` login info
Entry conditions	• `SysAdmin` is logged into `TURBORENT` • An `Employee` needs a new account on `TURBORENT`
Exit conditions	• New `Employee` has username and password for login with correct level of permission for them

Quality requirements	

Use case name	`ViewEmployees`
Participating actors	Initiated by `SysAdmin or Manager` Communicates with `Database`
Flow of events	1. `SysAdmin` activates the "View Employees" function on the terminal 2. `TURBORENT` responds with a list of employees with their employee IDs
Entry conditions	• `SysAdmin` is logged into TURBORENT
Exit conditions	• A list of employees are displayed
Quality requirements	

Use case name	`ChangeUserPassword`
Participating actors	Initiated by `Employee, Manager, or SysAdmin` (referred to as `Employee` for this use case) Communicates with `Database`
Flow of events	1. `Employee` activates change password function from terminal 2. `TURBORENT` responds by displaying screen for `Employee` to enter current password and new password 2 times 3. `Employee` enters current password and new password two times 4. `TURBORENT` authenticates `Employee with` current credentials 5. `TURBORENT` validates new password for strength requirements and both entries are identical 6. `TURBORENT` updates the `Employee` password in the database 7. `TURBORENT` confirms password update for `Employee`
Entry conditions	• `Employee` is logged into `TURBORENT`
Exit conditions	• `Employee` has new account password and remains logged into `TURBORENT` • `TURBORENT` displays reason for failure to change password

Quality requirements	

Use case name	`ViewEmployeeDetails`
Participating actors	Initiated by `SysAdmin` Communicates with `Database`
Flow of events	1. `SysAdmin` activates the view employee function on the terminal inputting an employee ID number 2. `TURBORENT` display employee information corresponding to the employee ID
Entry conditions	• `SysAdmin` is logged into TURBORENT
Exit conditions	• `TURBORENT` displays the `Employee` information
Quality requirements	

Use case name	`EditEmployee`
Participating actors	Initiated by `SysAdmin` Communicates with `Database`
Flow of events	1. Use case extends `ViewEmployeeDetails` 2. `SysAdmin` presses the “Modify” button 3. `SysAdmin` can change `Employee` data values. 4. `TURBORENT` confirms changes and submits them to the `Database` 5. `TURBORENT` logs any changes made 6. `TURBORENT` opens a dialog confirming that changes were made
Entry conditions	• `SysAdmin` is in the `ViewEmployees` use case
Exit conditions	• `TURBORENT` displays a dialog confirming that changes were made • `TURBORENT` opens a dialog specifying the reason why changes could not be made
Quality requirements	

Use case name	`RemoveEmployee`
Participating actors	Initiated by `SysAdmin` Communicates with `Database`
Flow of events	1. This use case extends the `EditEmployee` use case 2. `SysAdmin` selects the "Delete Employee" button 3. `TURBORENT` opens a dialog asking the `SysAdmin` to confirm deletion 4. `SysAdmin` confirms the deletion 5. `TURBORENT` marks the Employee as 'inactive'
Entry conditions	• `SysAdmin` is logged into TURBORENT
Exit conditions	• `Employee` is marked as 'inactive'
Quality requirements	

Use case name	`EditSysSettings`
Participating actors	Initiated by `SysAdmin`
Flow of events	1. `SysAdmin` presses the "System Settings" button 2. `TURBORENT` opens a window displaying system settings 3. `SysAdmin` can then change the values of certain settings like `SysTime`, `SysDate`, etc. 4. `SysAdmin` presses the "Apply Changes" button 5. `TURBORENT` confirms any changes, applies them and logs them
Entry conditions	• `SysAdmin` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` confirms any setting changes • `TURBORENT` opens a dialog stating why settings could not be changed
Quality requirements	

Use case name	`EditDatabaseSettings`
Participating actors	Initiated by `SysAdmin` Communicates with `Database`
Flow of events	1. `SysAdmin` presses the "Database Settings" button 2. `TURBORENT` opens a window displaying system settings 3. `SysAdmin` can then change the values of certain `Database` settings like the backup schedule 4. `SysAdmin` presses the "Apply" button 5. `TURBORENT` confirms any changes and applies them, then closes them
Entry conditions	• `SysAdmin` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` confirms any `Database` changes • `TURBORENT` displays the reason why changes could not be made
Quality requirements	• `TURBORENT` confirms any changes in less than five seconds

Use case name	`BackupDatabase`
Participating actors	Initiated by `SysAdmin` Communicates with `Database`
Flow of events	1. `SysAdmin` activates the "Backup Database" command from the terminal 2. `TURBORENT` creates a new backup of the `Database`, which is written to the "backups" folder 3. `TURBORENT` opens a new prompt confirming that a backup has been created
Entry conditions	• `SysAdmin` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` displays a dialog confirming that the database was backed up • `TURBORENT` displays a dialog stating the reason why the backup could not be created
Quality requirements	

Use case name	`SendReminder`
Participating actors	Initiated by `TURBORENT` Implemented by `SysAdmin`
Flow of events	1. `SysAdmin` activates "Send Reminder" function 2. `TURBORENT` responds by displaying the "Send Reminder" screen 3. `SysAdmin` sets the details of the automatic reminders such as timing before rental date begins and timing before rental is due back
Entry conditions	• `SysAdmin` is logged into `TURBORENT`
Exit conditions	• `SysAdmin` is notified that reminder settings set successfully • `SysAdmin` is notified why changing settings failed
Quality requirements	

Use case name	`ViewOverdueReports`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` presses the "Overdue Reports" tab on `TURBORENT` 2. `TURBORENT` displays the `OverdueReports` as a list
Entry conditions	• `Manager` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` displays the `OverdueReports`
Quality requirements	• `TURBORENT` displays the `OverdueReports` in less than three seconds

Use case name	`ViewOverdueReport`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` selects a specific `OverdueReport` and presses the "View" button 2. `TURBORENT` displays all relevant information regarding the `OverdueReport` in a new window.
Entry conditions	• `Manager` is in the `ViewOverdueReports` use case
Exit conditions	• `TURBORENT` displays all relevant information regarding an `OverdueReport`
Quality requirements	• `TURBORENT` displays the `OverdueReport` in less than three seconds

Use case name	`ManagerOveride`
Participating actors	Initiated by `Manager`, `Employee` Communicates with `Database`
Flow of events	1. `TURBORENT` informs `Employee` that action requires `Manager` override 2. `Employee` selects "Manager Override" 3. `TURBORENT` displays override login prompt for `Manager` 4. `Manager` inputs credentials and selects "Override" 5. `TURBORENT` authenticates `Manager` credentials 6. `Employee` completes action required 7. `TURBORENT` logs out `Manager` and returns terminal to regular employee screen
Entry conditions	• `Employee` is logged into TURBORENT • `Employee` needs to perform a one-time action that requires elevated credentials • `Manager` has valid credentials to login to TURBORENT
Exit conditions	• `Employee` remains logged in and `Manager` is not logged in • Use case/action requiring credential elevation is completed

Quality requirements	

Use case name	`PickupRental`
Participating actors	Initiated by `Employee or Manager` Communicates with `Database`
Flow of events	1. `Employee` views a `Reservation` that is being taken out 2. `Employee` presses the "Picked Up" button 3. `TURBORENT` marks the `Vehicle` in the `Reservation` as "out" and the `Reservation` as "in progress" 4. `TURBORENT` confirms to the `Employee` that the `Reservation` is now in progress
Entry conditions	• `Employee` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` confirms that the `Reservation` is now in progress • `TURBORENT` opens a dialog stating why `Reservation` could not be put into progress
Quality requirements	

Use case name	`ReturnRental`
Participating actors	Initiated by `Employee or Manager`
Flow of events	1. Extends `ViewReservation` use case 2. `Employee or Manager` presses the "Rental Complete" button 3. `TURBORENT` changes the state of the `Reservation` to "complete" and marks the state of the `Vehicle` to "maintenance" 4. `TURBORENT` confirms that the `Reservation` is complete
Entry conditions	• `Employee or Manager`is viewing a Reservation that is "in progress"
Exit conditions	• `TURBORENT` confirms that the `Reservation` is now complete • `TURBORENT` opens a dialog stating why `Reservation` could not be completed

Quality requirements	

Use case name	`ViewIncidentReports`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` selects the "Incident Reports" tab on `TURBORENT` 2. `TURBORENT` displays all `IncidentReports` as a list
Entry conditions	• `Employee` or `Manager` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` displays a list of `IncidentReports`
Quality requirements	• `TURBORENT` displays the `IncidentReports` in less than three seconds

Use case name	`ViewIncidentReport`
Participating actors	Initiated by `Employee` or `Manager` Communicates with `Database`
Flow of events	1. `Employee` or `Manager` selects an `IncidentReport` and presses the "View" button 2. `TURBORENT` displays a window with information regarding the IncidentReport
Entry conditions	• `Employee` or `Manager` is in the `ViewIncidentReports` use case
Exit conditions	• `TURBORENT` displays a window with information regarding the `IncidentReport`
Quality requirements	• `TURBORENT` displays the `IncidentReport` in less than three seconds

Use case name	`ViewSalesReports`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` selects the "Sales Reports" tab on `TURBORENT` 2. `TURBORENT` displays all the `SalesReports` 3. `Manager` can specify a filter such as "Daily", "Weekly", or "Monthly" to view the corresponding `SalesReports`
Entry conditions	• `Manager` is logged into `TURBORENT`
Exit conditions	• `TURBORENT` displays the `SalesReports` that correspond to a filter or all of them if no filter is applied
Quality requirements	

Use case name	`ViewSalesReportDetails`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` selects a `SalesReport` and presses the "View" button 2. `TURBORENT` displays the information about the `SalesReport` in a new window
Entry conditions	• Manager is logged into TURBORENT
Exit conditions	• `TURBORENT` displays the information corresponding to a specific `SalesReport` • `TURBORENT` displays an error message

Quality requirements	

Use case name	`CreateSalesReport`
Participating actors	Initiated by `Manager` Communicates with `Database`
Flow of events	1. `Manager` presses the "Create Sales Report" function on `TURBORENT` 2. `TURBORENT` displays a form for the `Manager` to fill out 3. `Manager` fills in necessary information such as the period of the `SalesReport` 4. `TURBORENT` validates and creates the `SalesReport` 5. `TURBORENT` displays the `SalesReport` in a new window
Entry conditions	● `Manager` is logged into `TURBORENT`
Exit conditions	● `TURBORENT` displays the new `SalesReport` ● `TURBORENT` displays an error message
Quality requirements	

Use case name	`CreateBalanceReport`
Participating actors	Initiated by `Manager or Employee` Communicates with `Database`
Flow of events	1. `Manager or Employee` selects Create Balance Report function on their terminal 2. `TURBORENT` communicates with Database to get information about the fleet and all equipment 3. TURBORENT displays the information as a formatted report on the screen showing the totals of each type of vehicle and equipment and the the counts of those rented out, our for repair or at the branch
Entry conditions	● `Manager or Employee` is logged into `TURBORENT`
Exit conditions	● `TURBORENT` displays the new `BalanceReport`

	• OR TURBORENT displays an error message explianing why the report couldn't be generated
Quality requirements	

Use case name	CustomerMakeReservation
Participating actors	Initiated by Customer Communicates with Database
Flow of events	1. Customer presses the "Make New Rental" function on TURBORENT 2. TURBORENT displays a form to make a new Reservation 3. The Customer provides the necessary details for car Reservation such as: pickupLocation, returnLocation, pickupDate, returnDate, pickupAfterTime, vehicle make, and, model 4. TURBORENT calculates and displays the total cost of that Reservation. 5. TURBORENT processes payment for the deposit 6. Customer then presses the "Submit" button 7. TURBORENT confirms that the Reservation has been created and sends a confirmation email to the customer with the confirmation number and secret key for Reservation edits
Entry conditions	• Customer is on the TURBORENT application
Exit conditions	• TURBORENT sends a confirmation email to the Customer • TURBORENT explains why the Reservation could not be made
Quality requirements	

