
SuperRent

Version management

Date	Author	Version	Comments
02/05/2015	Team 01	1.0	Initial version
03/01/2015	Alexander	1.1	Modification of use case
03/04/2015	Dandan	1.2	Modification of format
04/01/2015	Yaoyao&Cindy	1.3	Modification of static model & dynamic model
04/03/2015	Inian&Mohammad	1.4	Modification of system design
04/05/2015	Dandan	1.5	Modification of Database design
04/09/2015	Dandan	1.6	Modification of testing activities
04/13/2015	Cindy	1.7	Modification of reference& implementation
04/15/2015	Team 01	2.0	Final version

Contents

1 Introduction

1.1 System Purpose

The purpose of this super rent system is to present an application which can provide services of recording, assisting and managing kinds of activities of the company Super Rent. The main function of this application is designed to improve the efficiency of managing vehicle rental business. This project focuses on the development of user-friendly and feasible application that allows employees to efficiently carry out various operations, such as vehicle rental, fleet management and customer management.

1.2 System function

The system can manage multiple branches of the company's vehicle reserving and returning activities and record the relevant data. Additionally, it can handle all the customers, employees and vehicles information management. The client is only open to store employee and customers need to reserve and return of the vehicle through the clerk. Managers can manage and check all vehicles, customers and employees' information. The system can calculate and record all transactions of credit card or cash payments, but the certified checking process is not included in the system.

1.3 Objective and limit of the project

The objective of this project is to provide a fully functional and user-friendly management system for the company to manage all the vehicles, customers and employees' information. The final software product should meet the requirements of all the regular features as well as improve all functions in order to improve the user experience and efficiency.

1.4 References

https://netbeans.org/kb/index.html

http://testng.org/doc/index.html

http://en.wikipedia.org/wiki/Scrum_(software_development)

http://en.wikipedia.org/wiki/Agile_software_development#Overview

http://en.wikipedia.org/wiki/TestNG

R. Fagin. Normal forms and relational database operators. In Proc. A CM SIGMOD Conf. on the Management of Data, 1979.

N. G. Leveson, Safeware: System Safety And Computers, Addison-Wesley, Reading, MA, 1995.

J.-L. Lions, ARIANE 5 Flight 501 Failure: Report by the Inquiry Board, http:/ /ravel.esrin.esa.it/docs/esa-x-1819eng.pdf, 1996.

B. Liskov, "Data abstraction and hierarchy," SIGPLAN Notices, Vol. 23, No. 3, May, 1988.

B. Liskov & J. Guttag, Abstraction and Specification in Program Development, MIT Press, McGraw-Hill, New York, 1986.

Lotus, http://www.lotus.com/. L. Macaulay, Requirements Engineering, Springer-Verlag, London, 1996.

A. MacLean, R. M. Young, V. Bellotti, & T. Moran, "Questions, options, and criteria: Elements of design space analysis," Human-Computer Interaction, Vol. 6, pp. 201–250, 1996.

R. Marshall, What really happened at K2? Chapter 24 in [Bonatti 2001].

1.5 Overview

The purpose of this document is to summarize the entire project and present an overall description of the system. The document is organized as follows: The first section (this section) presents an introduction to the project. Section 2 presents all the specific requirements of the project and Section 3 presents the system design. Section 4 presents the system design and the database design of the project. Section 5 gives an instruction of the implementation parts and Section 6 presents the performance of the system. Additionally, Section 7 presents the test activities of the system and Section 8 presents the system's management activities. Finally, Section 9 introduces the supporting information.

2 Specific requirements

2.1 Functional requirements

This sections describes the functional requirements for users based on various positions.

2.1.1 Customer

- ❖ The customer shall be able to apply for the Club membership. ❖ The customer shall be able to log in.
- ❖ The customer shall be able to check the availability of vehicles by providing the location, the type of the vehicle and the day and time for which she/he would like to pick up/return the vehicle and additional options.
- ❖ The customer shall be able to make reservations by providing the location, the type of the vehicle and the day and time for which she/he would like to pick up/ return the vehicle and additional options.
- ❖ The customer shall be able to check the estimation of the cost per one reservation.
- ❖ The customer shall be able to get a reservation number.
- ❖ The customer shall be able to cancel reservations by providing either the confirmation number or their phone number and the dates.
- ❖ The customer shall be able to check the reservations.
- ❖ The customer shall be able to rent a car or truck by providing the confirmation number or their phone number.
- ❖ The customer shall be able to check the points she/he has accumulated.

2.1.2 Clerk

- The clerk shall be able to log in.
- The clerk shall be able to set up new accounts for new customers.
- The clerk shall be able to check reservations.
- The clerk shall be able to calculate costs.
- The clerk shall be able to help the customer to process a rental by checking his/her reservation, driver's license number and credit card information consisting of the card number and expiry date.
- The clerk shall be able to help the customer to return a vehicle by entering date, the time, the odometer reading and whether the gas tank is full.
- The clerk shall be able to check points of a customer and reduce the cost by using the points.
- The clerk shall be able to check the vehicles of a specified category that are available in a given location for a given set of dates (usually given as from-date and to-date).
- The clerk shall be able to check the vehicles in a specified location and category that are overdue.
- The clerk shall be able to check the vehicles in a specified location and category that are for sale and their sale prices.

2.1.3 Administrator

- The DBA shall be able to add users.
- The DBA shall be able to remove users.
- The DBA shall be able to change users' passwords.

2.1.4 Manager

- The manager shall be able to add vehicles.
- The manager shall be able to remove vehicles from rental list to sale list.
- The manager shall be able to remove vehicles from sale to rental list.
- The manager shall be able to generate reports based on the company's requirements, including daily rentals, daily rentals for branch, daily returns and daily returns for branch.
- The manager shall be able to set all the rates and costs.
- The manager shall be able to set the points redeem policy.
- The manager shall be able to check the vehicles in a specified location and category that are older than a specified number of the year. If the location or category is left out all qualifying vehicles are shown grouped by category and/or location.

2.1.5 C.C Verifier

- The C.C. Verifier shall be able to verify a transaction.
- The C.C. Verifier shall be able to process a transaction.

2.2 Non-functional requirements

This section describes the non-functional requirements of the system.

2.2.1 Usability

- Easy/friendly GUI interface for Customer (i.e. help option)

2.2.2 Reliability

- Must be able to handle concurrent actions requests (such as reservation, rents, etc)
- System can recover from crash/failure

2.2.3 Performance

- Store and query data efficiently
- Rapid GUI User/Event Response time
- Screen refresh time (e.g. display updated info. every 1 sec)
- Process orders rapidly (transaction/second)

2.2.4 Supportability

- Allow for centralizing administration of data for DBA purposes.
- Implementation:
- Back-up server to provide data redundancy.
- Database should provide an abstract view of data such that application program are not able to directly access data structures.
- Establish external schema for customizing data access.
- Incomplete transactions are removed from database as part of system restore.
- Locking protocol is established to ensure the net of effects of all transactions are serial in order.
- Accept only CC payments: AMEX, MasterCard, Visa.

2.2.5 Operation

- Ensure data integrity during storage into DB (e.g. enforce integrity

constraints - updated fields in a table do not interfere with constraints in other tables/fields).

2.2.6 Security

- Enforce access control that governs what data is visible to different classes of users.

2.2.7 Performance

- Crash Recovery: Protect applications/users from effects of system failures.
- Generate reports with-in 1 minute time frame.
- Report must contain update to information from database.
- Must be able to handle generation of large loads
- Perform CC verification.

2.2.8 Legal

- Maintain encrypted payments.
- Keep customer information confidential.
- Maintain vehicle registration with local/municipal authorities
- Vehicles in inventory must confer to applicable environmental standards.
- Ensure all customers purchase insurance.

2.2.9 Organizational Policy

- Ensure duplicate keys are held for all vehicles in inventory.
- Maintain new vehicles in inventory.
- Customers must have good standing (Credit verification, overdue balances, etc)

2.3 Scenarios

Use case name	**Rent without reservation**
Participating actors	***Customer*** ***Clerk*** through .
Flow of events	1. *Customer* visits rental outlet. 2. *Customer* interacts with *Clerk* to inquire about renting details including cost, legal requirements, vehicle availability, and available accessories 3. *Clerk* submits login request including his credentials to *SuperRent*. 4. *SuperRent* confirms the request by granting clerk-level access to the system (include use case ***Login***) 5. *Clerk* requests to check most up to date information regarding fleet availability and pricing 6. *SuperRent provides corresponding information (include use case* ***Calculate Cost****, and* ***Query Database****).* 7. *Customer* makes a rental request through the *Clerk* who inputs relevant information into the system. 8. *SuperRent ensures that the proposed rental request is possible. (include use case* ***Query Database****,* ***Check reservation, provides other***

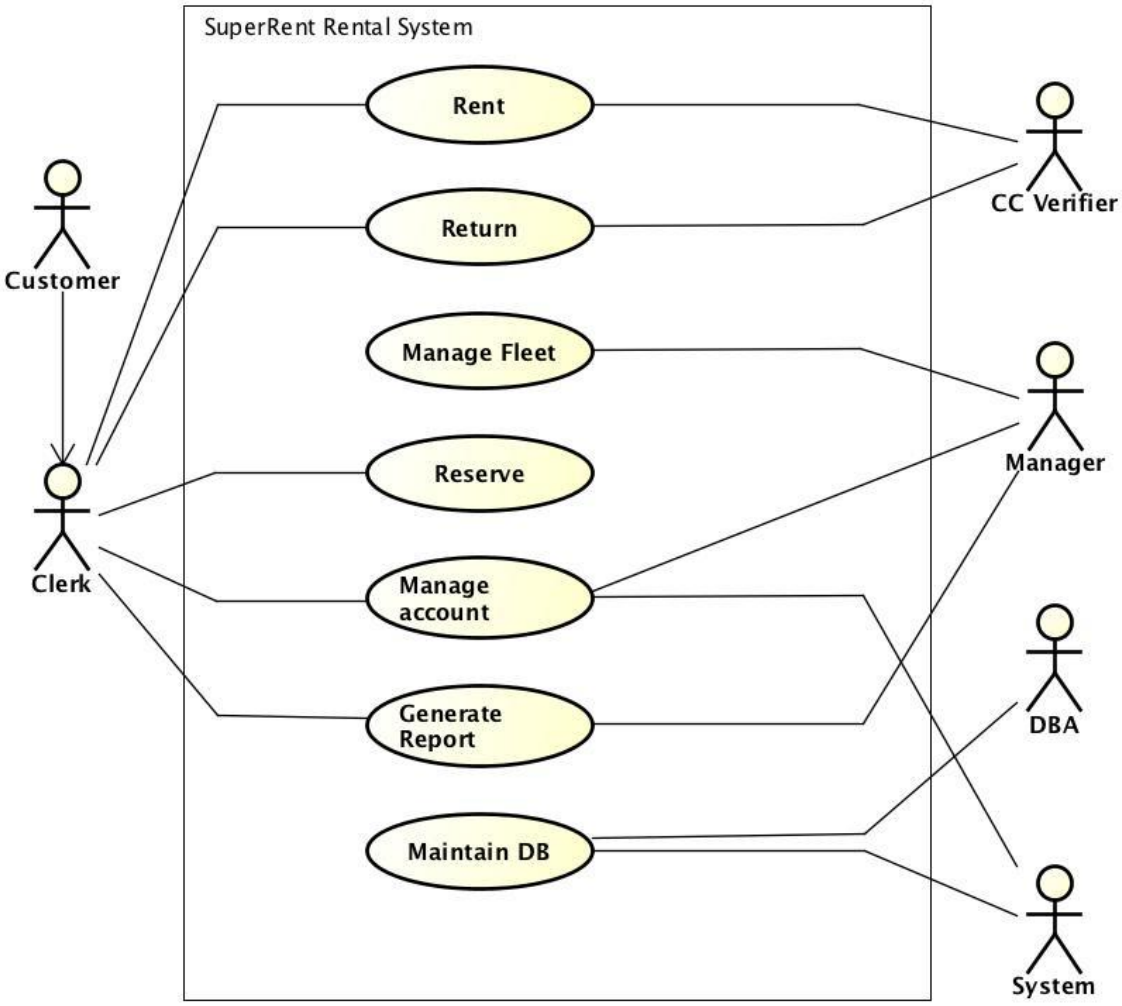
	possible choices**)* 9. If the proposed request is not available, the *Clerk* informs the *Customer* who makes a new request. 10. When an available rental request is made by the *Clerk*, the *Clerk* requests the *Customer*'s profile. *11. SuperRent reports whether the Customer already owns an account (include use case **Check registration**).* 12. If *Customer* so desires, the *Clerk* send an account registration request to *SuperRent*. *13. SuperRent provides a form for account registration.* 14. *Clerk* gathers related information and payment information from the *Customer* and input into *SuperRent*. *15. SuperRent replied success by creating a new account or rejected request if information is not valid. (include use case **Add a new club member**)* 16. *Clerk* prepares a rental agreement. (include use case **Process rental agreement**). 17. *Customer* reviews and signs the rental agreement. 18. *Clerk* acknowledge *SuperRent* the completion of the transaction. *19. SuperRent updates its database information and provides a confirmation message. (include **Validate information with DB**, **Accept

	***Changes**, **Input information**, **Update Database**)* 20. *Customer* leaves the outlet with the rental.
Entry	***Customer*** visits a rental outlet looking to rent a car and there is a available to assist him.
Exit condition	***Customer*** has left the outlet with or without a rental car.