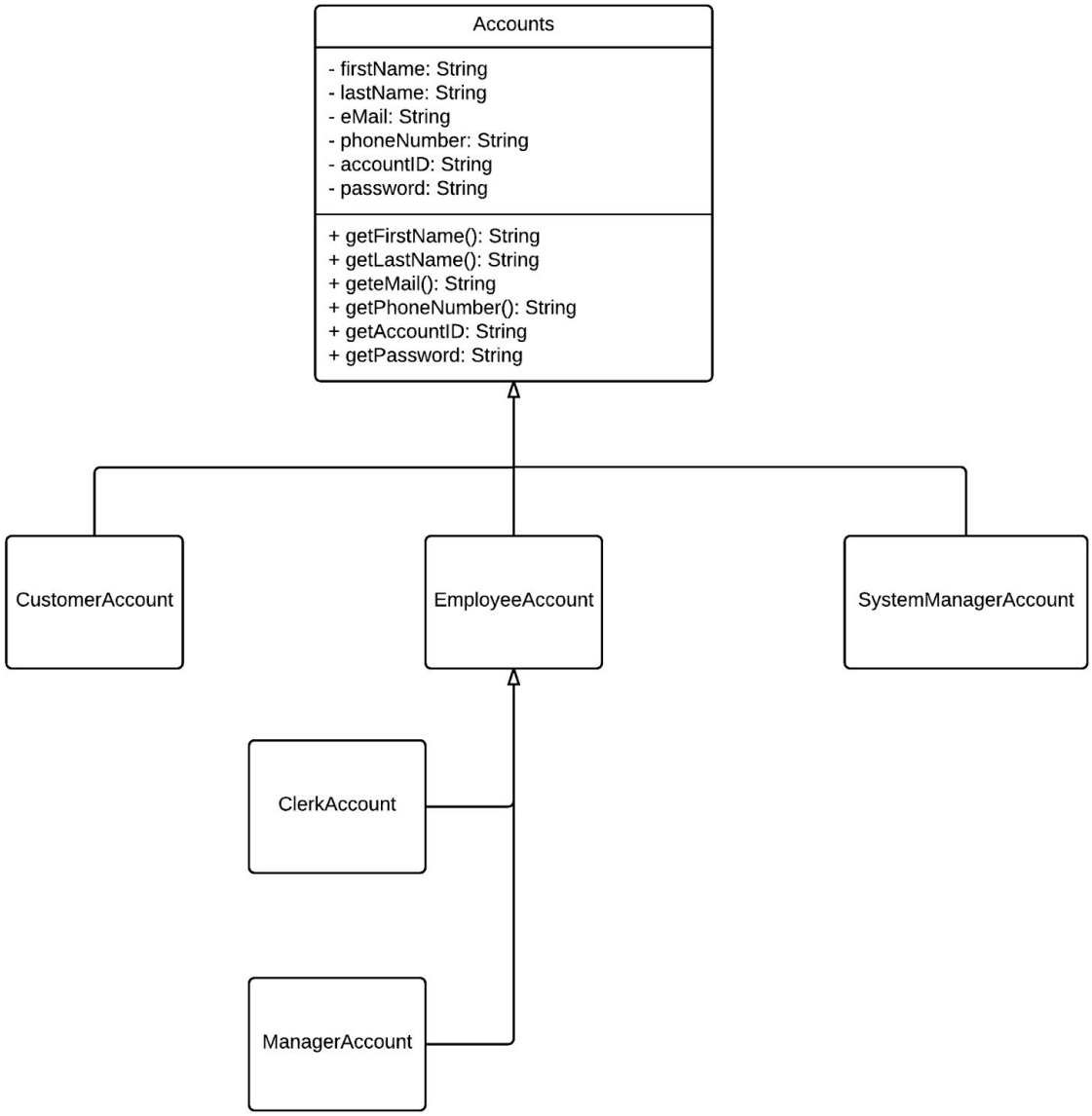
Use case name	CustomerViewReservation
Participating actors	Initiated by Customer Communicates with Database
Flow of events	1. Customer presses the "Check Reservation" function on the TURBORENT application and enters his/her confirmation number

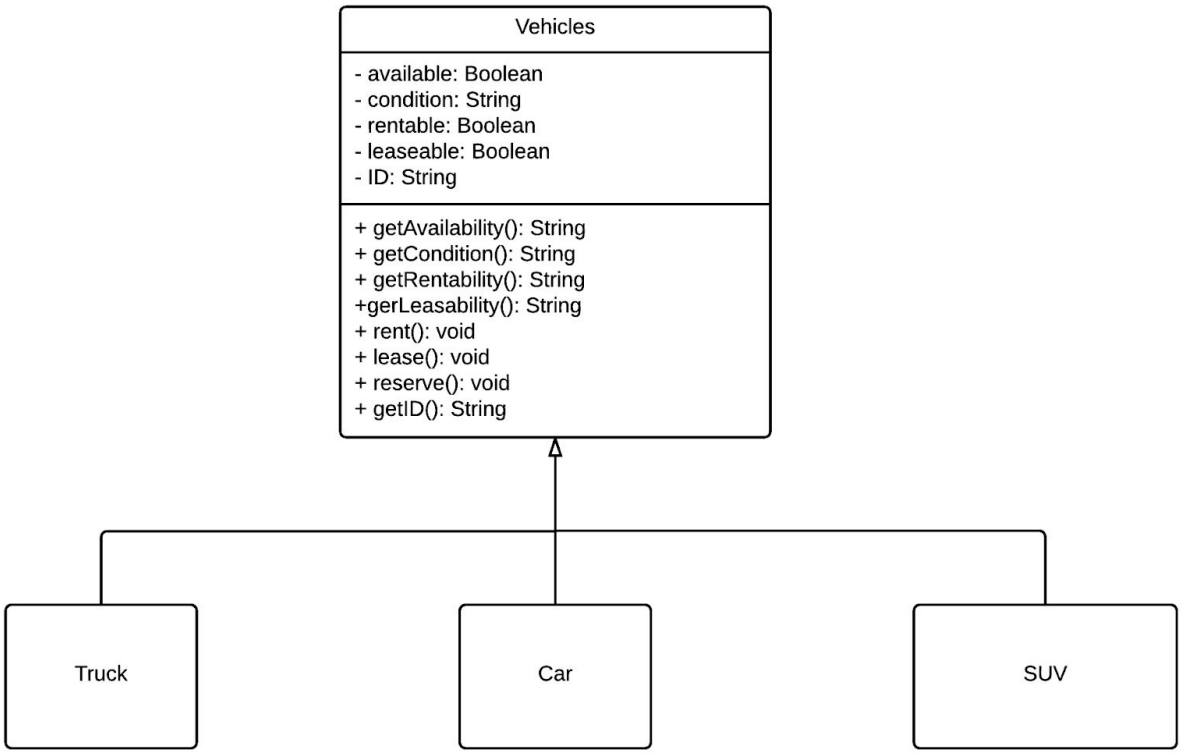
	2. `TURBORENT` fetches the relevant Reservation information and displays it in a form.
Entry conditions	• `Customer` has previously made a `Reservation` and is on the `TURBORENT` application
Exit conditions	• `TURBORENT` displays the relevant `Reservation` information • `TURBORENT` displays an error message
Quality requirements	

Use case name	`CustomerEditReservation`
Participating actors	Initiated by `Customer` Communicates with `Database`
Flow of events	1. Use case extends `CustomerViewReservation` 2. `Customer` presses the "Edit Reservation" button from the form 3. `TURBORENT` asks for the secret key sent in the confirmation email 4. `Customer` enters the secret key into the input box and presses "Next" 5. `TURBORENT` displays an edit form where the `Customer` can edit the `Reservation` (to values that are valid). 6. `Customer` changes edits the `Reservation` and presses "Next" 7. `TURBORENT` displays the new cost for the `Reservation` 8. `Customer` presses the "Apply Changes" button 9. `TURBORENT` changes the values of the `Reservation` and opens a confirmation dialog
Entry conditions	• `Customer` is in the `CustomerViewReservation` use case
Exit conditions	• `TURBORENT` confirms the changes to the `Reservation` • `TURBORENT` displays an error message
Quality requirements	

iii. Object model

The following are objects in which will be interacting with the system.

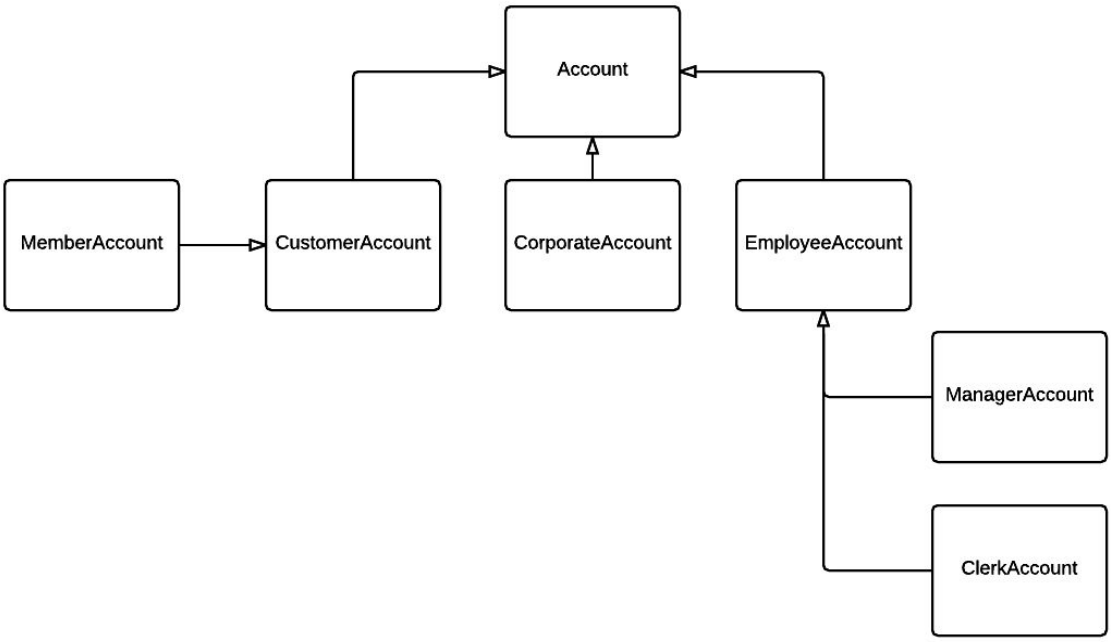



Vehicles
- available: Boolean
- condition: String
- rentable: Boolean
- leaseable: Boolean
- ID: String
+ getAvailability(): String
+ getCondition(): String
+ getRentability(): String
+gerLeasability(): String
+ rent(): void
+ lease(): void
+ reserve(): void
+ getID(): String
Truck
Car
SUV

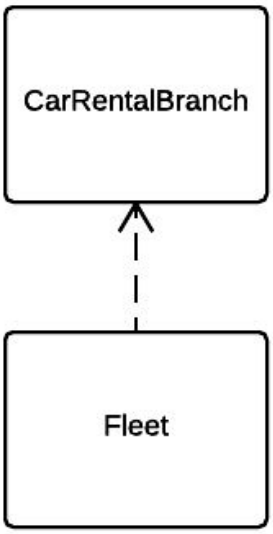
iv. Class diagrams

These diagrams are tentative models for how the structure of the program will be laid out.

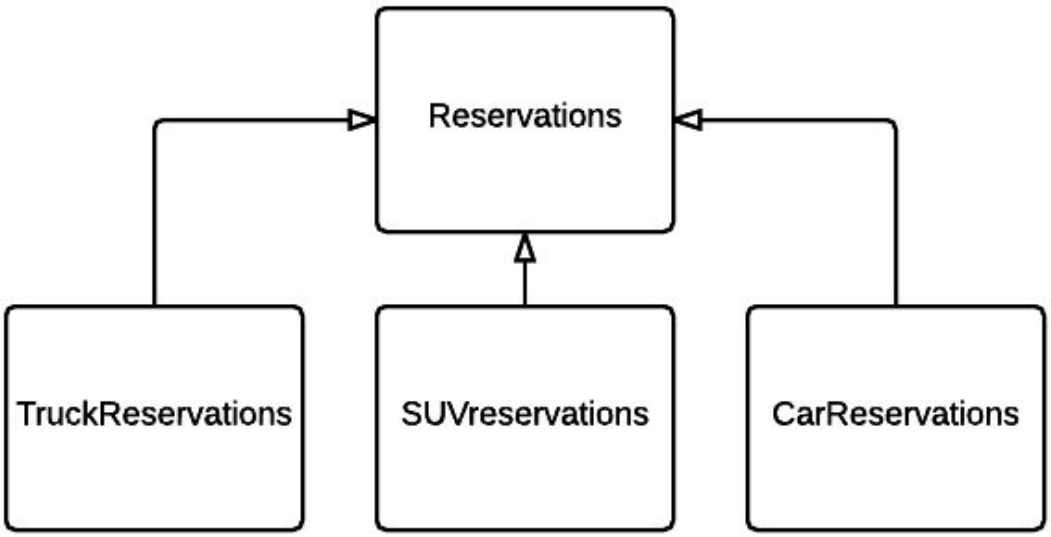
application.turborent.accounts



application.turborent.rentalbranches

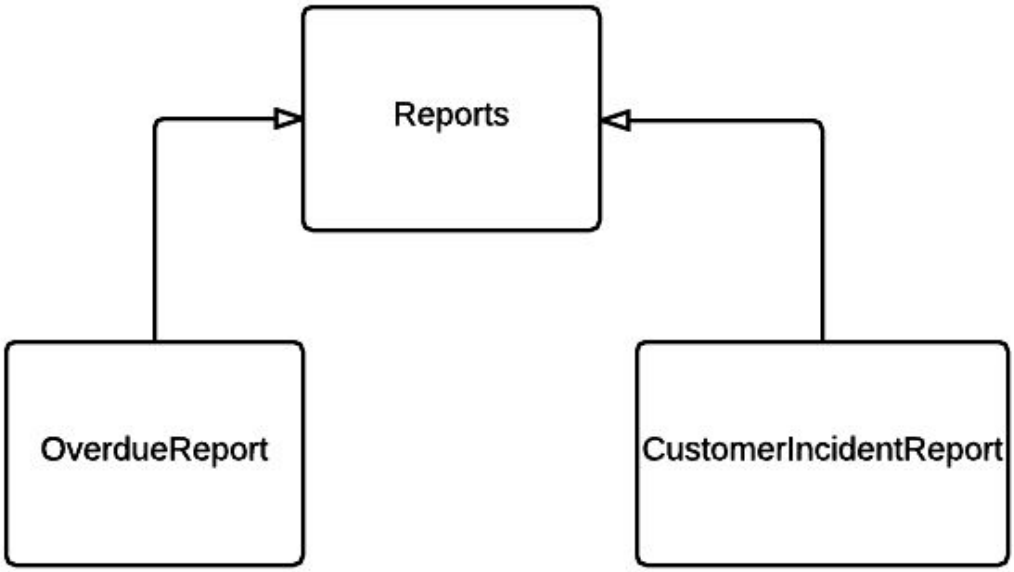


application.turborent.reservations

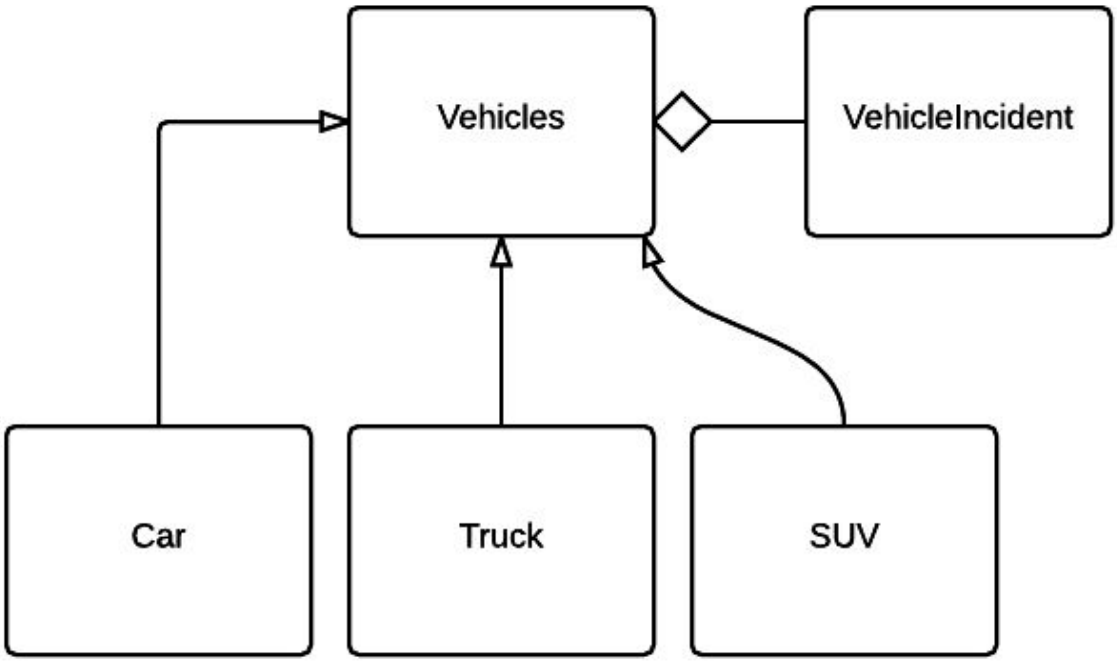


application.turborent.interfaces

application.turborent.reports