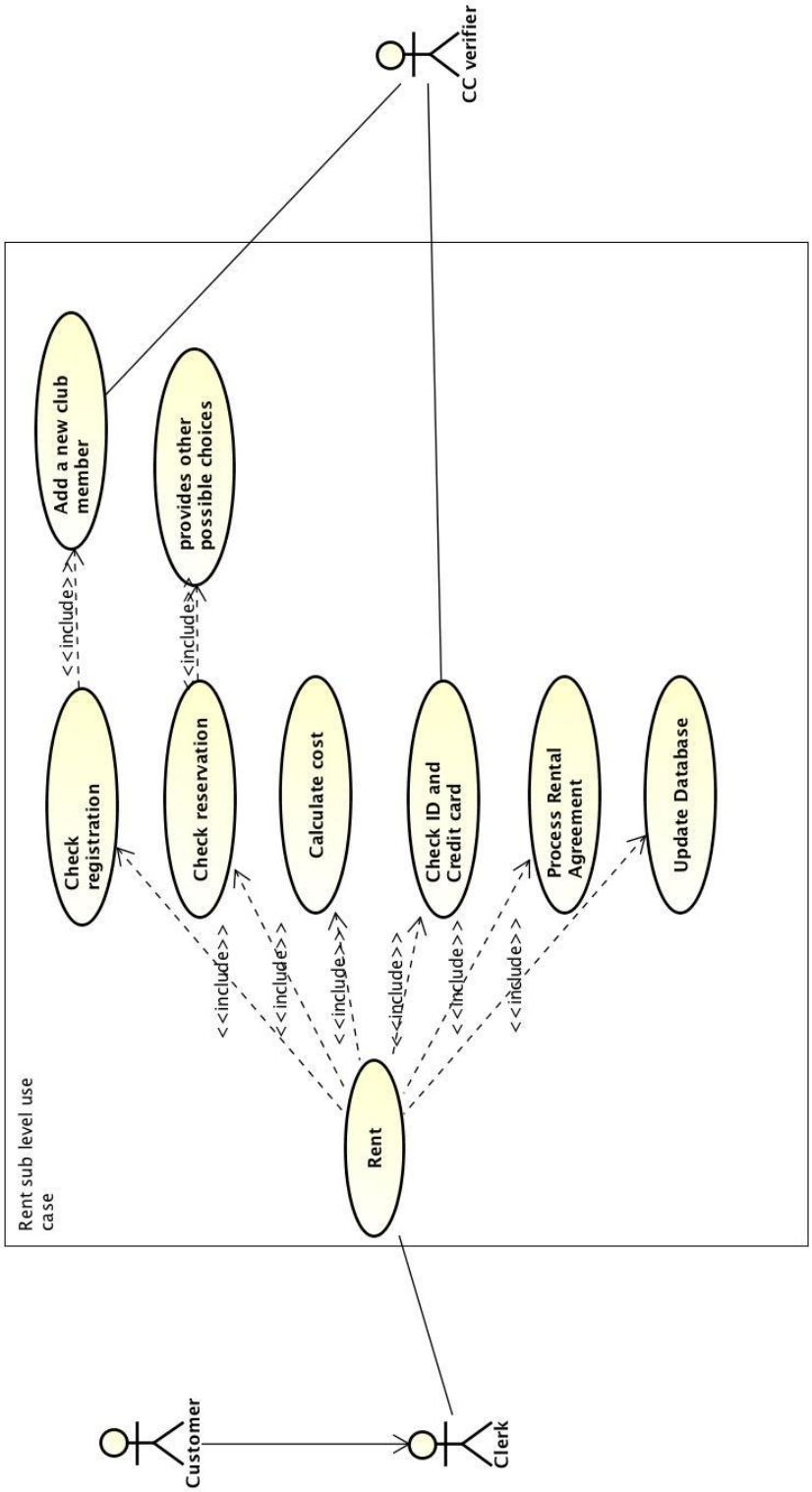
3 System design

3.1 Use case diagram

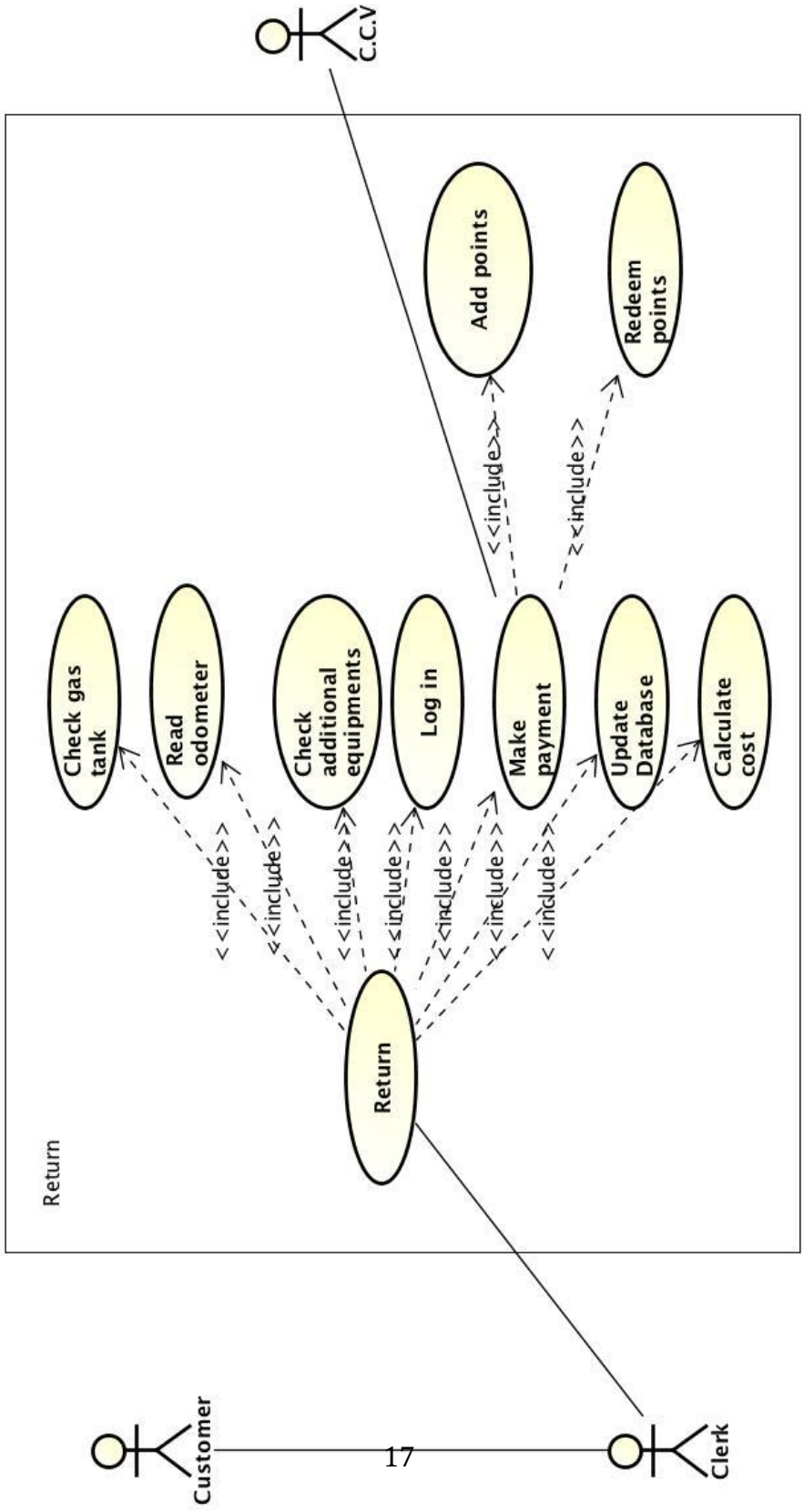
3.1.1 Top level use case



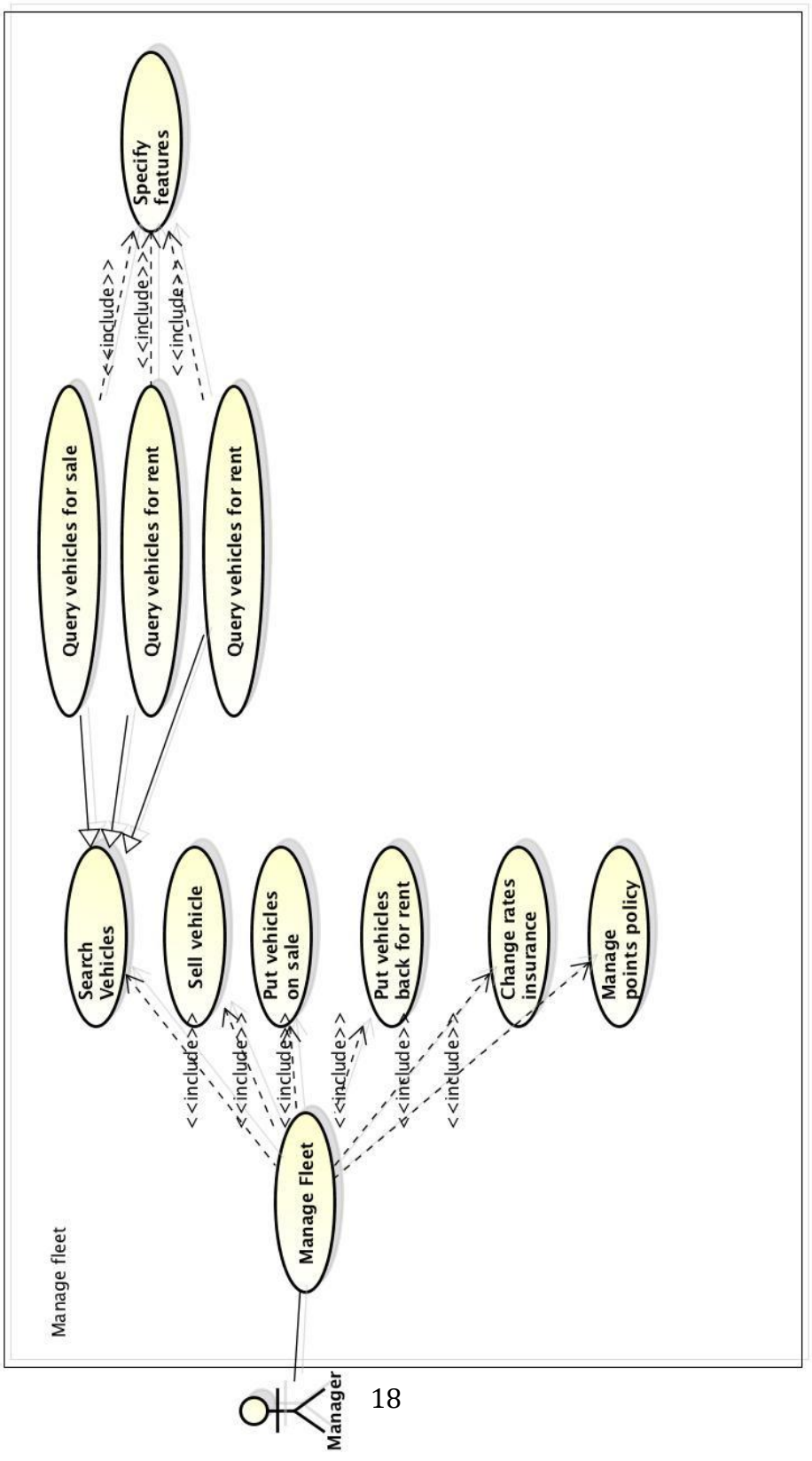
3.1.2 Rent


CC verifier
Add a new club member
provides other possible choices
<<include>>
<<include>>
Check registration
Check reservation
Calculate cost
Check ID and Credit card
Process Rental Agreement
Update Database
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
Rent
Rent sub level use case
Customer
Clerk

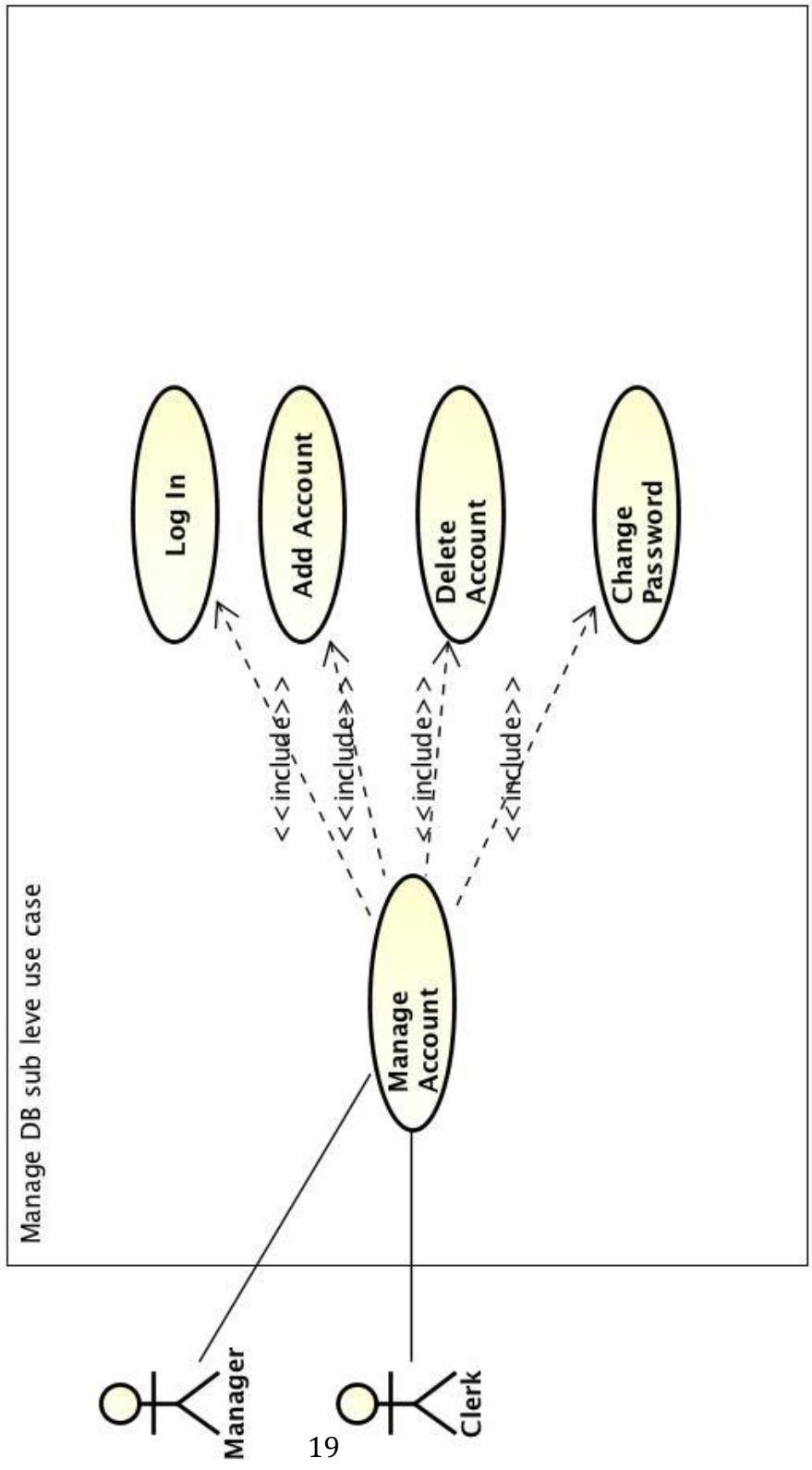
3.1.3 Return


C.C.V
Add points
Redeem points
<<include>>
<<include>>
Check gas tank
Read odometer
Check additional equipments
Log in
Make payment
Update Database
Calculate cost
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
Return
Return
Customer
Clerk

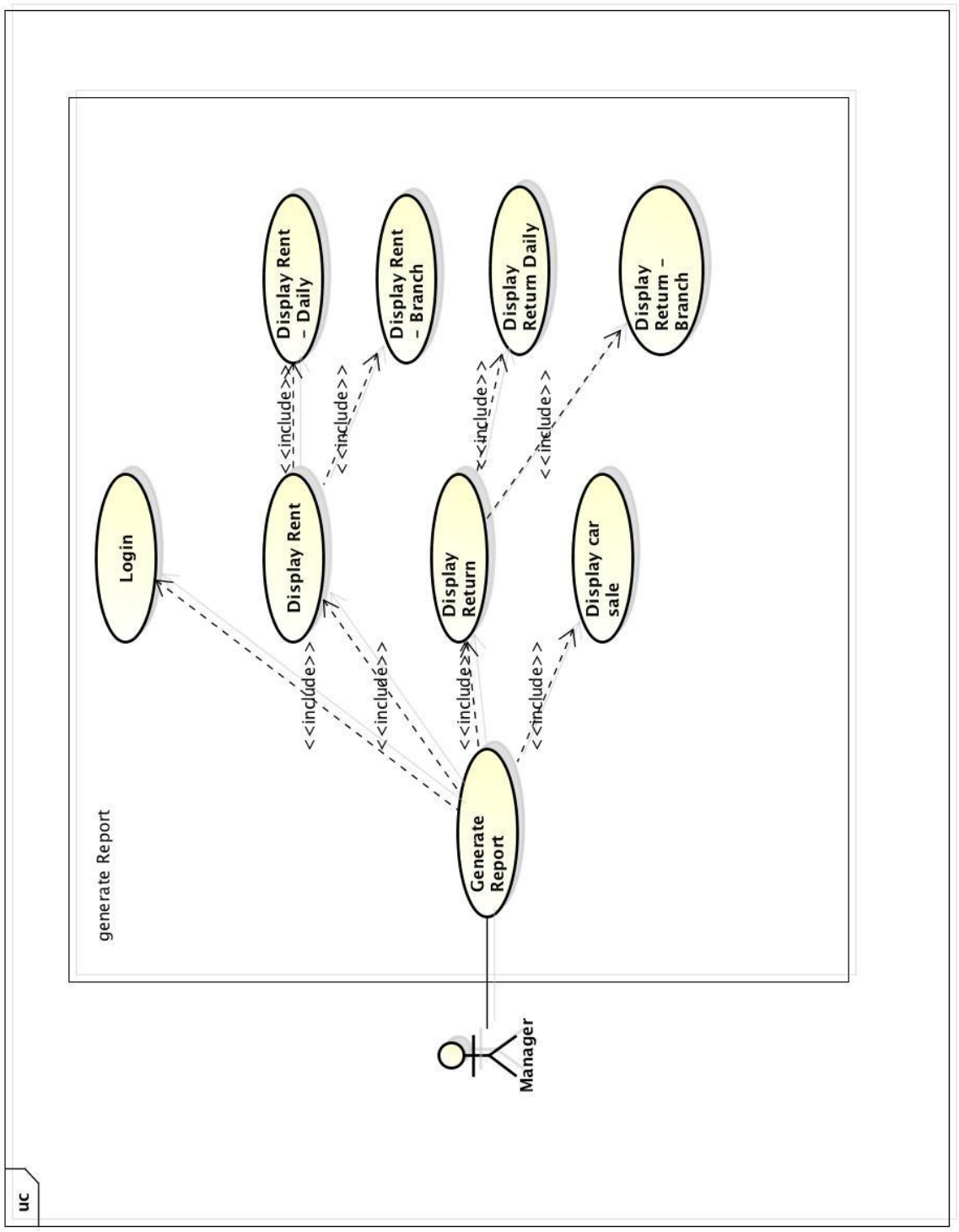
3.1.4 Manage fleet



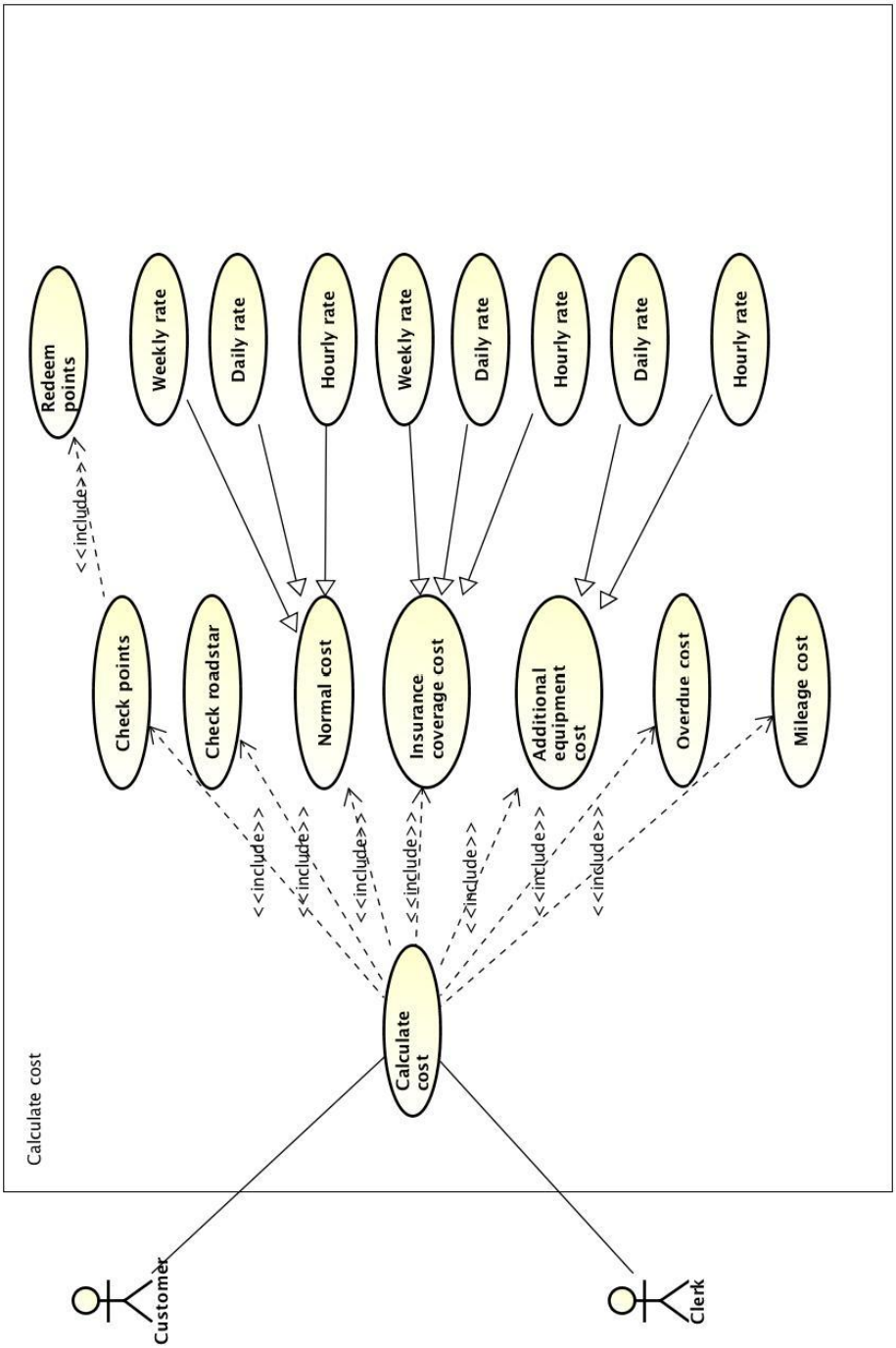
3.1.5 Manage account



3.1.6 Generate Report

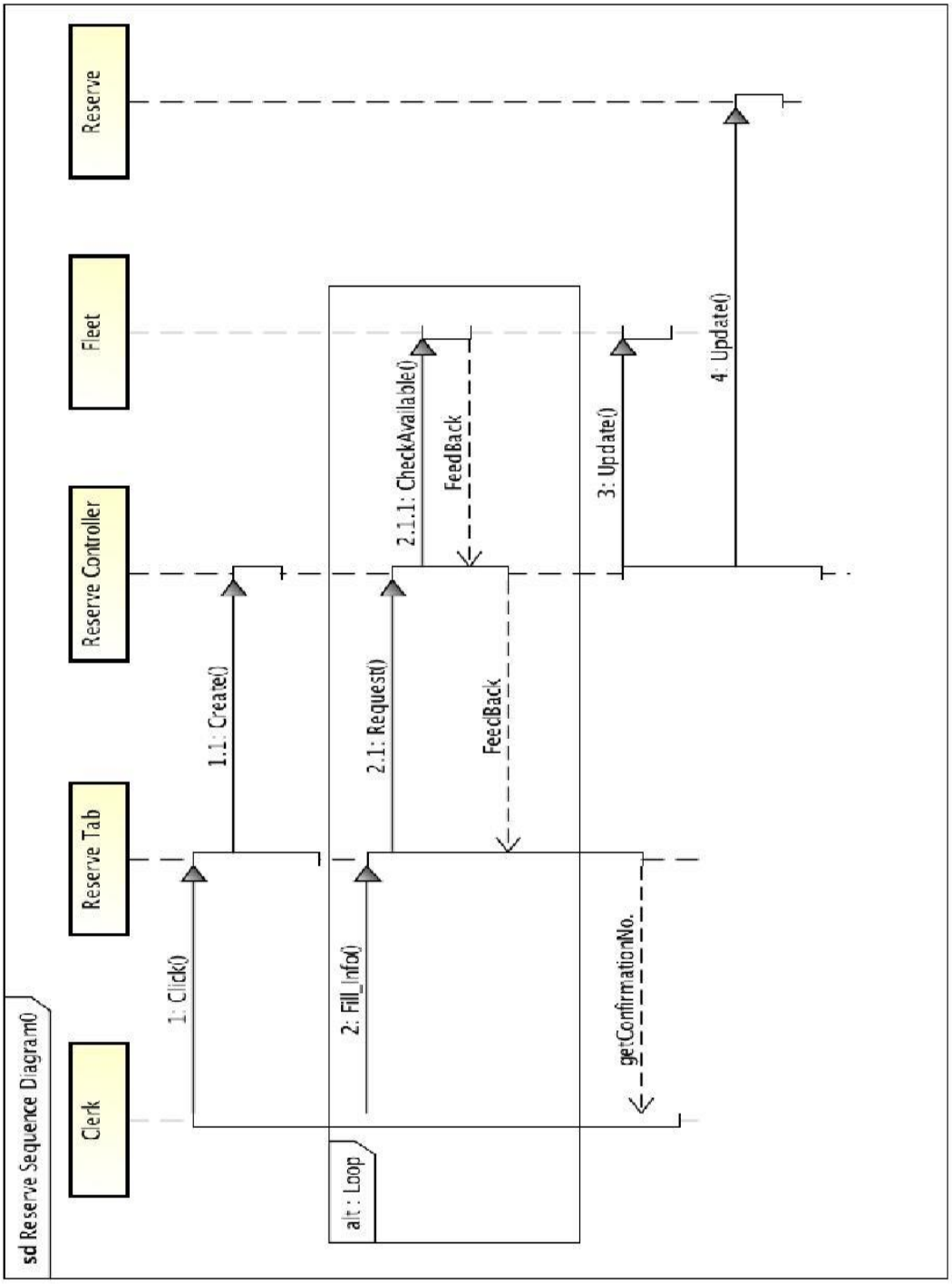


3.1.7 Calculate price

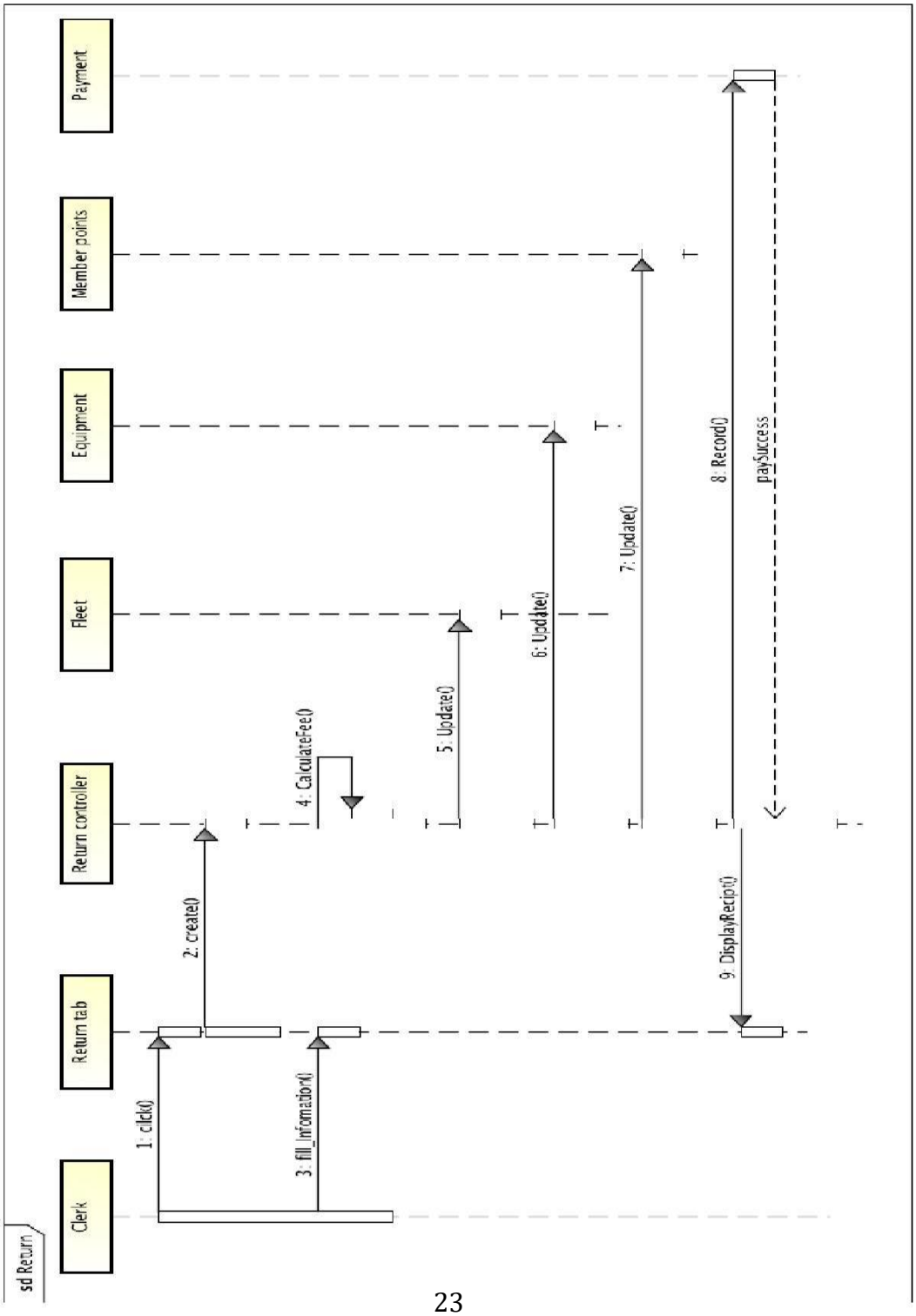


3.2 Sequence diagram

3.2.1 Reserve sequence diagram

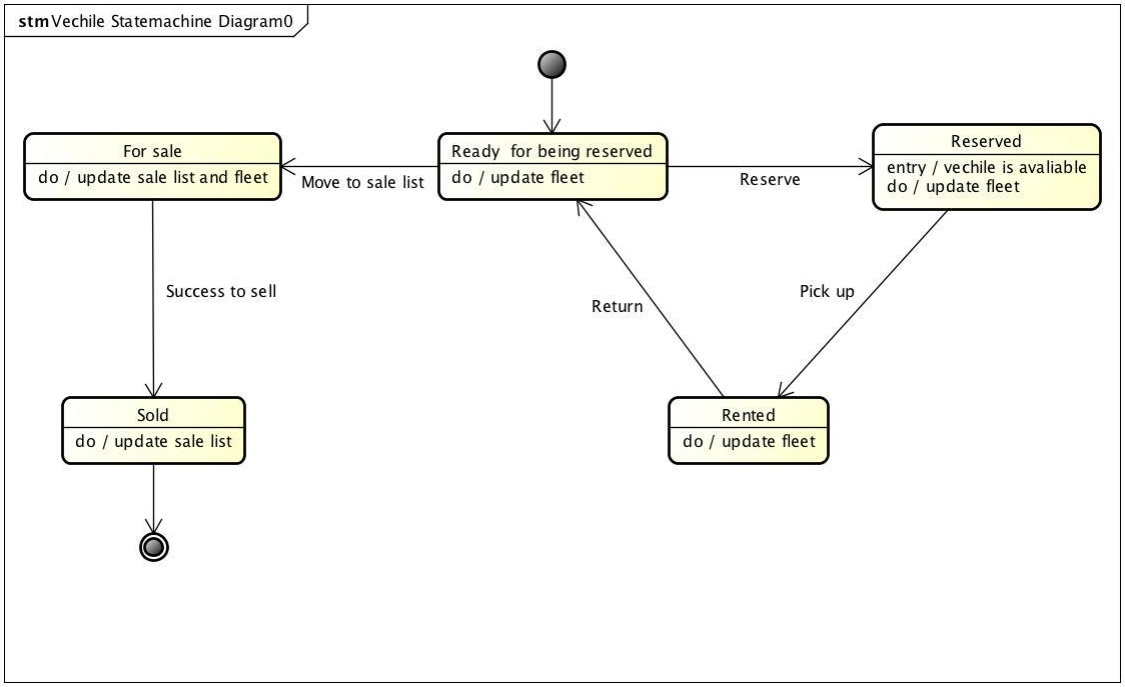


3.2.2 return sequence diagram



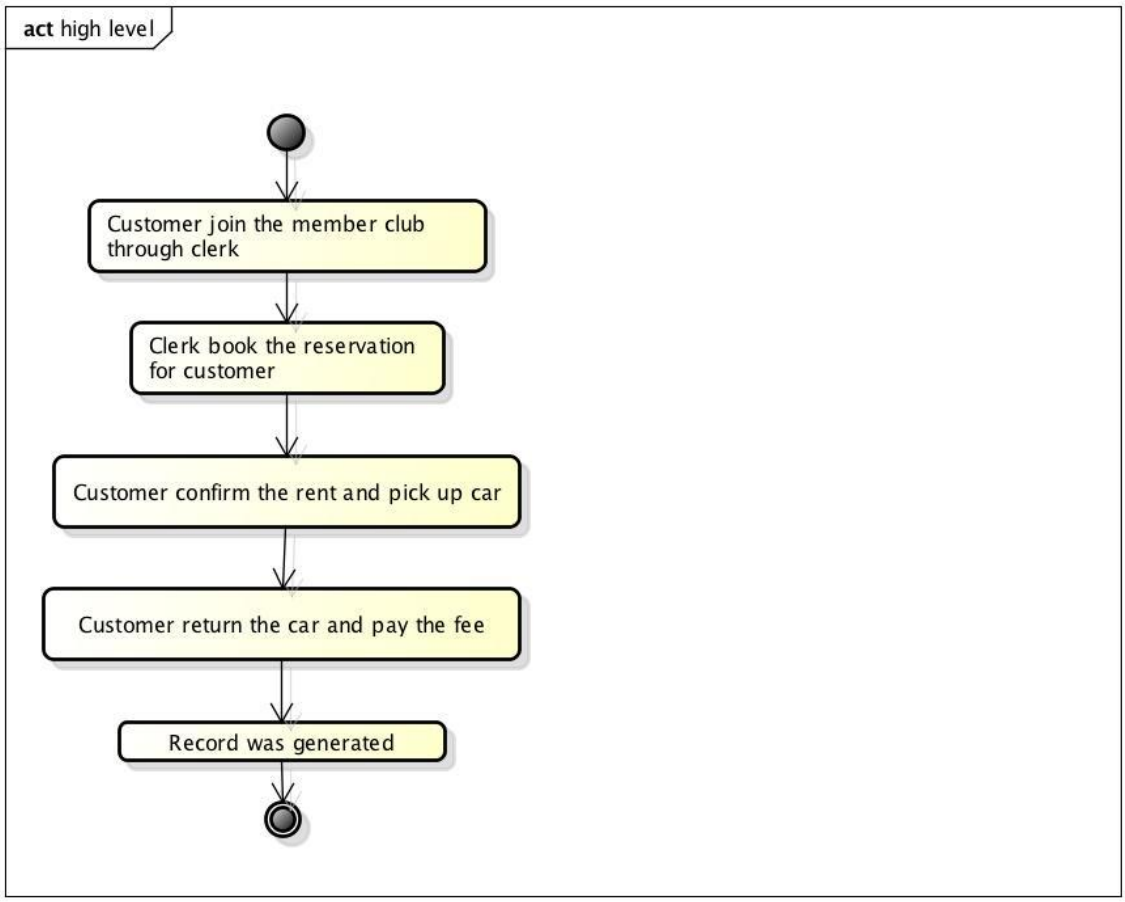