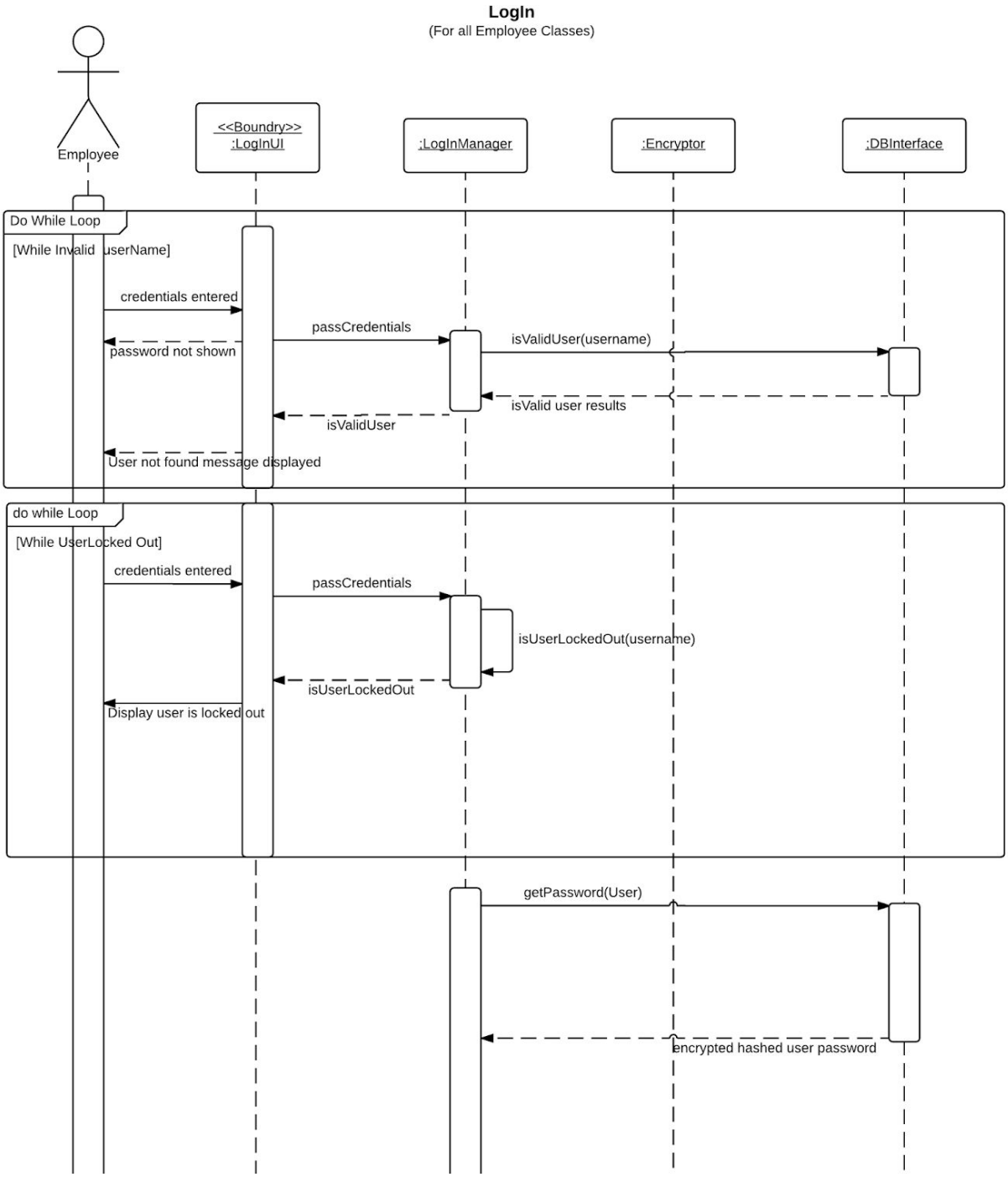
application.turborent.vehicles

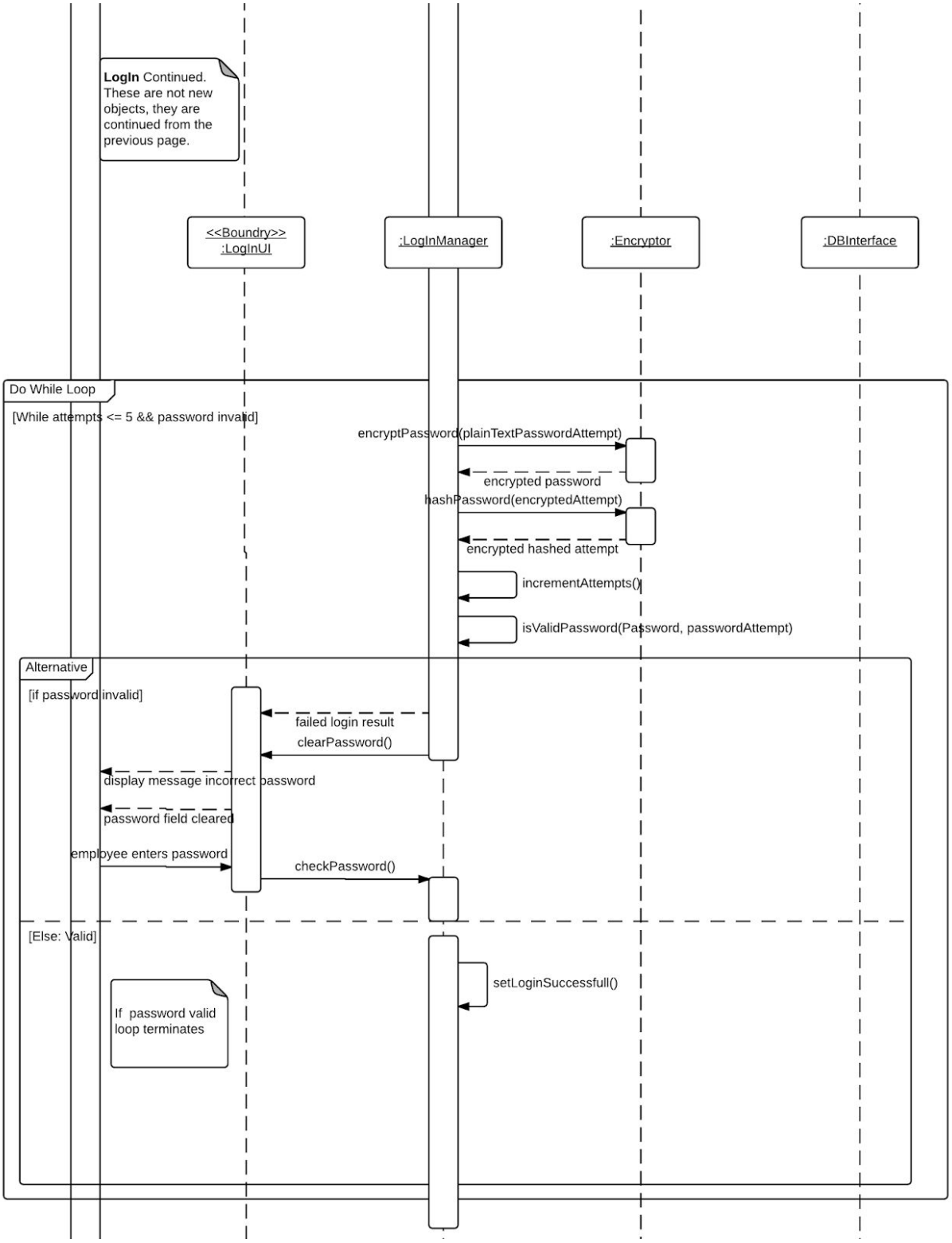


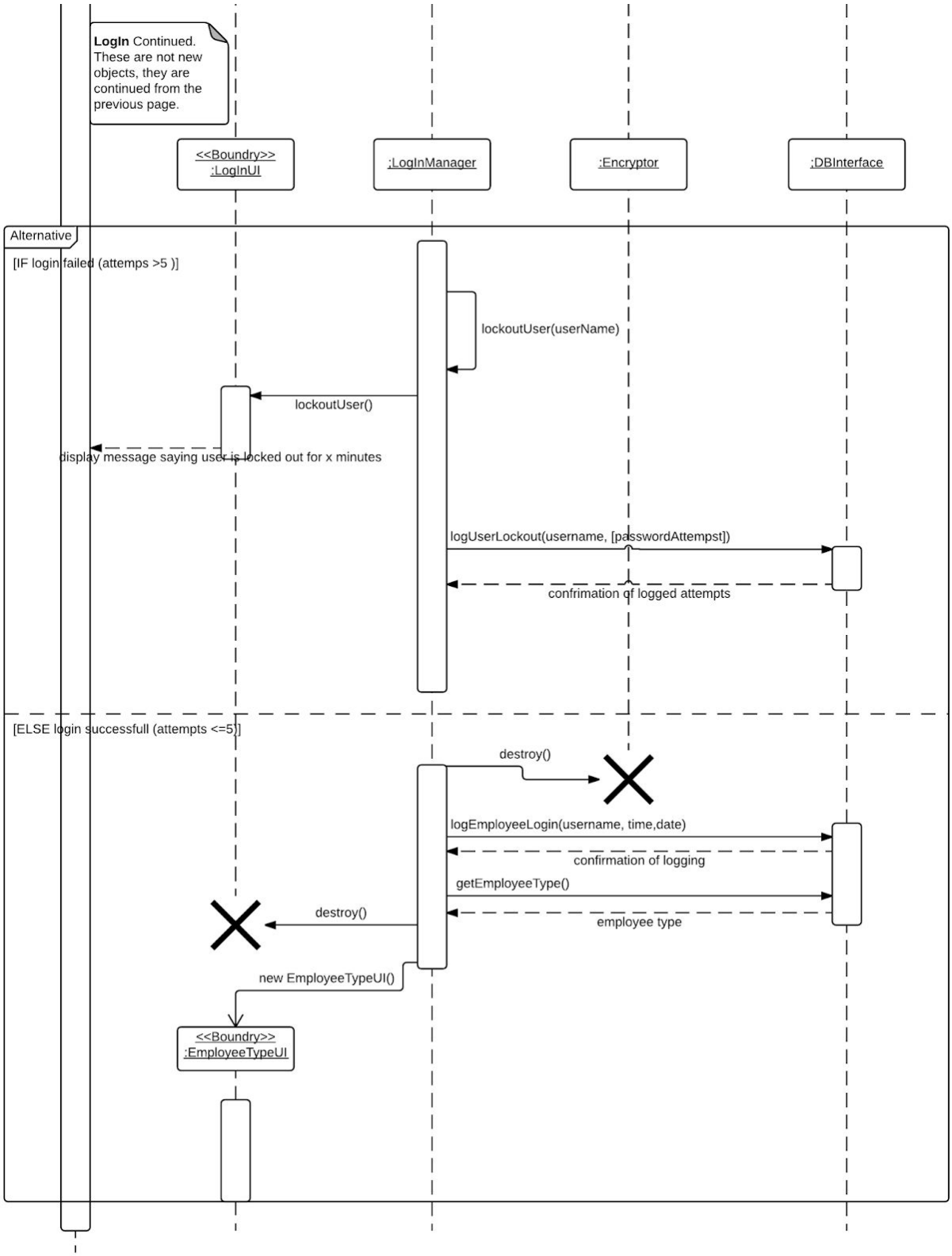
v. Dynamic models

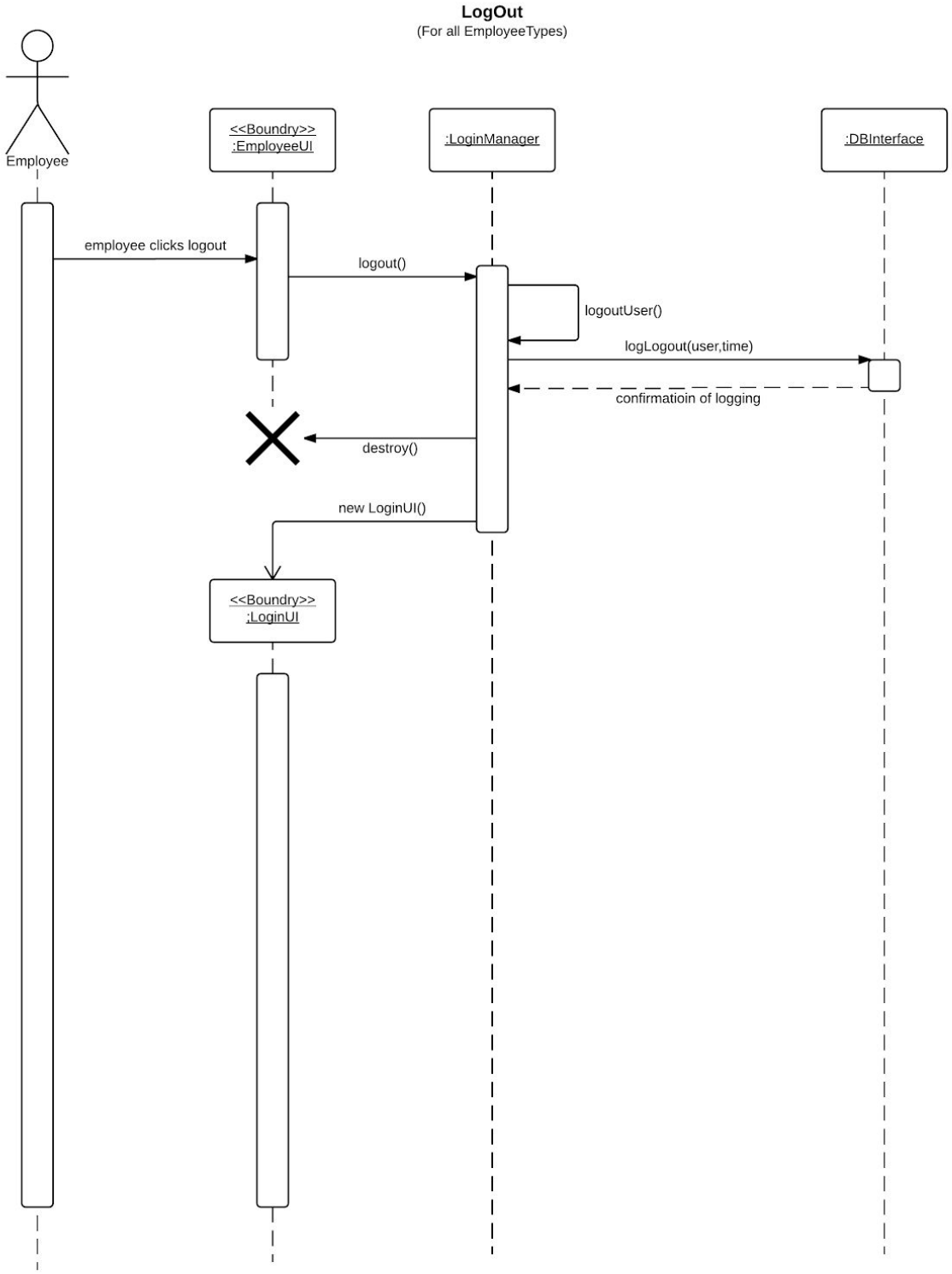
Sequence Diagrams

The following diagrams show what happens to some of the most common program interactions.

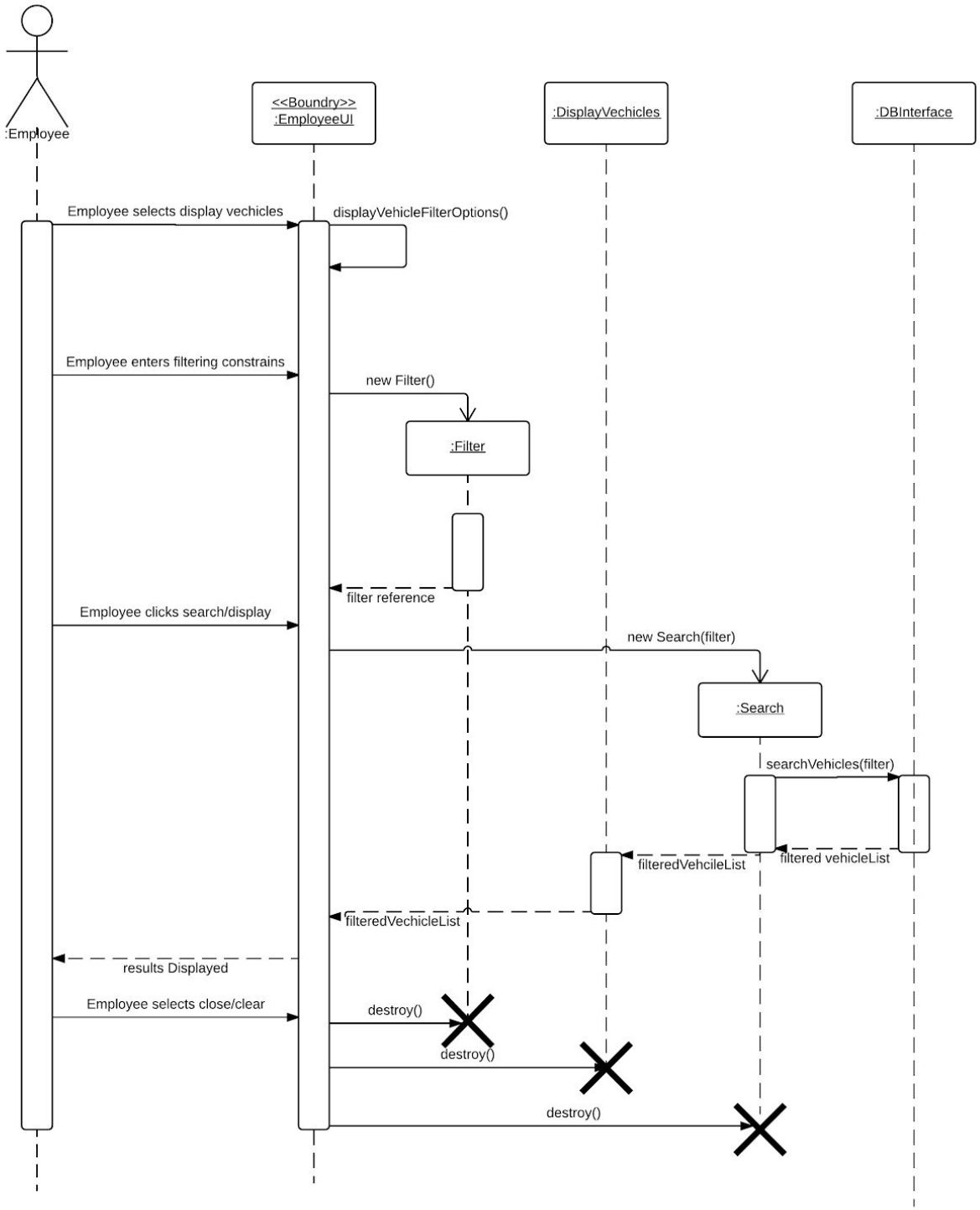


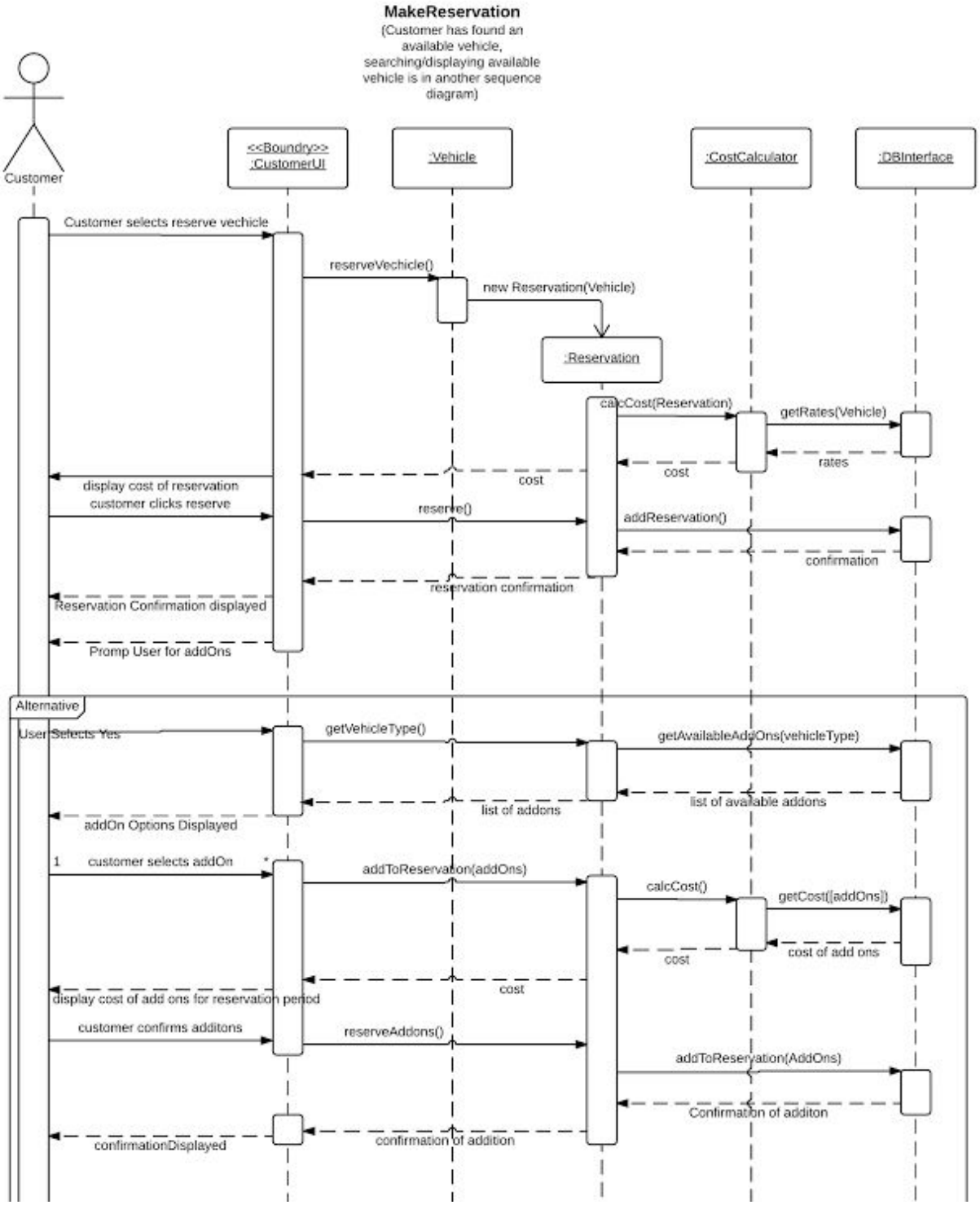

LogIn Continued. These are not new objects, they are continued from the previous page.
<<Boundry>> :LogInUI
:LogInManager
:Encryptor
:DBInterface
Do While Loop
[While attempts <= 5 && password invalid]
encryptPassword(plainTextPasswordAttempt)
encrypted password
hashPassword(encryptedAttempt)
encrypted hashed attempt
incrementAttempts()
isValidPassword(Password, passwordAttempt)
Alternative
[if password invalid]
failed login result
clearPassword()
display message incorrect password
password field cleared
employee enters password
checkPassword()
[Else: Valid]
setLoginSuccessfull()
If password valid loop terminates

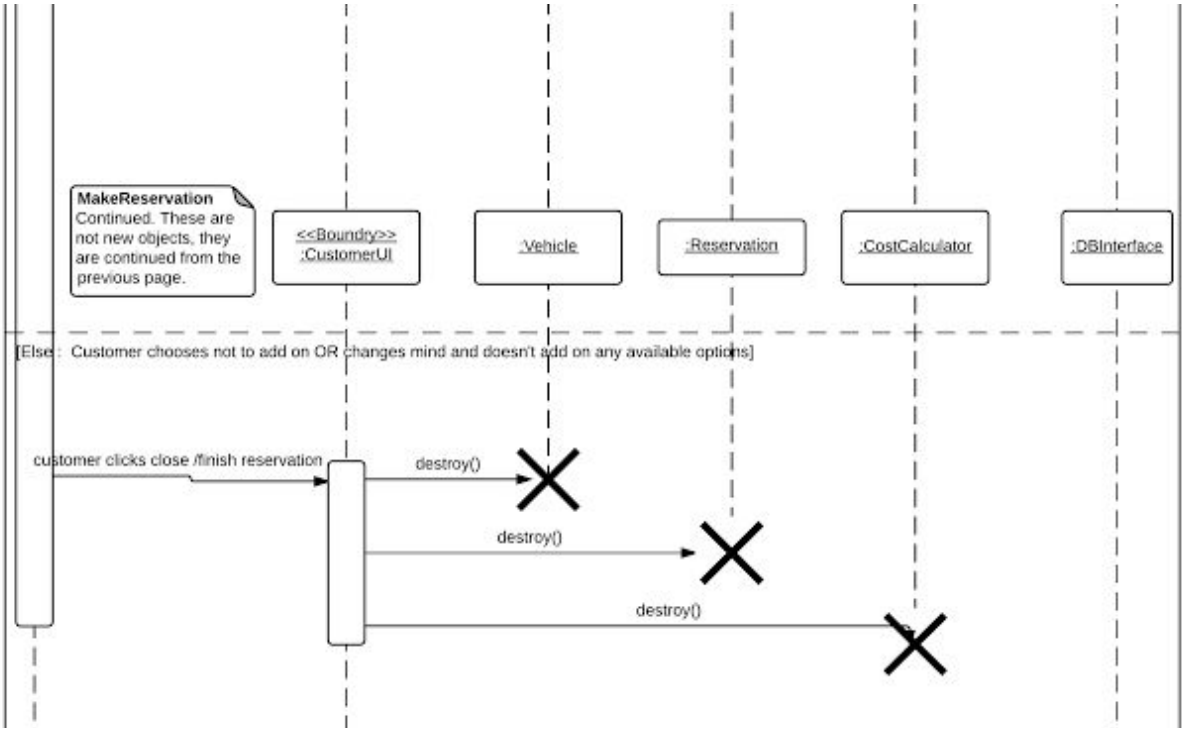

LogIn Continued. These are not new objects, they are continued from the previous page.
<<Boundry>> :LogInUI
:LogInManager
:Encryptor
:DBInterface
Alternative
[IF login failed (attemps >5)]
lockoutUser(userName)
lockoutUser()
display message saying user is locked out for x minutes
logUserLockout(username, [passwordAttempst])
confrimation of logged attempts
[ELSE login successfull (attempts <=5)]
destroy()
logEmployeeLogin(username, time,date)
confirmation of logging
getEmployeeType()
destroy()
employee type
new EmployeeTypeUI()
<<Boundry>> :EmployeeTypeUI

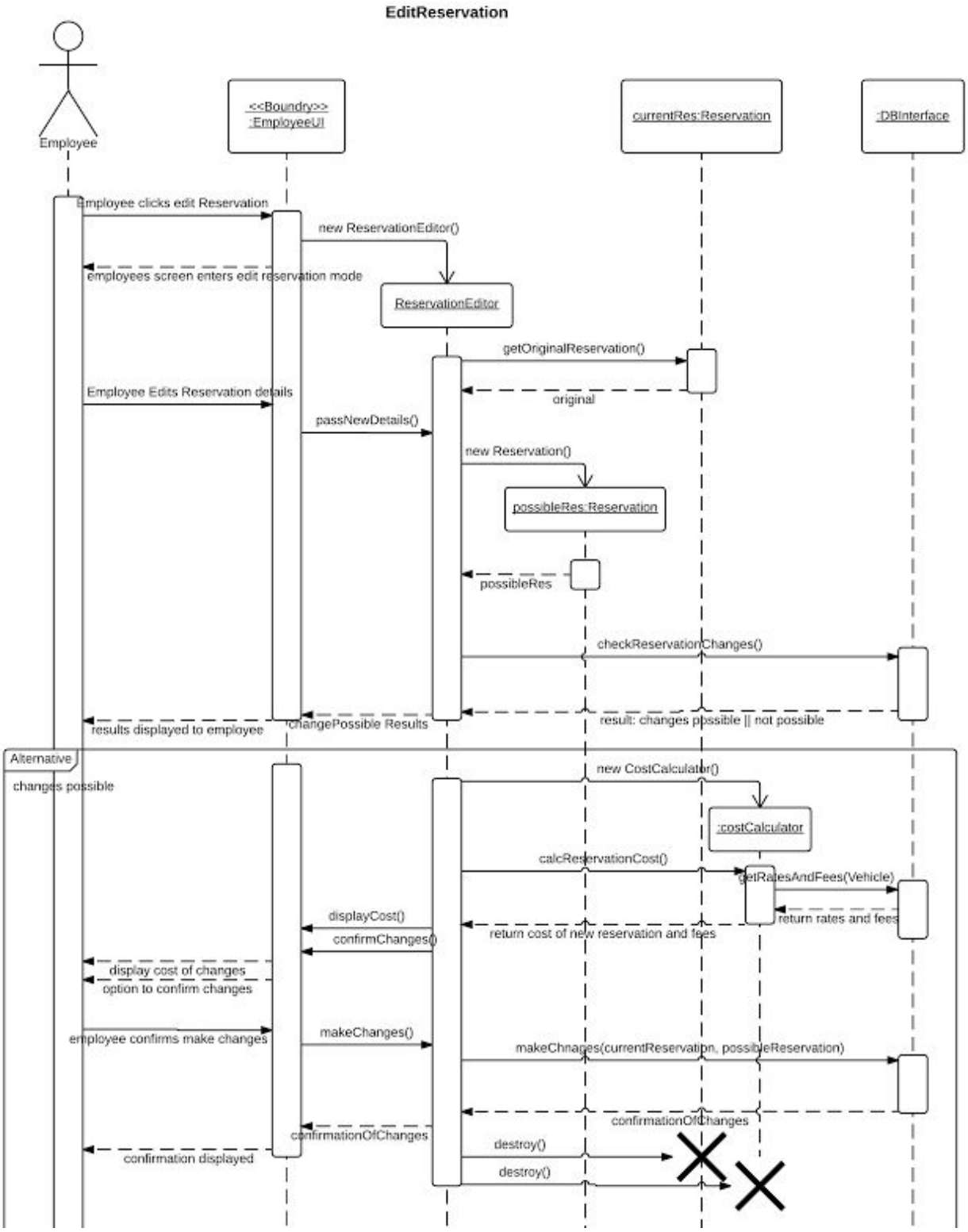

LogOut
(For all EmployeeTypes)
Employee
<<Boundry>>
:EmployeeUI
:LoginManager
:DBInterface
employee clicks logout
logout()
logoutUser()
logLogout(user,time)
confirmatioin of logging
destroy()
new LoginUI()
<<Boundry>>
:LoginUI