3.3 State machine diagram

3.3.1 vehicle state machine diagram

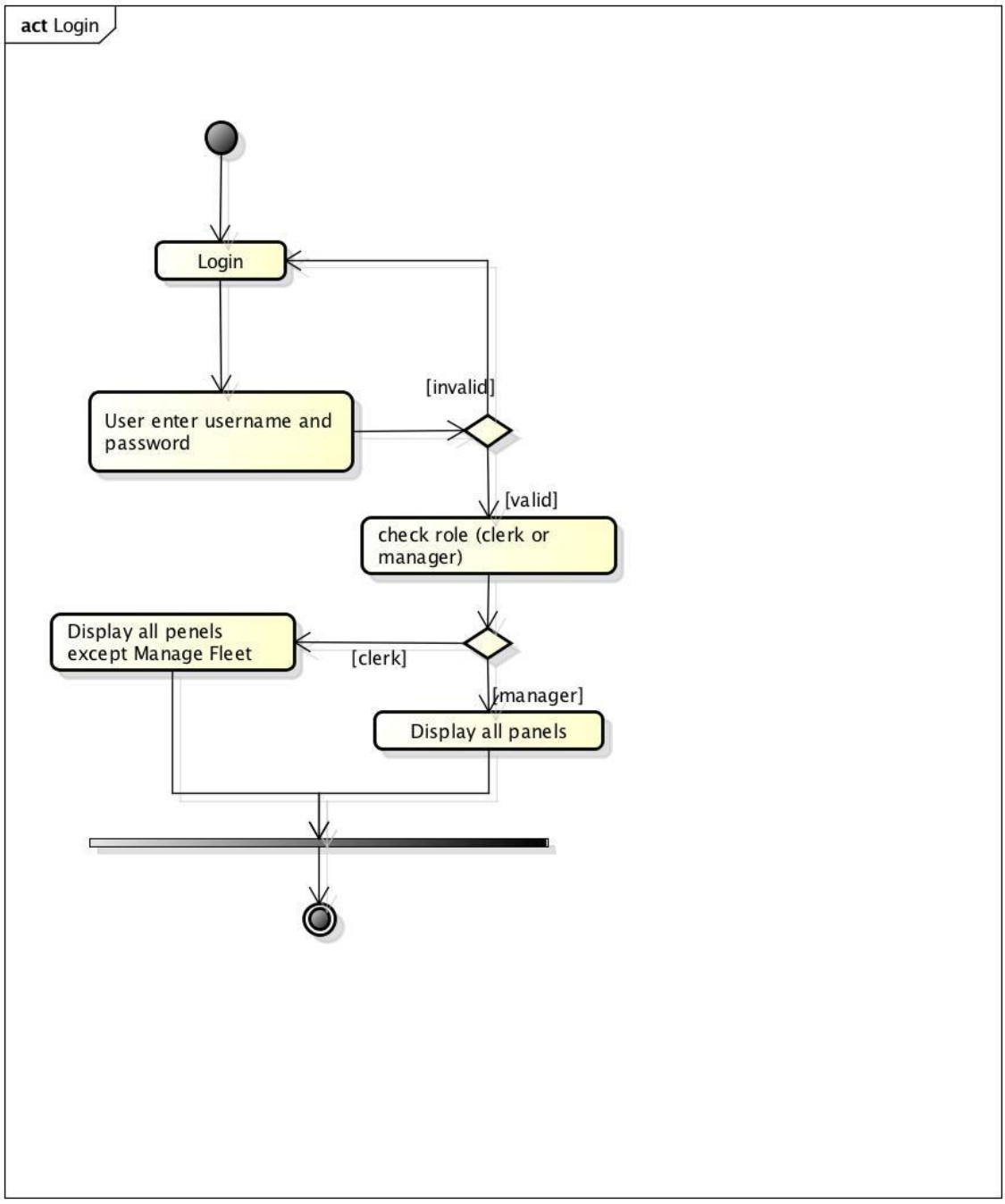


3.4 Activity diagram

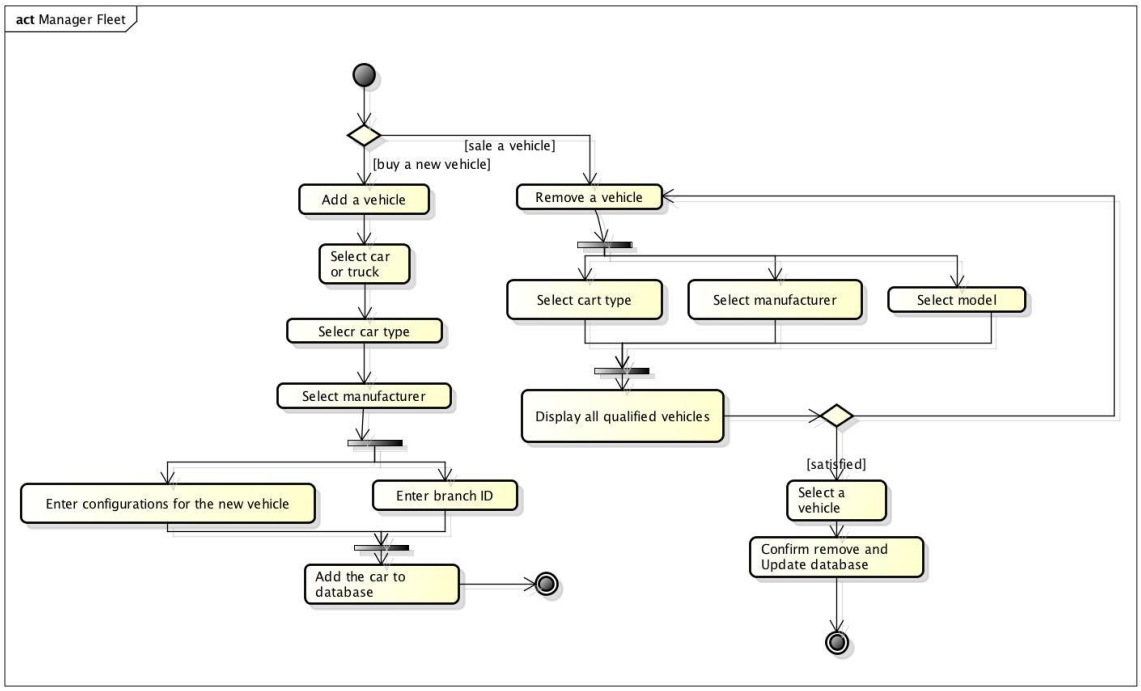
3.4.1 High level



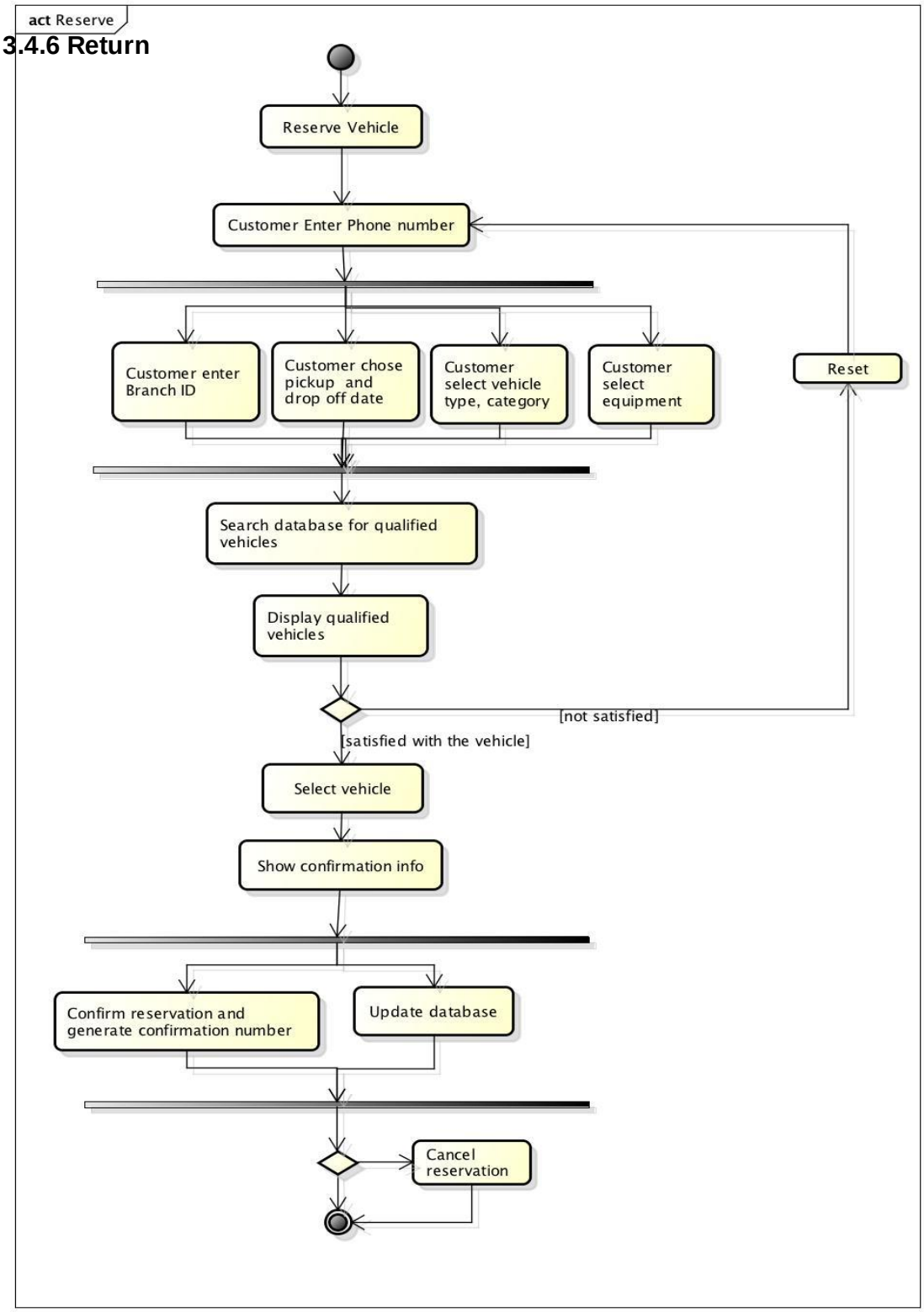
3.4.2 Login


act Login
Login
User enter username and password
[invalid]
[valid]
check role (clerk or manager)
Display all penels except Manage Fleet
[clerk]
[manager]
Display all panels

3.4.3 Manage fleet

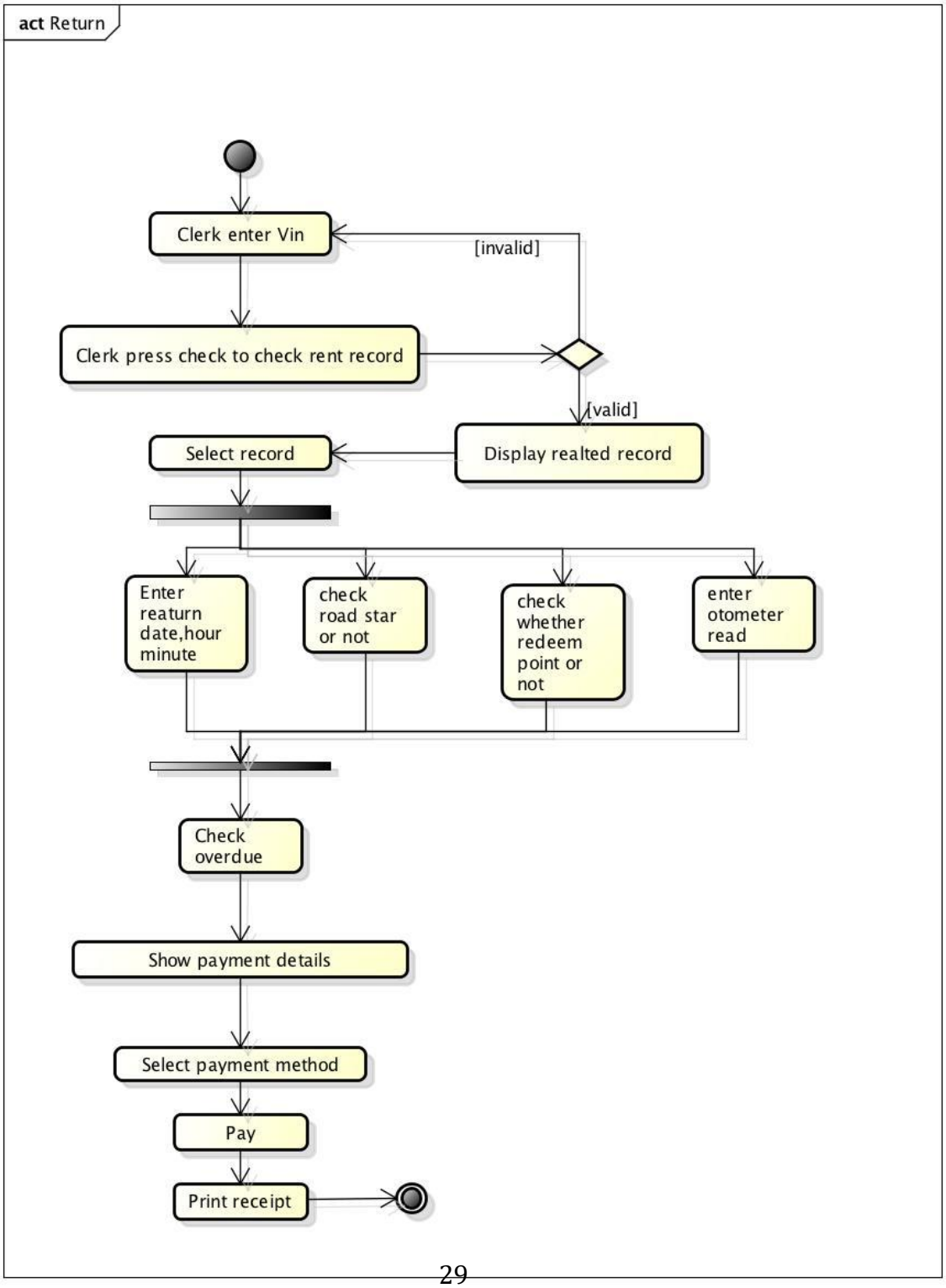


3.4.4 Reserve

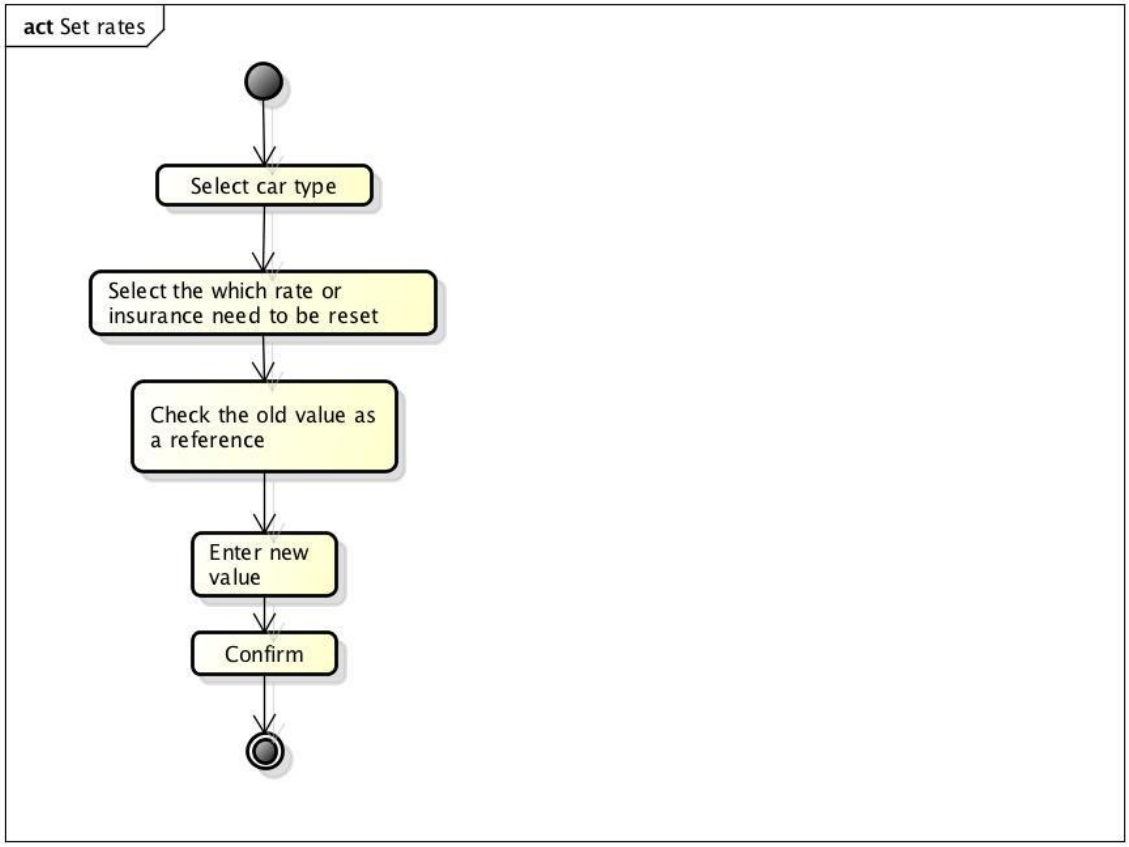


3.4.6 Return

3.4.5 Return

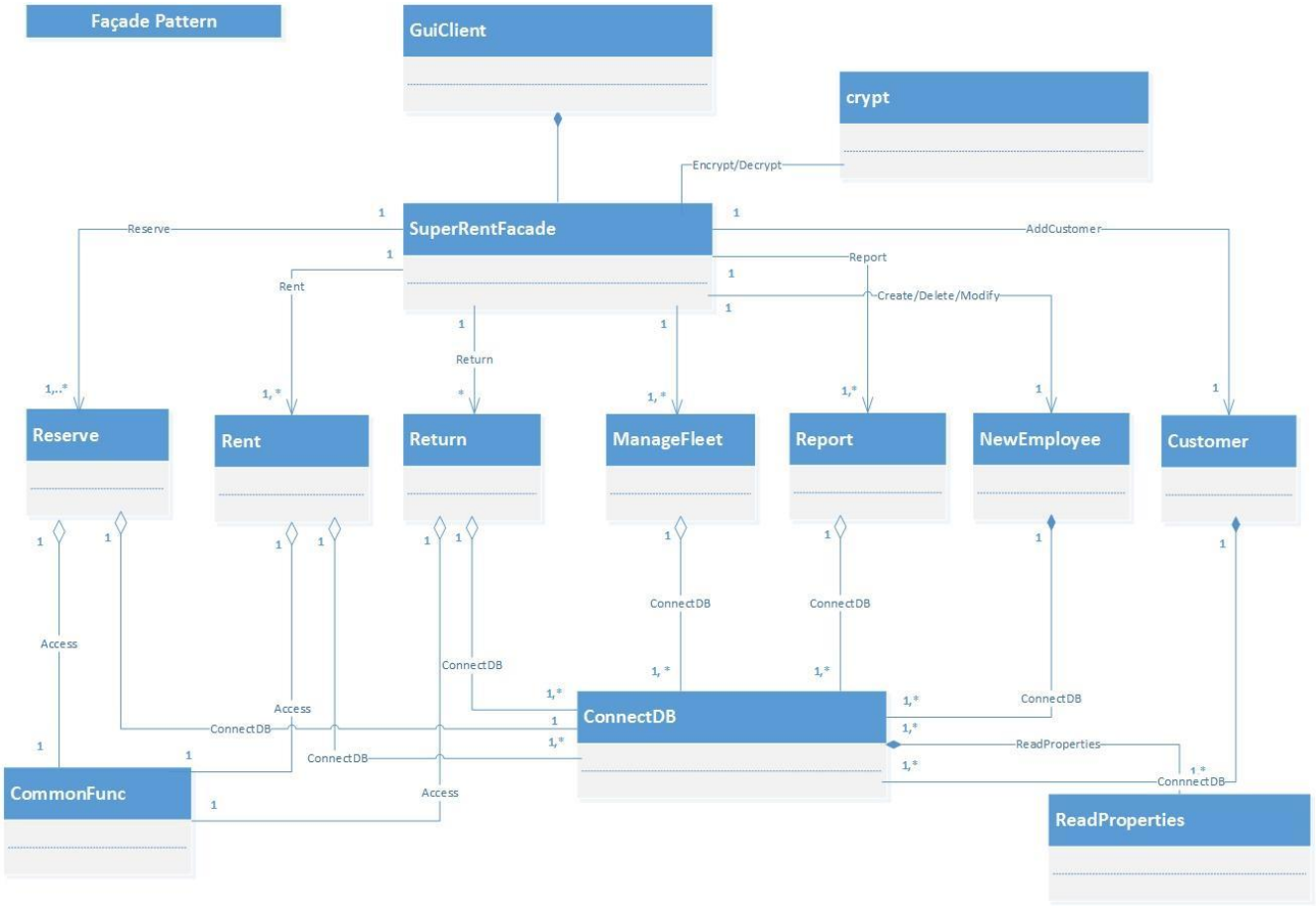

act Return
Clerk enter Vin
[invalid]
Clerk press check to check rent record
[valid]
Select record
Display realted record
Enter reaturn date,hour minute
check road star or not
check whether redeem point or not
enter otometer read
Check overdue
Show payment details
Select payment method
Pay
Print receipt