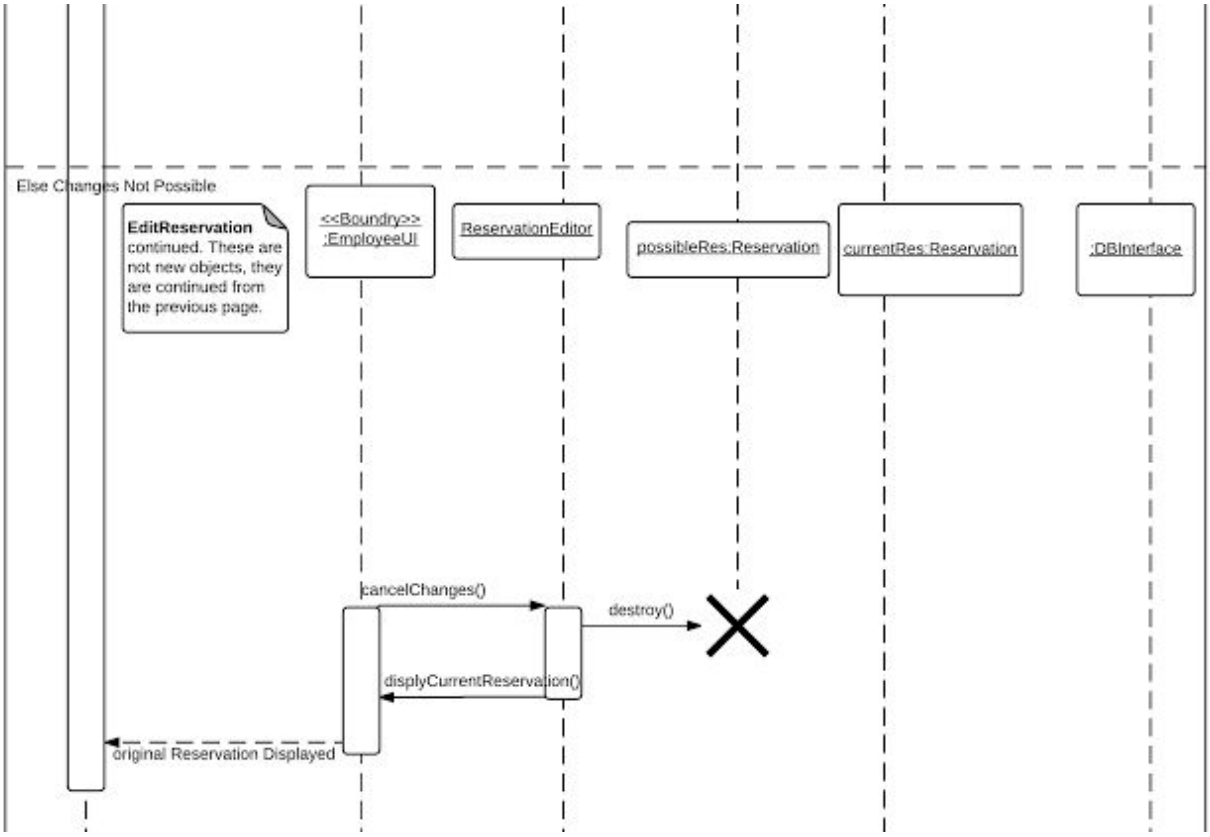
Displaying Vehicles in Rental Fleet

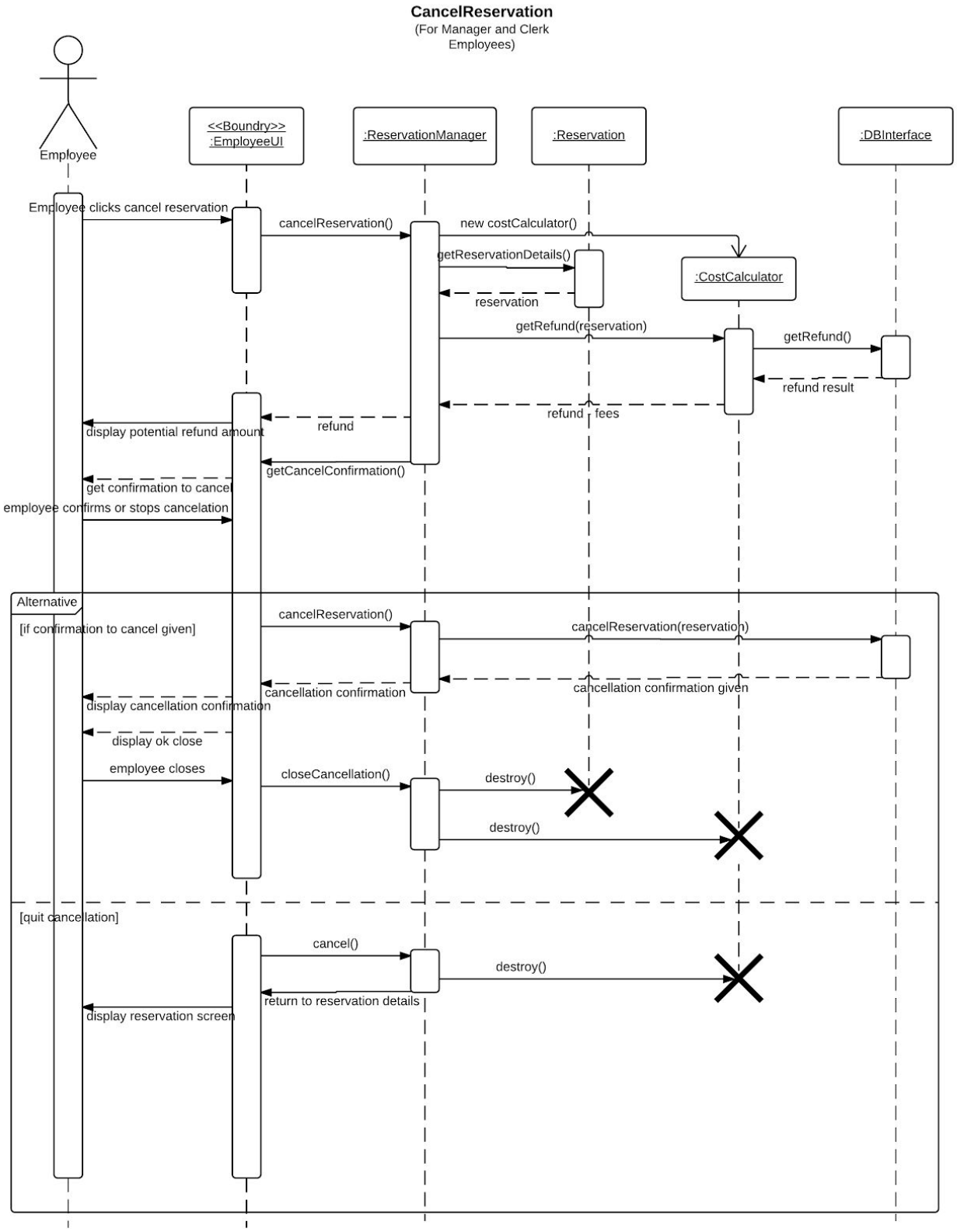


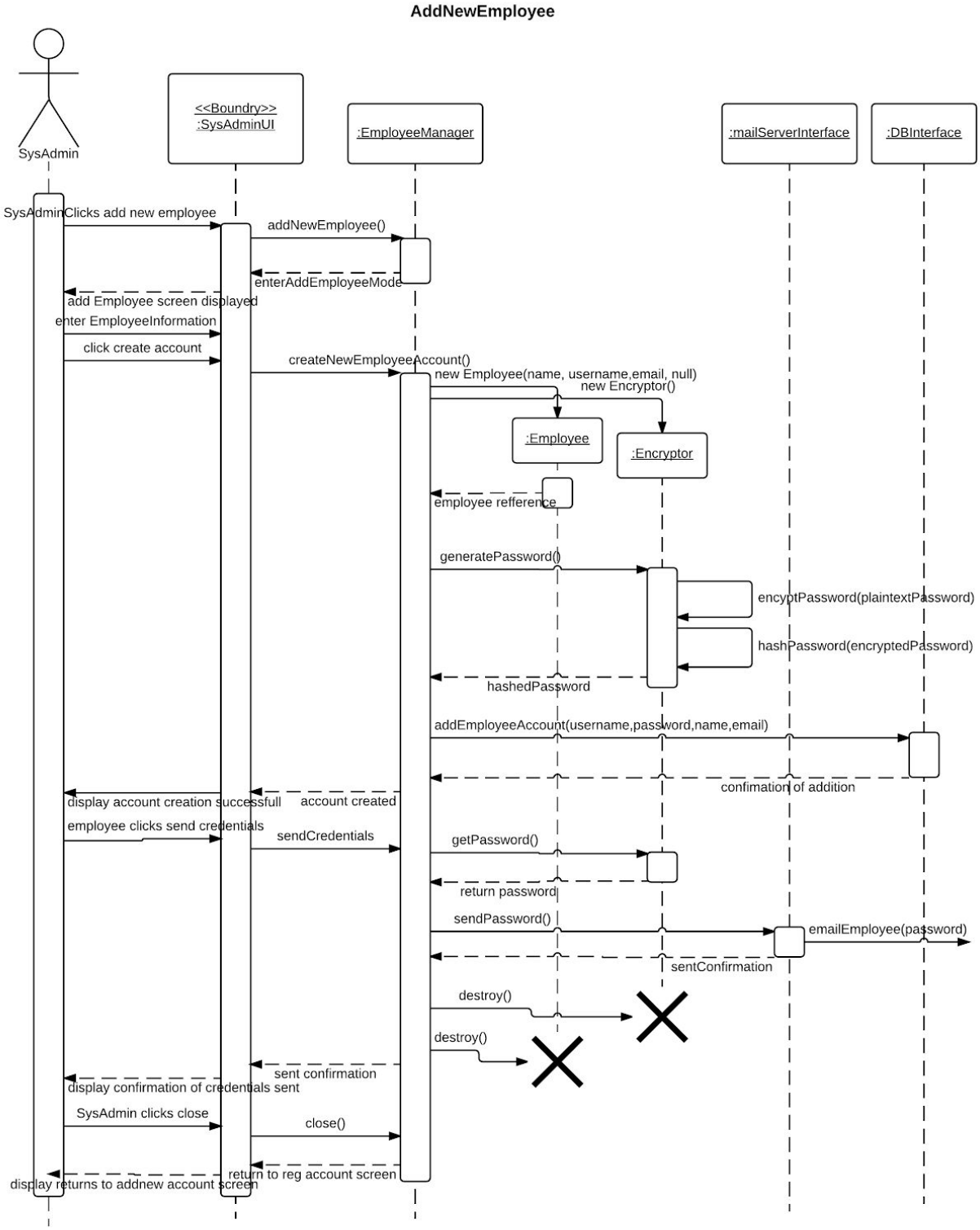

MakeReservation
(Customer has found an available vehicle, searching/displaying available vehicle is in another sequence diagram)
Customer
<<Boundry>>
:CustomerUI
:Vehicle
:CostCalculator
:DBInterface
Customer selects reserve vechicle
reserveVechicle()
new Reservation(Vehicle)
:Reservation
calcCost(Reservation)
getRates(Vehicle)
rates
cost
cost
display cost of reservation
customer clicks reserve
reserve()
addReservation()
confirmation
reservation confirmation
Reservation Confirmation displayed
Promp User for addOns
Alternative
User Selects Yes
getVehicleType()
getAvailableAddOns(vehicleType)
list of addons
list of available addons
addOn Options Displayed
1
customer selects addOn
*
addToReservation(addOns)
calcCost()
getCost([addOns])
cost
cost of add ons
cost
display cost of add ons for reservation period
customer confirms additons
reserveAddons()
addToReservation(AddOns)
Confirmation of additon
confirmationDisplayed
confirmation of addition


MakeReservation
Continued. These are not new objects, they are continued from the previous page.
<<Boundry>>
:CustomerUI
:Vehicle
:Reservation
:CostCalculator
:DBInterface
[Else : Customer chooses not to add on OR changes mind and doesn't add on any available options]
customer clicks close /finish reservation
destroy()
destroy()
destroy()