3.4.6 Set rates



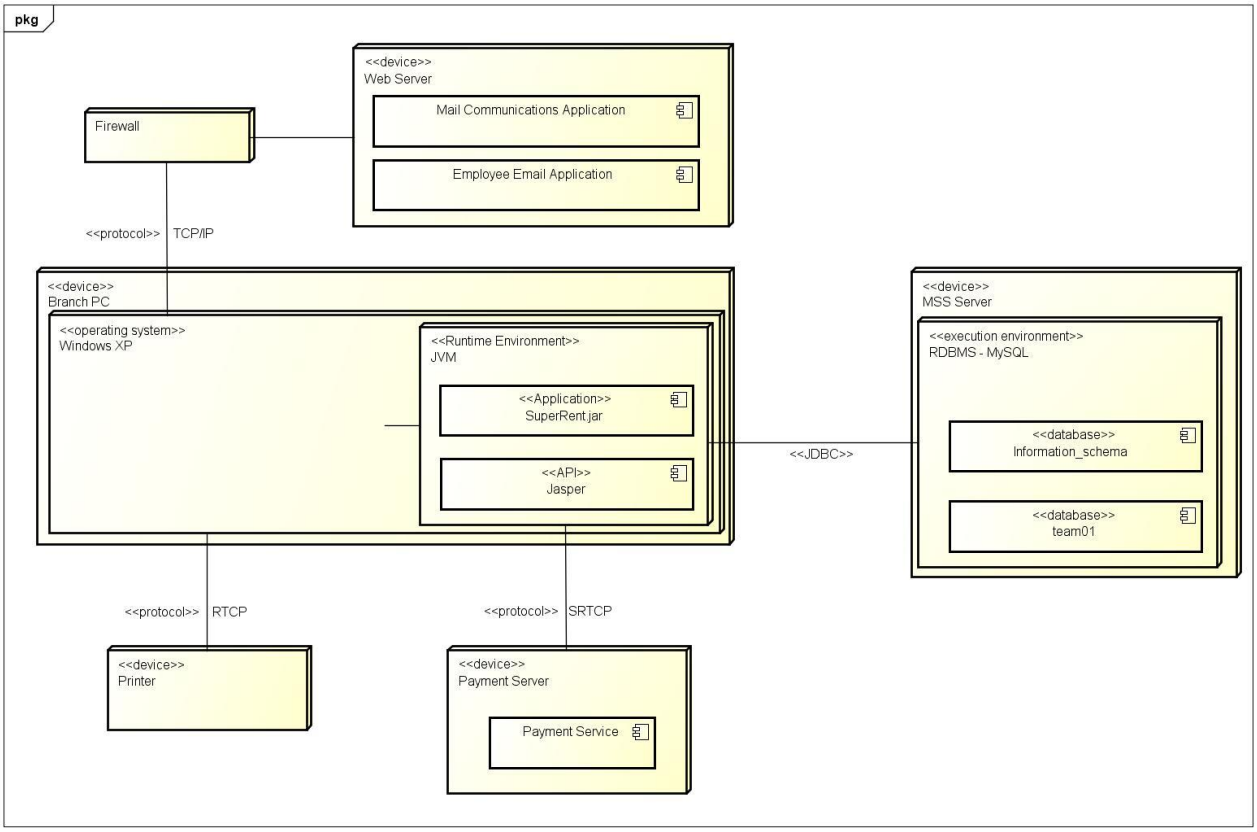
3.5 Class diagram

3.5.1 Design pattern

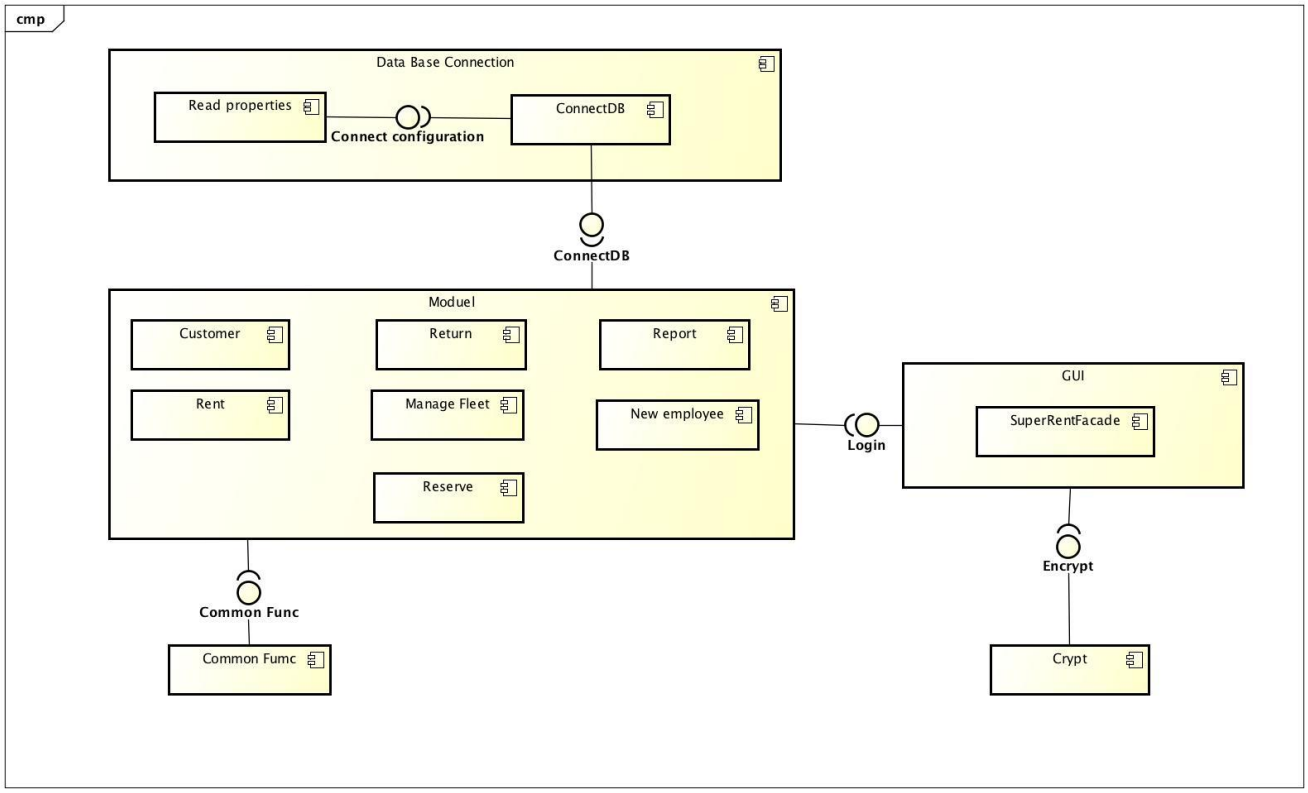



SuperRent
+ SpinnerModel modelPickHH
+ SpinnerModel modelPickMM
+ SpinnerModel modelDropHH
+ SpinnerModel modelDropMM
+ Calendar now
- Reserve rt
- String role
- Rent rent
- ArrayList equipSelected
- HashMap equipSelectedUnits
- ManageFleet mf
- JTabbedPane ModulesTab
- JPanel NewEmployee
- JPanel Rent
- JPanel Reports
- JTextField ReturnCash_jTextField2
- JButton ReturnCheckOverDue_jButton2
- JButton ReturnCheck_jButton2
- JTable ReturnCheckOverDueDis_jTable2
- JTextField lastNameField
- JLabel lastNameLbl
- JButton loginBtn
- JPanel loginpanel
- JPanel manageCustomer
- JPanel manageFleet
- JPanel managepointjpanel
- JPanel mfaddpanel
- JComboBox mfbranchCombo
- JCheckBox mfcarCheckBox
- JList selectedList
- JButton setforrentBtn
- JButton setforsaleBtn
- JTextField streetAddField
- JTable unreservedVehiTable
- JTextField userField
- JTextField userNameField
- JLabel userNameLbl
- JComboBox vehicleCatCombo
- JComboBox vehicleTypeCombo
- JTextField zipTextField
- JTextField firstNameField
- JLabel firstnameLbl
- JCheckBox jCheckBox3
- JLayeredPane jLayeredPane1
- void addBtnActionPerformed(ActionEvent)
- void addEquipBtnActionPerformed(ActionEvent)
- void addEmployeeBtnActionPerformed(ActionEvent)
- void brnchIdComboItemStateChanged(ItemEvent)
- void cancelConfirmationActionPerformed(ActionEvent)
- void clearEqupList()
- void clerkComboBoxActionPerformed(ActionEvent)
- void initComponents()
- void initEmployee()
- void initManageFleet()
- voidinitRent()
- void initReturn()
- void jTextField1ActionPerformed(ActionEvent)
- void loginBtnActionPerformed(ActionEvent)
- void mfcarCheckBoxActionPerformed(ActionEvent)
- void mfcolorComboActionPerformed(ActionEvent)
- void mfmanufacturerComboActionPerformed(ActionEvent)
- void mfremoveBtnActionPerformed(ActionEvent)
- void mfs1BtnActionPerformed(ActionEvent)
- void mfs2BtnActionPerformed(ActionEvent)
- void mfs3BtnActionPerformed(ActionEvent)
- void mfscartypeComboActionPerformed(ActionEvent)
- void mfset1BtnActionPerformed(ActionEvent)
- void mfset2BtnActionPerformed(ActionEvent)
- void mfset3BtnActionPerformed(ActionEvent)
- void mfsetcartypeComboActionPerformed(ActionEvent)
- void mfsetmanufacturerComboActionPerformed(ActionEvent)
- void mfsmanufacturerComboActionPerformed(ActionEvent)
- void mftruckCheckBoxActionPerformed(ActionEvent)
- void ModulesTabMouseClicked(MouseEvent)
- void removeBtnActionPerformed(ActionEvent)
- void removeEquipBtnActionPerformed(ActionEvent)
- void rentBtnActionPerformed(ActionEvent)
- void resetBtnActionPerformed(ActionEvent)
- void ReturnCalculate_jButton6ActionPerformed(ActionEvent)
- void ReturnCheck_jButton2ActionPerformed(ActionEvent)
- void ReturnCheckOverDue_jButton2ActionPerformed(ActionEvent)
- void ReturnCheckOverDueDis_jTable2MouseClicked(MouseEvent)
- void ReturnCreditCard_jTextField1ActionPerformed(ActionEvent)
- void ReturnFuelReading_jTextField2ActionPerformed(ActionEvent)
- void ReturnPay_jButton2ActionPerformed(ActionEvent)
- void ReturnRedeem_jCheckBox1ActionPerformed(ActionEvent)
- void ReturnRoadStar_jCheckBox2ActionPerformed(ActionEvent)
- void ReturnVin_jTextField8ActionPerformed(ActionEvent)
- void searchBtnActionPerformed(ActionEvent)
- void searchEqupBtnActionPerformed(ActionEvent)
- void setforrentBtnActionPerformed(ActionEvent)
- void setforsaleBtnActionPerformed(ActionEvent)
- void vehicleCatComboActionPerformed(ActionEvent)
- void vehicleTypeComboItemStateChanged(ItemEvent)
- void zipTextFieldActionPerformed(ActionEvent)
ManageFleet
-int car_or_truck
-String function
-String sqlforInfo = new String()
+ManageFleet(String: function)
+void getCar_or_truck(int :car_or_truck)
+void fillManufacturer(JComboBox :modelCombo)
+void fillBranch(JComboBox :branchidCombo)
+void fillColor(JComboBox: colorCombo)
+void fillCarType(JComboBox : cartypeCombo)
+void fillsCarType(JComboBox : cartypeCombo)
+void fillssModel(JComboBox : modelCombo, String :manufacturer,String: cartype)
+void fillssManufacturer(JComboBox : manufacturerpeCombo, String : cartype)
+ void setsqlInfo (int : i,String : cartype, String: manufacturer, String: model)
+String[][] getonsaleInfo()
+void clearTable(JTable : mfremoveTable)
+void settable(int : i,String : cartype, String: manufacturer, String :model, JTable: mfremoveTable)
+void remove(JTable : mfremoveTable)
+void sendback(JTable : mfremoveTable)
+void showerror(String: s)
+void changepoint(double : cash)
+ void managepoint(int: i, int: j)
Rent
+ int validateConfirmationNo(JTextField :ConfirmationNo)
+ rentVehicle(JTextField : dl, JTextField : ConfirmationNo)
CommonFunc
+ static Date changeDateFormat(JDateChooser :jdate)
+ static int compareDates(Date: first, Date: second)
+static String sqlTime(JSpinner:HH, JSpinner:MM)
+static Date StringToDate(String:MyDate)
+static void triggerMail(String toPerson, String Subject, String Msg)
CalculateRate
+static double CalculateNormalRate(Date:st, Date: end, int: pick,int :drop, float :weekly_rate, float: daily_rate, float :hourly_rate)
+ static double CalculateInsRate(Date: st , Date :end ,int pitime,int drop, float: weekrate, float :daily_rate, float :hourly_rate, boolean: star)
+ static double CalculateEquipRate(Date start, Date:end,int :pitime,int :dropti, float weekly_rate, float daily_rate, float hourly_rate)
Reserve
+ void fillBranchIDCombo(JComboBox : branchCombo)
+ void fillVehicleCatCombo(JComboBox :vehicleCatCombo, JComboBox: branchCombo, JComboBox :vehicleType)
+ DefaultListModel fillEquipmentList(JList: equipments, JComboBox :cartypeCombo, JComboBox: branchid)
+ int validateCustomerInfo(JTextField : cusNumField)
+ static long genUniqueID()
+ DefaultTableModel getAvailableVehicles(String : vehicleCat, int :branchId, String : VehicleType, Date : pickup, Date : dropoff)
+ int countAvailableVehicles(String: vehicleCat, int : branchId, String: VehicleType)
+ void cancelReservation(JTextField : confirmationNo)
Return
+ int getNumberofWeeks(Date : pickupdt, Date : returndt)
+ int getNumberofDays(Date: pickupdt, Date: returndt)
+ int getNumberofhours(Date : pickupdt, Date : returndt)
+ String[][] getResrInfo(String : vin)
+ String[] getResrTime(String : vin)
+ String[][] getOverDue()
+ String getEstimateFee(String : Vin, Date : drop)
+ String[][] getCalculateFee(String: Vin, boolean : roadStar, String : Fuel, String : Distance, boolean: Reedem, Date : Drop)
+ String[] getFeeRate(String : Vin)
+double getExtendedDistance(String : Vin, String : OdeReading)
+ String[] getEquipInfo(String : Vin)
+ int getEquipNum(String :equip, String :branch_id)
+ int[] getRedeemInfo(String: Vin)
+ String getVechileType(String: Vin)
+ int getVechilePointsEX(String : Vin)
+ String[] getStaffInfo(String : username)
Crypt
- static final String ALGORITHM = "AES"
- static final String KEY = "1Hbfh667adfDEJ78"
+ static String encrypt(String : value)
+ static String decrypt(String value)
+ static Key generateKey()

3.6 Deployment diagram

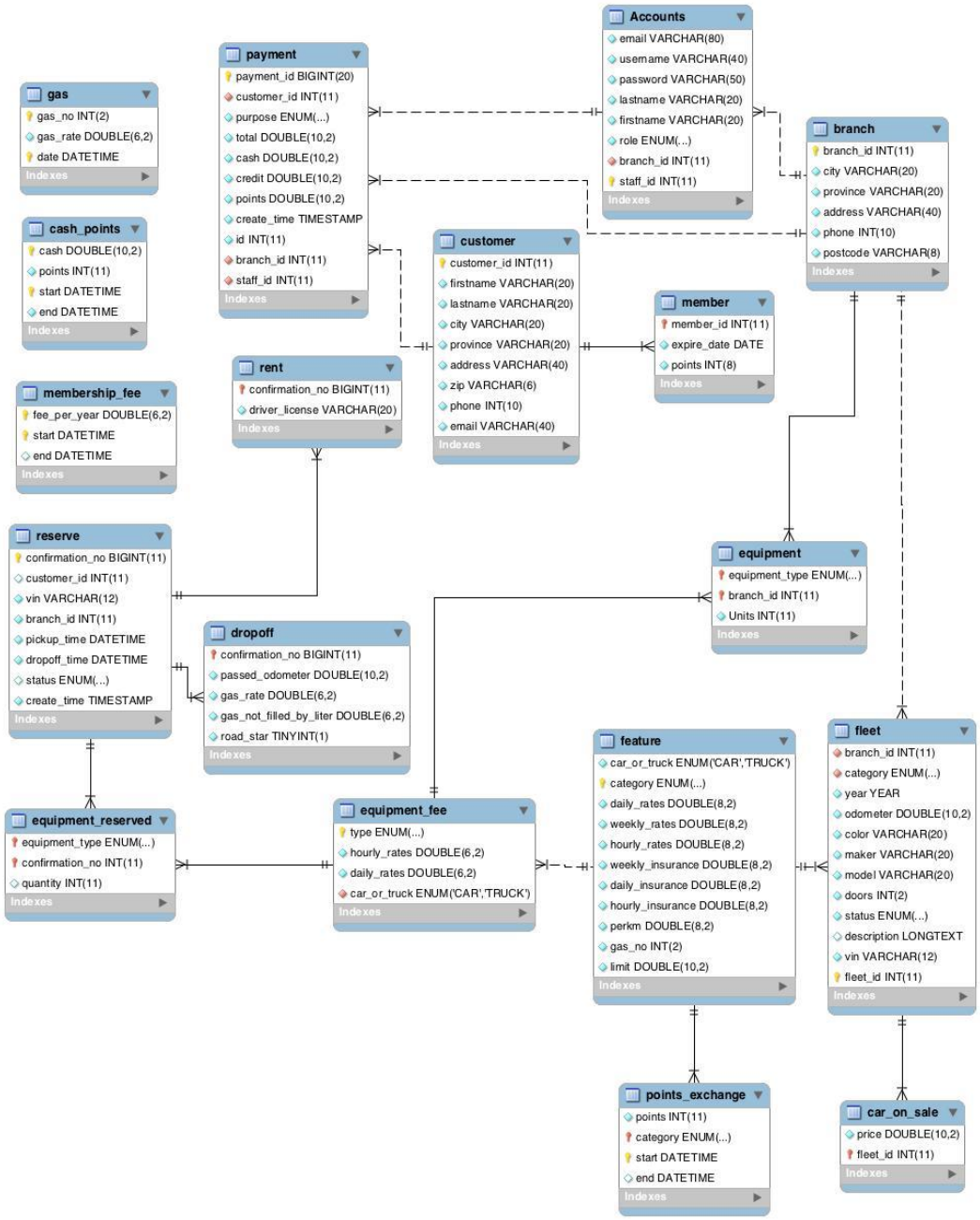


3.7 Component diagram



4 Database design

4.1 ER diagram for the project database



4.2 Additional constraints and functional dependencies

- A customer is usually identified by their phone number (Phone number of the customer should be unique)

- When a customer first joins the club, she/he gets 500 points. (default value for points is 500)

- (phone, pickupdate, dropoffdate)->confirmation_no

4.3 Set of tables

The following is a list of the tables obtained from the ERD-to-Tables process.

- accounts (staff_id, branch_id, role, firstname, lastname, username, password, email)

  Represents: entity set Accounts
  Primary key: staff_id
  Foreign Keys: branch_id
  (branch_id) references branch
  Constraints: every username must be unique
  every email must be unique

- branch (branch_id, city, province, address, postcode, phone)

  Represents: entity set branch
  Primary key: branch_id
  Constraints: every phone must be unique

- car_on_sale (price, fleet_id)

  Represents: entity set car_on_sale
  Primary key: vin
  Foreign Keys: vin
  (vin) references fleet

- cash_points(cash, points, start, end)

  Represents: entity set cash_points
  Primary key: cash, start

- Customer(customer_id, firstname, lastname, city, province, address, zip, phone, email)

  Represents: entity set customer
  Primary key: customer_id
  Constraints: every phone must be unique
  every email must be unique

- Dropoff (confirmation_no, passed_odometer, gas_not_filled_by_liter, road_star)

  Represents: relationship set Dropoff
  Primary key: confirmation_no
  Foreign Keys: confirmation_no
  (confirmation_no) references reserve

- equipment (equipment_type, branch_id, units)

  Represents: entitiy set equipment
  Primary key: equipment_type, branch_id

Foreign Keys: branch_id
(branch_id) references branch

- equipment_fee (type, hourly_rates, daily_rates, car_or_truck)

  Represents: entity set equipment_fee
  Primary key: type

- equipment_reserved (equipment_type, confirmation_no, quantity)

  Represents: relationship set equipment_reserved
  Primary key: confirmation_no, equipment_type
  Foreign Keys: confirmation_no
  (confirmation_no) references reserve

- feature (car_or_truck, category, daily_rates, weekly_rates, hourly_rates, weekly_insurance, daily_insurance, hourly_insurance, perkm, gas_no, limit)

  Represents: entity set
  feature Primary key: category
  Foreign Keys: gas_no
  (gas_no) references gas

- fleet(fleet_id, category, branch_id, year, odometer, color, maker, model, doors, status, description, vin)

  Represents: entity set fleet
  Primary key: fleet_id
  Foreign Keys: branch_id, category

  (branch_id) references branch

(category) references feature
constraints: vin must be unique

- gas(gas_no, gas_rate, date)

  Represents: entity set gas
  Primary key: gas_no, date

- member (member_id, expire_date, points)

  Represents: entity set member
  Primary key: member_id
  Foreign Keys: member_id
  (member_id) references customer

- membership_fee (fee_per_year, start, end)

  Represents: entity set membership_fee
  Primary key: fee_per_year, start

- Payment (payment_id ,customer_id, purpose, total, cash, credit, points, create_time, id, branch_id, staff_id)

  Represents: entity set Payment
  Primary key: payment_id
  Foreign Keys: customer_id, branch_id, staff_id
  (customer_id) references customer
  (branch_id) references branch
  (staff_id) references accounts

- points_exchange (points, category, start, end)

  Represents: relationship set points_exchange

Primary key: category, start
Foreign Keys: category
(category) references feature

- rent (confirmation_no, driver_license)

  Represents: relationship set rent
  Primary key: confirmation_no
  Foreign Keys: confirmation_no
  (confirmation_no) references reserve

- reserve(confirmation_no, vin, customer_id, branch_id, pickup_time, dropoff_time, status, create_time)

  Represents: relationship set reserve
  Primary key: confirmation_no
  Foreign Keys: vin, customer_id, branch_id,
  (vin) references customer
  (customer _id) references customer
  (branch _id) references branch

4.4 Table normalization

- accounts (staff_id, branch_id, role, firstname, lastname, username, password, email)

  staff_id → branch_id, role, firstname, lastname, username, password, email
  In BCNF.

- branch (branch_id, city, province, address, postcode, phone)

branch _id → city, province, address, postcode, phone

phone→ branch_id
postcode→ city
postcode→ province
city, province, address→ postcode

This table is not in BCNF. Phone , postcode, city, province, address are not keys. To make it into BCNF, we need to separate this table into several different tables. Like (branch_id, phone), (branch_id, city, province, address), (postcode, city, province, address), which makes the query not as efficient as in one table, and wastes more space. The reason why we did not use phone as the primary key is that from the management view, it is not proper to manage a branch without a id, using the phone number. The best thing is to actually combine these columns into one table in a way which does not lead to redundancies and we did this by storing the data in one table.

- car_on_sale (price, fleet_id)

  fleet_id → price

  In BCNF.

- cash_points(cash, points, start, end)

  start → points, cash, end
  In BCNF.

- Customer(customer_id, firstname, lastname, city, province, address,

zip, phone, email)

customer_id→ firstname, lastname, city, province, address, zip, phone, email

phone→ customer_id
postcode→ city
postcode→ province
city, province, address→ zip

This table is not in BCNF. Phone , postcode, city, province, address are not keys. To make it into BCNF, we need to separate this table into several different tables. Like (customer_id, phone), (customer_id, city, province, address), (postcode, city, province, zip), (customer_id, firstname, lastname, email), which makes the query not as efficient as in one table, and wastes more space, and the point is that we actually cannot guarantee that all the address data that the customers gave us are correct. The reason why we did not use phone as the primary key is that from the management view, it is not proper to manage a customer without a id, using the phone number. Although there is a The best thing is that actually combining these columns into one table does not lead to redundancies, so we decide to store those data in one table.

- Dropoff (confirmation_no, passed_odometer, gas_not_filled_by_liter, road_star)

  confirmation_no → passed_odometer, gas_not_filled_by_liter, road_star

  In BCNF.

- equipment (equipment_type, branch_id, units)

  equipment_type, branch_id → units

  In BCNF.

- equipment_fee (type, hourly_rates, daily_rates, car_or_truck)

  type → hourly_rates, daily_rates, car_or_truck
  In BCNF.

- equipment_reserved (equipment_type, confirmation_no, quantity)

  equipment_type, confirmation_no → quantity
  In BCNF.

- feature (car_or_truck, category, daily_rates, weekly_rates, hourly_rates, weekly_insurance, daily_insurance, hourly_insurance, perkm, gas_no, limit)

- category → car_or_truck, daily_rates, weekly_rates, hourly_rates, weekly_insurance, daily_insurance, hourly_insurance, perkm, gas_no, limit

  In BCNF.

- fleet(category, branch_id, year, odometer, color, maker, model,

doors, status, description, vin)

vin → category, branch_id, year, odometer, color, maker, model, doors, status, description

In BCNF.

- gas(gas_no, gas_rate, date)

  gas_no, date → gas_rate In

  BCNF.

- member (member_id, expire_date, points)

  member_id → expire_date, points

  In BCNF.

- membership_fee (fee_per_year, start, end)

  start → fee_per_year

  In BCNF.

- Payment (payment_id ,customer_id, purpose, total, cash, credit, points, create_time, id, branch_id, staff_id)

  payment_id→ customer_id, purpose, total, cash, credit, points,

create_time, id, branch_id, staff_id

In BCNF.

- points_exchange (points, category, start,

  end) category, start → points, end

  In BCNF.

- rent (confirmation_no, driver_license)

  confirmation_no → driver_license

  In BCNF.

- reserve(confirmation_no, vin, customer_id, branch_id, pickup_time, dropoff_time, status, create_time)

  confirmation_no → vin, customer_id, branch_id, pickup_time, dropoff_time, status, create_time

  In BCNF.

4.5 Final Refinement

- points_exchange
  This is used to store how the points can be exchange to the free use of vehicle is stored by period, which seems useless in this starting

phase. However, from the business management and the marketing view, the super rent company would be very likely to carry out promotions. This table is prepared for setting up a promotion and record all the promotion history in the future use.

- membership_fee
  similar use as above.

- Gas
  This table records the gas rates for everyday. For financial reports in the future, the company need to get to know the detail of a payment, gas rate is one of the important factors. If we record those in the payment table, it would waste a lot of space, considering that the super rent company would run a big business.