EditReservation
Employee
<<Boundry>>
:EmployeeUI
currentRes:Reservation
:DBInterface
Employee clicks edit Reservation
new ReservationEditor()
employees screen enters edit reservation mode
ReservationEditor
getOriginalReservation()
original
Employee Edits Reservation details
passNewDetails()
new Reservation()
possibleRes:Reservation
possibleRes
checkReservationChanges()
results displayed to employee
changePossible Results
result: changes possible || not possible
Alternative
changes possible
new CostCalculator()
:costCalculator
calcReservationCost()
getRatesAndFees(Vehicle)
return rates and fees
displayCost()
confirmChanges()
return cost of new reservation and fees
display cost of changes
option to confirm changes
employee confirms make changes
makeChanges()
makeChnages(currentReservation, possibleReservation)
confirmationOfChanges
confirmationOfChanges
confirmation displayed
destroy()
destroy()

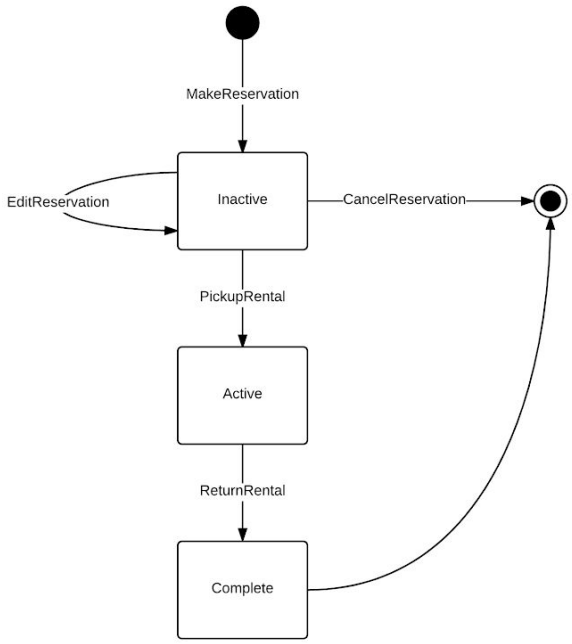

Else Changes Not Possible
EditReservation
continued. These are not new objects, they are continued from the previous page.
<<Boundry>>
:EmployeeUI
ReservationEditor
possibleRes:Reservation
currentRes:Reservation
:DBInterface
cancelChanges()
destroy()
displyCurrentReservation()
original Reservation Displayed

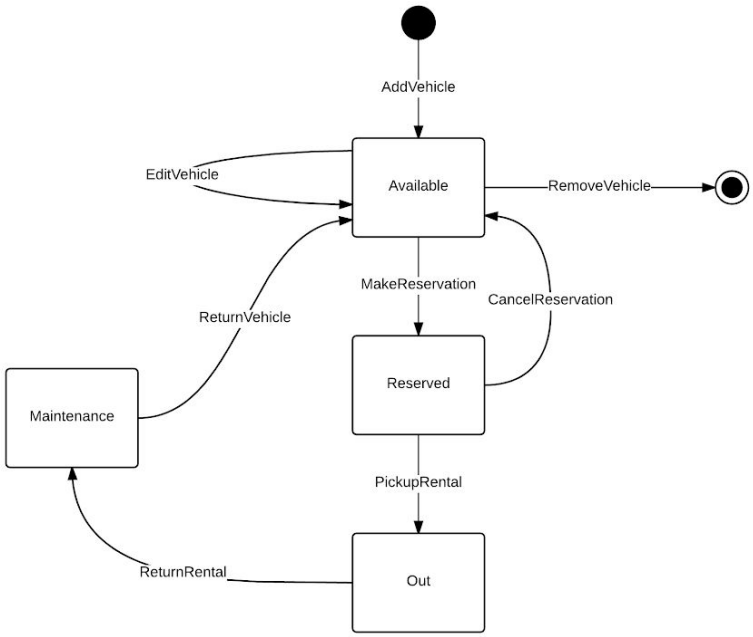

CancelReservation
(For Manager and Clerk Employees)
Employee
<<Boundry>>
:EmployeeUI
:ReservationManager
:Reservation
:DBInterface
:CostCalculator
Employee clicks cancel reservation
cancelReservation()
new costCalculator()
getReservationDetails()
reservation
getRefund(reservation)
getRefund()
refund result
refund - fees
refund
display potential refund amount
getCancelConfirmation()
get confirmation to cancel
employee confirms or stops cancelation
Alternative
[if confirmation to cancel given]
cancelReservation()
cancelReservation(reservation)
cancellation confirmation given
cancellation confirmation
display cancellation confirmation
display ok close
employee closes
closeCancellation()
destroy()
destroy()
[quit cancellation]
cancel()
destroy()
return to reservation details
display reservation screen

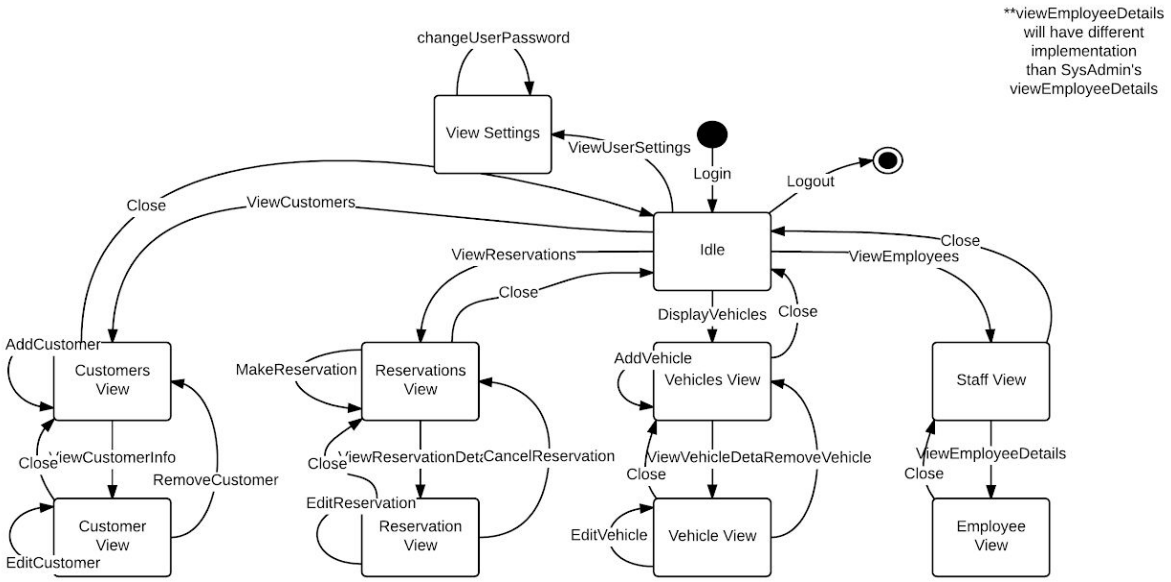

AddNewEmployee
SysAdmin
<<Boundry>>
:SysAdminUI
:EmployeeManager
:mailServerInterface
:DBInterface
SysAdminClicks add new employee
addNewEmployee()
enterAddEmployeeMode
add Employee screen displayed
enter EmployeeInformation
click create account
createNewEmployeeAccount()
new Employee(name, username,email, null)
new Encryptor()
:Employee
:Encryptor
employee refference
generatePassword()
encyptPassword(plaintextPassword)
hashPassword(encryptedPassword)
hashedPassword
addEmployeeAccount(username,password,name,email)
confimation of addition
display account creation successfull
account created
employee clicks send credentials
sendCredentials
getPassword()
return password
sendPassword()
emailEmployee(password)
sentConfirmation
destroy()
destroy()
sent confirmation
display confirmation of credentials sent
SysAdmin clicks close
close()
return to reg account screen
display returns to addnew account screen

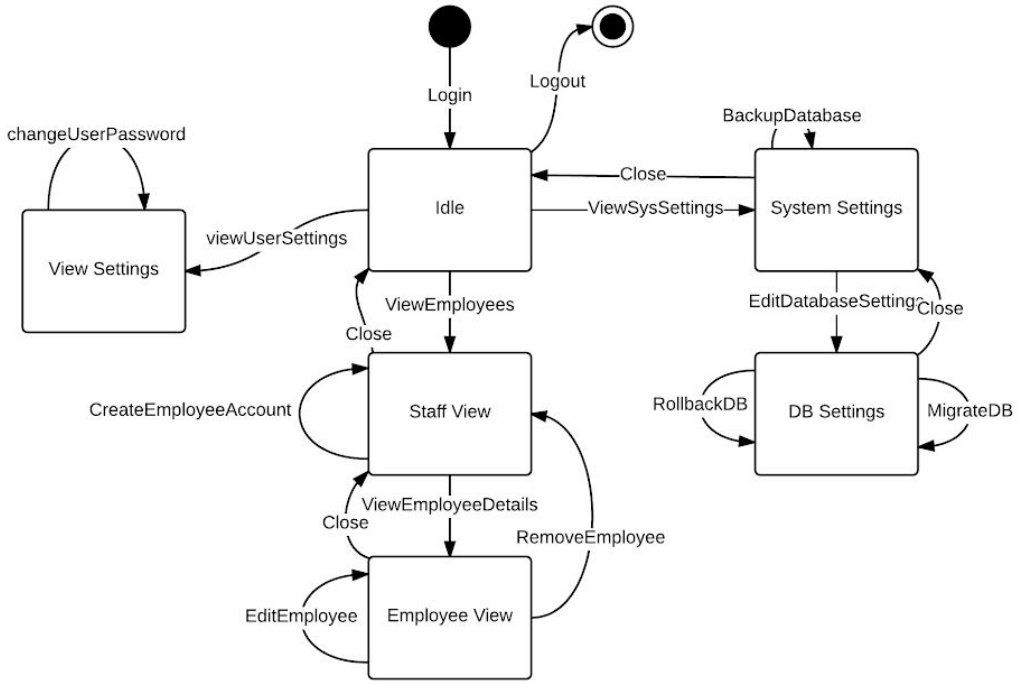
State Diagrams

These diagrams show the possible states that the system will or can be in at anytime. They show the changes in states when certain actions are taken.



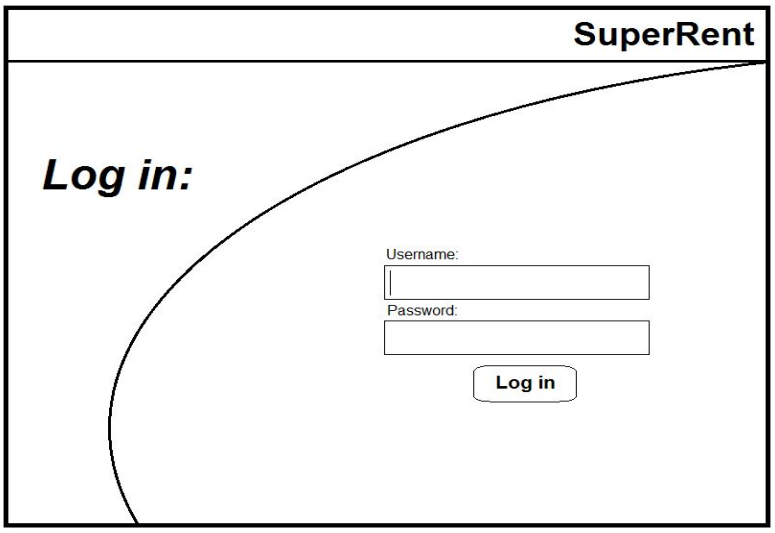


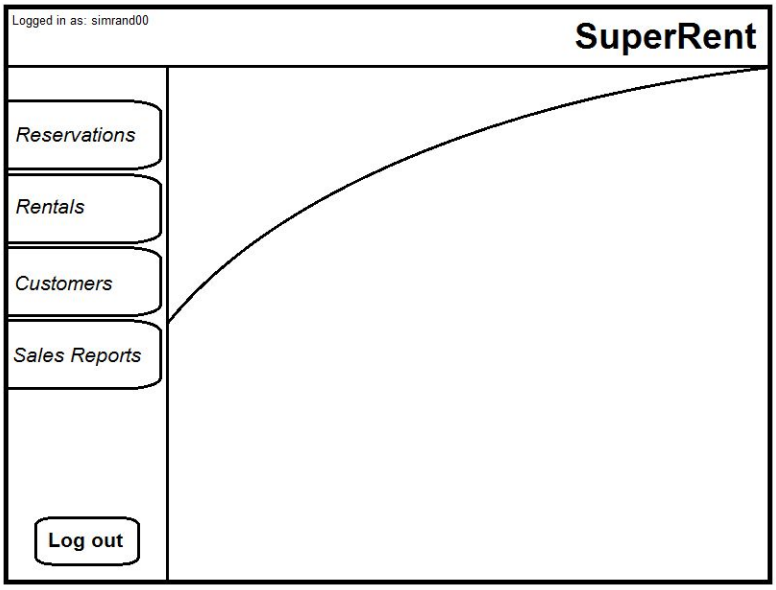

**viewEmployeeDetails will have different implementation than SysAdmin's viewEmployeeDetails
changeUserPassword
View Settings
ViewUserSettings
Login
Logout
Close
ViewCustomers
Idle
ViewReservations
Close
ViewEmployees
Close
DisplayVehicles
Close
AddCustomer
Customers View
MakeReservation
Reservations View
AddVehicle
Vehicles View
Staff View
Close
ViewCustomerInfo
RemoveCustomer
Close
ViewReservationDet
CancelReservation
ViewVehicleDeta
RemoveVehicle
Close
ViewEmployeeDetails
Close
EditReservation
Customer View
Reservation View
EditVehicle
Vehicle View
Employee View
EditCustomer

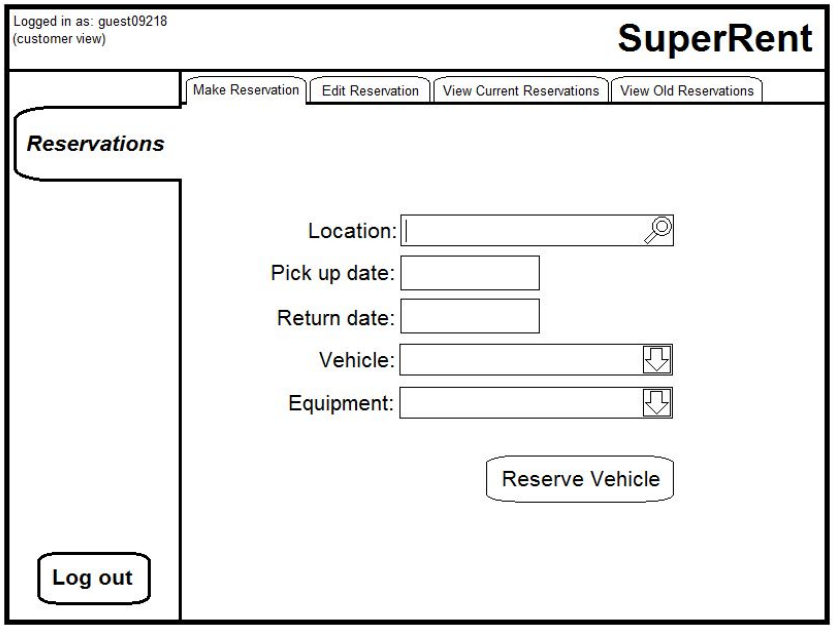

Login
Logout
BackupDatabase
changeUserPassword
Close
Idle
ViewSysSettings
System Settings
viewUserSettings
View Settings
ViewEmployees
EditDatabaseSettings
Close
Close
CreateEmployeeAccount
Staff View
RollbackDB
DB Settings
MigrateDB
ViewEmployeeDetails
Close
RemoveEmployee
EditEmployee
Employee View

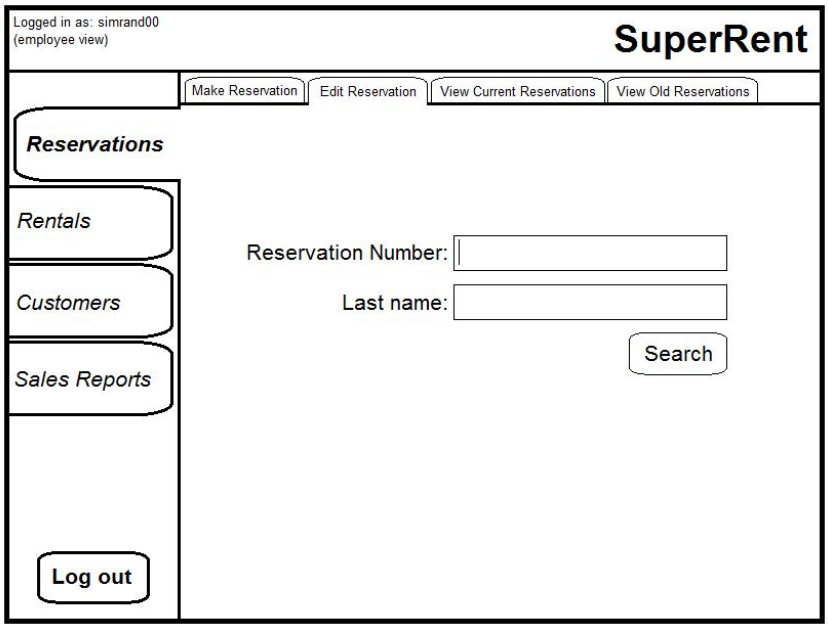
vi. User interface

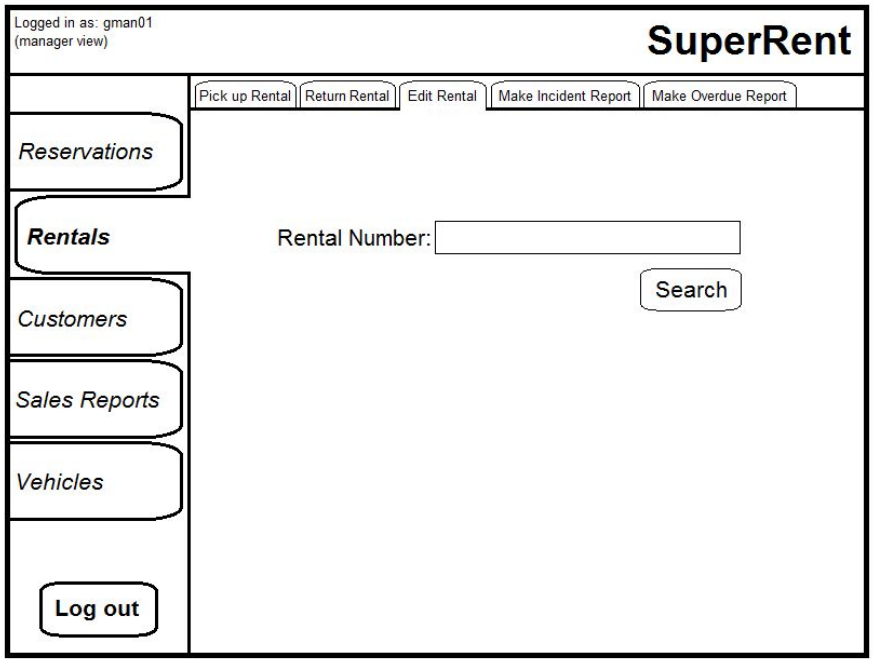
The general GUI format for the rental software will look like the following pictures below.

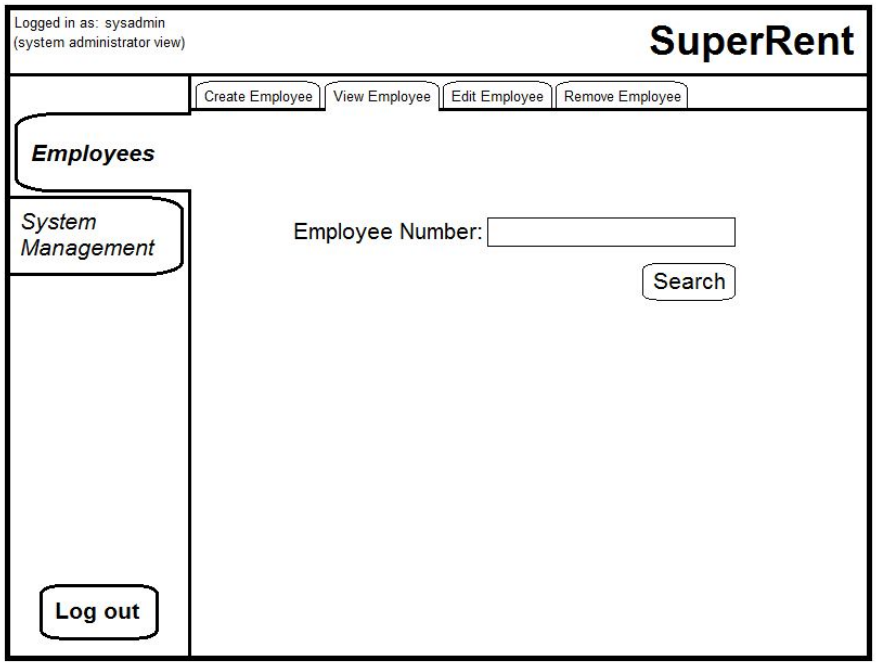





Logged in as: guest09218
(customer view)
SuperRent
Make Reservation
Edit Reservation
View Current Reservations
View Old Reservations
Reservations
Location:
Pick up date:
Return date:
Vehicle:
Equipment:
Reserve Vehicle
Log out


Logged in as: simrand00
(employee view)
SuperRent
Make Reservation
Edit Reservation
View Current Reservations
View Old Reservations
Reservations
Rentals
Customers
Sales Reports
Reservation Number:
Last name:
Search
Log out


Logged in as: gman01
(manager view)
SuperRent
Pick up Rental
Return Rental
Edit Rental
Make Incident Report
Make Overdue Report
Reservations
Rentals
Customers
Sales Reports
Vehicles
Log out
Rental Number:
Search


Logged in as: sysadmin
(system administrator view)
SuperRent
Create Employee
View Employee
Edit Employee
Remove Employee
Employees
System Management
Log out
Employee Number:
Search

4. Glossary

API: Application Programming Interface.

Customer: A user interacting with the system in hopes of making a reservation for a car rental.

Employee: A user interacting with the system, capable of making reservations for Customers, confirm/return rentals, view/edit sales reports, etc.

Fleet: The whole sum of vehicles available at a given location.

GUI: Graphical User Interface.

HTTPS: HyperText Transport Protocol Secure.

Incident Report: A formal form documenting an incident occured on a vehicle and to be reported.

Manager: A user interacting with the system, capable of making reservations for Customers, confirm/return rentals, view/edit sales reports, view/edit vehicles and promotions, etc.

Membership: A customer who can receive and make points for exchange in future rentals, paid for on application.

Promotional Code: A unique code assigned to a discount valid during times of promotional events.

Reminders: An automated message sent to customers reminding them on details about their reservation or rental.

Rent: The exchange of money (or membership points) for the use of a vehicle during a predetermined period of time.

Reservation: An instance of requesting a vehicle to rent during a future period of time.

SQL: Structured Query Language.

SSL: Secure Sockets Layer.

SuperRent: "the company", "the store".

System Administrator: A user interacting with the system, capable of view/editing employees, backing up the database, etc.

TurboRent: Software name.

URL: Uniform Resource Locator.

Vehicle Identification Number: A number used to distinguish a vehicle from others.

5. References

A huge thanks to the people below for helping us by providing very crucial information on their day to day interactions with their rental programs, and for taking the time out of their day to help provide us with this information.

Discount Car and Truck Rentals

Raj Singh

8326 St George Street

Vancouver, BC

https://www.discountcar.com

Dollar Thrifty Automotive Group Canada Inc.

Simran Dhillon

#3-790 SW Marine Drive

Vancouver, BC

https://www.thrifty.com

Hertz Canada Limited

Charvon Iza

#3-790 SW Marine Drive

Vancouver, BC

https://www.hertz.com