4.6 SQL Defination

```
-- ----------------------------
-- Table structure for `Accounts`
-- ----------------------------
DROP TABLE IF EXISTS `Accounts`;
CREATE TABLE `Accounts` (
`email` varchar(80) NOT NULL,
 `username` varchar(40) NOT NULL,
 `password` varchar(50) NOT NULL,
 `lastname` varchar(20) NOT NULL,
 `firstname` varchar(20) NOT NULL,
 `role` enum('MANAGER','CLERK') NOT NULL,
 `branch_id` int(11) NOT NULL,
 `staff_id` int(11) NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (`staff_id`),
 KEY `branch_id` (`branch_id`),
 CONSTRAINT `Accounts_ibfk_1` FOREIGN KEY (`branch_id`) REFERENCES `branch`
(`branch_id`) ON DELETE NO ACTION ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `branch`
```

```
-- ----------------------------
DROP TABLE IF EXISTS `branch`;
CREATE TABLE `branch` (
  `branch_id` int(11) NOT NULL AUTO_INCREMENT,
  `city` varchar(20) NOT NULL,
  `province` varchar(20) NOT NULL,
  `address` varchar(40) NOT NULL,
  `phone` varchar(20) NOT NULL,
  `postcode` varchar(20) NOT NULL,
  PRIMARY KEY (`branch_id`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `car_on_sale`
-- ----------------------------
DROP TABLE IF EXISTS `car_on
_sale`; CREATE TABLE `car_on_sale`
( `price` double(10,2) NOT NULL,
`vin` varchar(12) NOT NULL,
PRIMARY KEY (`vin`),
  CONSTRAINT `vin` FOREIGN KEY (`vin`) REFERENCES `fleet` (`vin`) ON DELETE
NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=COMPACT;

-- ----------------------------
-- Table structure for `cash_points`
-- ----------------------------
DROP TABLE IF EXISTS `cash_
points`; CREATE TABLE
`cash_points` ( `cash` double(10,2)
NOT NULL, `points` int(11) NOT
NULL, PRIMARY KEY (`cash`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `customer`
-- ----------------------------
DROP TABLE IF EXISTS `customer`;
CREATE TABLE `customer` (
  `customer_ id` int(11) NOT NULL AUTO_INCREMENT,
  `firstname` varchar(20) NOT NULL,
  `lastname` varchar(20) NOT NULL,
  `city` varchar(20) NOT NULL,
```

```
  `province` varchar(20) NOT NULL,
  `address` varchar(40) NOT NULL,
  `zip` varchar(6) NOT NULL, `phone`
  bigint(11) NOT NULL, `email`
  varchar(40) DEFAULT NULL,
  PRIMARY KEY (`customer_id`,`phone`)
) ENGINE=InnoDB AUTO_INCREMENT=82 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `dropoff`
-- ----------------------------
DROP TABLE IF EXISTS `dropoff`; CREATE
TABLE `dropoff` ( `confirmation_no`
bigint(11) NOT NULL,
  `passed_odometer` double(10,2) NOT NULL,
  `gas_rate` double(6,2) NOT NULL,
  `gas_not_filled_by_liter` double(6,2) NOT NULL,
  `road_star` tinyint(1) NOT NULL,
  PRIMARY KEY (`confirmation_no`),
  CONSTRAINT `confir_no` FOREIGN KEY (`confirmation_no`) REFERENCES `reserve`
(`confirmation_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `equipment`
-- ----------------------------
DROP TABLE IF EXISTS `equipment`;
CREATE TABLE `equipment` (
  `equipment _type` enum('ski_rack','child_seat','lift_gate','car_tow') NOT
  NULL, `branch_id` int(11) NOT NULL,
  `Units` int(11) NOT NULL,
  `car_or_truck` enum('CAR','TRUCK') DEFAULT NULL,
  PRIMARY KEY (`equipment_type`,`branch_id`),
  KEY `branch_id_idx` (`branch_id`),
  CONSTRAINT `branch_id` FOREIGN KEY (`branch_id`) REFERENCES
`branch` (`branch_id`) ON DELETE NO ACTION ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `equipment_fee`
-- ----------------------------
DROP TABLE IF EXISTS `equipment_fee`;
CREATE TABLE `equipment_fee` (
```

```
  `type` enum('ski_rack','child_seat','lift  _gate','car_tow') NOT
  NULL, `hourly_rates` double(6,2)  NOT NULL,
  `daily_rates` double(6,2)  NOT NULL,
  `car_or_truck` enum('CAR','TRUCK')  NOT
  NULL, PRIMARY KEY (`type`),
  KEY `car_or_truck` (`car_or_truck`),
  CONSTRAINT `equipment_fee_ibfk_1`  FOREIGN KEY (`car_or_truck`)  REFERENCES
`feature` (`car_or_truck`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
--  Table structure for `equipment_reserved`
-- ----------------------------
DROP TABLE IF EXISTS `equipment_reserved`;
CREATE TABLE `equipment_reserved`  (
`equipment_type`  varchar(20)  NOT NULL,
`confirmation_no`  bigint(11)  NOT NULL,
`quantity`  int(11)  NOT NULL,
  PRIMARY KEY (`equipment_type`,`confirmation_no`),
  KEY `confirmation_no_idx` (`confirmation_no`),
  CONSTRAINT `confirmation_ no` FOREIGN KEY (`confirmation_no`) REFERENCES
`reserve` (`confirmation_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
--  Table structure for `feature`
-- ----------------------------
DROP TABLE IF EXISTS `feature`;
CREATE TABLE `feature` (
  `car_or_truck` enum('CAR','TRUCK')  NOT NULL,
  `category` enum('Economy','Compact','Mid-size','Standard','Full-
size','Premium','Luxury','SUV','Van','24-foot','15-foot','12-foot','Box-Trucks','Cargo-
Vans') NOT NULL,
  `daily_rates` double(8,2)  NOT NULL,
  `weekly_rates` double(8,2)  NOT NULL,
  `hourly_rates` double(8,2)  NOT NULL,
  `weekly_insurance` double(8,2)  NOT NULL,
  `daily_insurance` double(8,2)  NOT NULL,
  `hourly_ insurance` double(8,2)  NOT NULL,
  `perkm` double(8,2)  NOT NULL,
  `gas_no` int(2) NOT NULL,
  `limit` double(10,2) NOT NULL,
  PRIMARY KEY (`car_or_truck`,`category`),
```

```
  KEY `gas_no` (`gas_no`),
  KEY `category` (`category`),
  CONSTRAINT `feature_ibfk_1` FOREIGN KEY (`gas_no`) REFERENCES `gas` (`gas_no`)
ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `fleet`
-- ----------------------------
DROP TABLE IF EXISTS `fleet`;
CREATE TABLE `fleet` (
  `category` enum('Economy','Compact','Mid-size','Standard','Full-
size','Premium','Luxury','SUV','Van','24-foot','15-foot','12-foot','Box-Trucks','Cargo-
Vans') NOT NULL,
  `car_or_truck`  enum('CAR','TRUCK')  NOT
  NULL, `branch_id` int(11) NOT NULL,
  `year` year(4) NOT NULL,
  `odometer` double(10,2) NOT NULL,
  `color` varchar(20) NOT NULL,
  `maker` varchar(20) NOT NULL,
  `model` varchar(20) NOT NULL,
  `doors` int(2) NOT NULL,
  `status`  enum('sale','rent','sold')  NOT
  NULL, `description` longtext,
  `vin` varchar(12) NOT NULL,
  PRIMARY KEY (`vin`),
  KEY `category` (`category`),
  KEY `car_or_truck` (`car_or_truck`),
  KEY `branch_id` (`branch_id`),
  KEY `vin` (`vin`),
  CONSTRAINT `fleet_ibfk_1` FOREIGN KEY (`category`) REFERENCES `feature`
(`category`) ON DELETE NO ACTION ON UPDATE CASCADE,
  CONSTRAINT `fleet_ibfk_2` FOREIGN KEY (`car_or_truck`) REFERENCES
`feature` (`car_or_truck`) ON DELETE NO ACTION ON UPDATE CASCADE,
  CONSTRAINT `fleet_ibfk_3` FOREIGN KEY (`branch_id`) REFERENCES
`branch` (`branch_id`) ON DELETE NO ACTION ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `gas`
-- ----------------------------
DROP TABLE IF EXISTS `gas`;
CREATE TABLE `gas` (
```

```
  `gas_no` int(2) NOT NULL,
  `gas_rate` double(6,2)  NOT NULL,
  `date` date NOT NULL,

  PRIMARY KEY (`gas_no`, `date`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
--  Table structure for `member`
-- ----------------------------
DROP TABLE IF EXISTS `member`;
CREATE TABLE `member` (
`member_id` int(11) NOT NULL,
`expire_ date` date NOT NULL,
`points` int(8) NOT NULL,
PRIMARY KEY (`member_id`),
  CONSTRAINT `member_ibfk_1`  FOREIGN KEY (`member_id`)  REFERENCES
`customer` (`customer_id`)  ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
--  Table structure for `membership_fee`
-- ----------------------------
DROP TABLE IF EXISTS `membership_fee`;
CREATE TABLE `membership _fee` (
`fee_per_ year` double(6,2) NOT NULL,
PRIMARY KEY (`fee_per_year`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
--  Table structure for `payment`
-- ----------------------------
DROP TABLE IF EXISTS `payment`;
CREATE TABLE `payment` (
`payment_id` bigint(20) NOT NULL,
`customer_id` int(11) NOT NULL,
  `purpose` enum('rent','membership','buyout')  NOT
  NULL, `total` double(10,2)  NOT NULL,
  `cash` double(10,2)  NOT NULL,
  `credit` double(10,2)  NOT NULL,
  `points` double(10,2)  NOT NULL,
  `create_time` timestamp  NOT NULL DEFAULT '0000-00-00 00:00:00' ON
UPDATE CURRENT_TIMESTAMP,
```

```
  `id` int(11) NOT NULL,
  `branch_id` int(11) NOT NULL,
  `staff_id` int(11) NOT NULL,
  PRIMARY KEY (`payment_id`),
  KEY `customer_id` (`customer_id`),
  KEY `branch_id` (`branch_id`),
  KEY `staff_id` (`staff_id`),
  CONSTRAINT `payment_ibfk_1` FOREIGN KEY (`customer_id`) REFERENCES `customer`
(`customer_id`) ON UPDATE CASCADE,
  CONSTRAINT `payment_ ibfk_2` FOREIGN KEY (`branch_id`) REFERENCES
`branch` (`branch_id`) ON UPDATE CASCADE,
  CONSTRAINT `payment_ibfk_3` FOREIGN KEY (`staff_id`) REFERENCES
`Accounts` (`staff_id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `points_exchange`
-- ----------------------------
DROP TABLE IF EXISTS `points_exchange`;
CREATE TABLE `points_exchange` (
`points` int(11) NOT NULL,
  `car_or_truck` enum('CAR','TRUCK') NOT NULL,
  `category` enum('Economy','Compact','Mid-size','Standard','Full-
size','Premium','Luxury','SUV','Van','24-foot','15-foot','12-foot','Box-Trucks','Cargo-
Vans') NOT NULL,
  PRIMARY KEY (`category`,`car_or_truck`),
  KEY `car_or_truck` (`car_or_truck`),
  KEY `category` (`category`),
  CONSTRAINT `points_exchange_ibfk_1` FOREIGN KEY (`car_or_truck`) REFERENCES
`feature` (`car_or_truck`) ON UPDATE CASCADE,
  CONSTRAINT `points_exchange_ ibfk_2` FOREIGN KEY (`category`)
REFERENCES `feature` (`category`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `rent`
-- ----------------------------
DROP TABLE IF EXISTS `rent`;
CREATE TABLE `rent` (
  `confirmation_no` bigint(11) NOT NULL,
  `driver_license` varchar(20) NOT NULL,
  PRIMARY KEY (`confirmation_no`),
  CONSTRAINT `confirm_no` FOREIGN KEY (`confirmation_no`) REFERENCES `reserve`
```

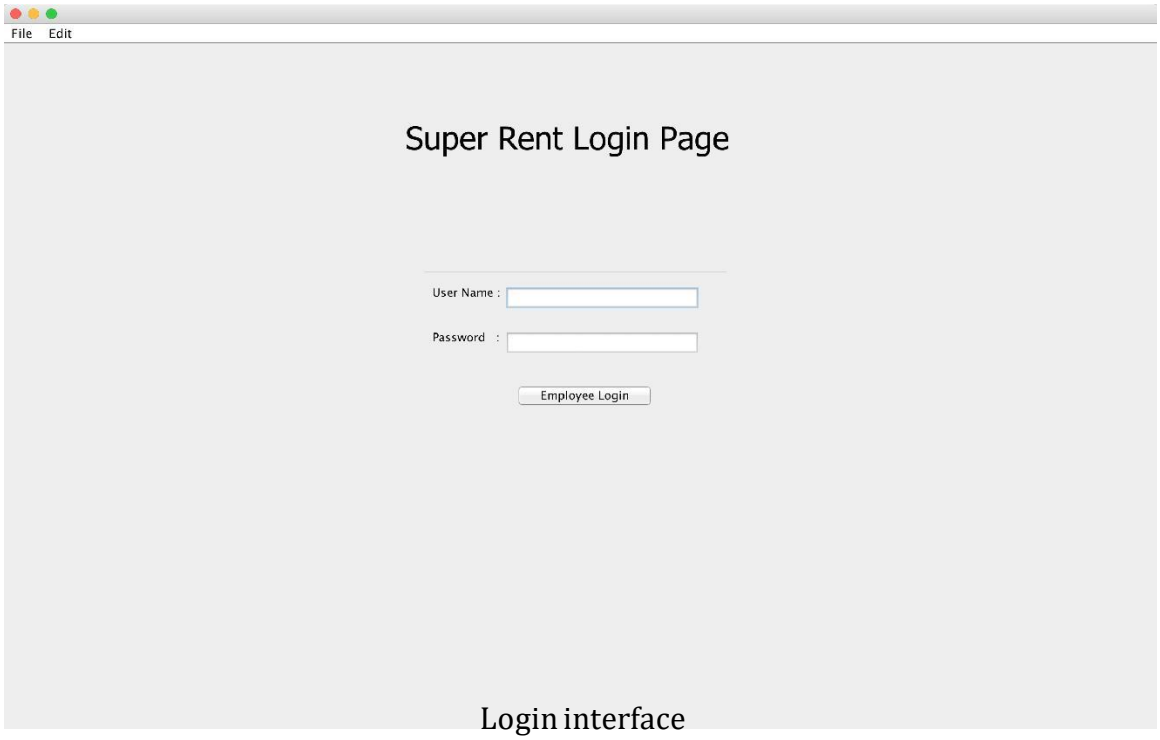
```
(`confirmation_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for `reserve`
-- ----------------------------
DROP TABLE IF EXISTS `reserve`; CREATE
TABLE `reserve` ( `confirmation_no`
bigint(11) NOT NULL, `vin` varchar(12)
NOT NULL, `customer_id` int(11)
DEFAULT NULL, `branch_id` int(11) NOT
NULL,
 `phone` bigint(20) NOT NULL,
 `pickup_time` datetime NOT NULL,
 `dropoff_time` datetime NOT NULL,
 `status` enum('reserved','rented','returned','overdue') DEFAULT NULL,
`create_time` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00' ON UPDATE
CURRENT_TIMESTAMP,
 PRIMARY KEY (`confirmation_no`),
 KEY `vin` (`vin`),
 KEY `branch_id` (`branch_id`),
 KEY `cus_id_idx` (`customer_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=COMPACT;

SET FOREIGN_KEY_CHECKS = 1;
```

5 Implementation parts

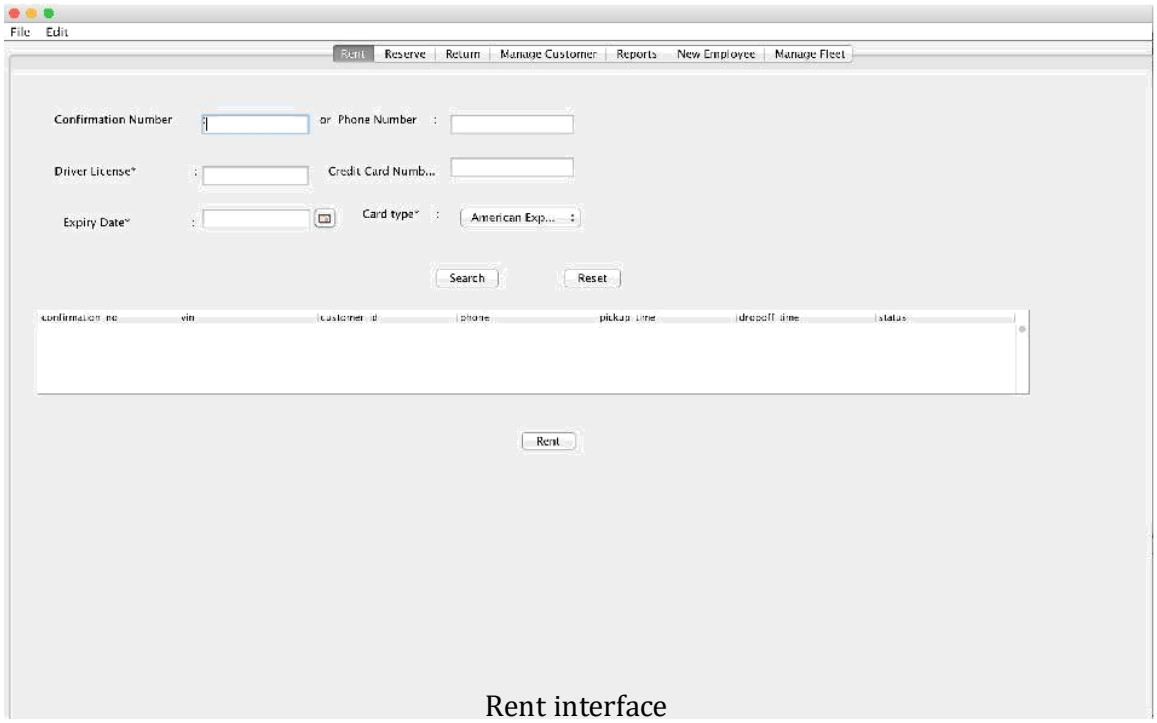
The IDE used for development is NetBeans IDE 8.0.2 (Build 201411181905), JDK 1.8 and JRE 1.8.49 for running. The application uses SWING API for graphics, mysql -connector-java driver for JDBC connection and TestNG for unit testing we executed all out unit test cases were executed for every deployment. All the developer promotions were pushed to GitHub(Configuration Management Tool) . We used maven for dependency management. The Supporting third party like Joda-time-2.4.jar for date time manipulations, mail.jar to send email about the confirmation number and reservation status, etc. JTattoo.jar and synthetic.jar for application

theme and Jasper.jar for reporting. MySql workbench for database management.

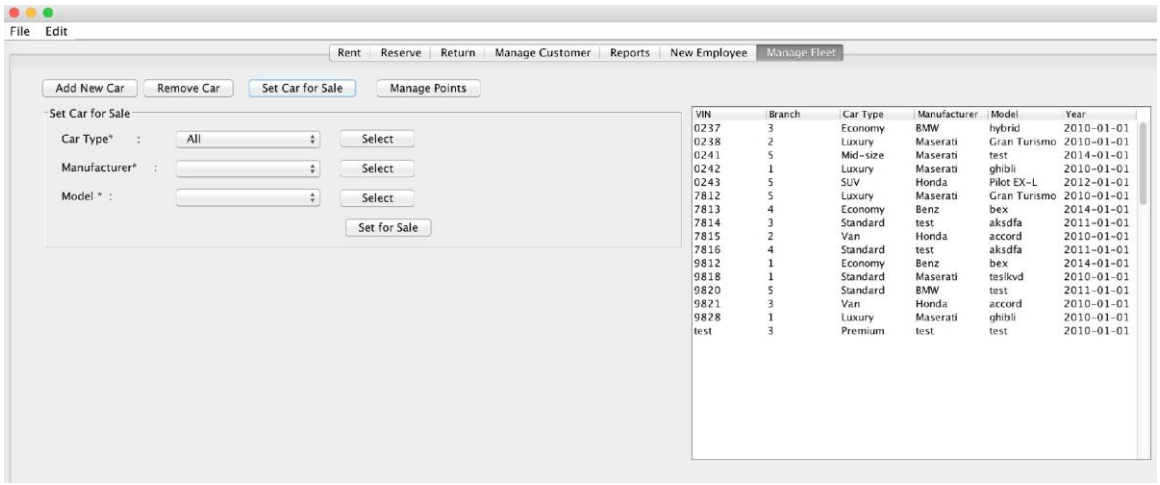
6 Performance of the system



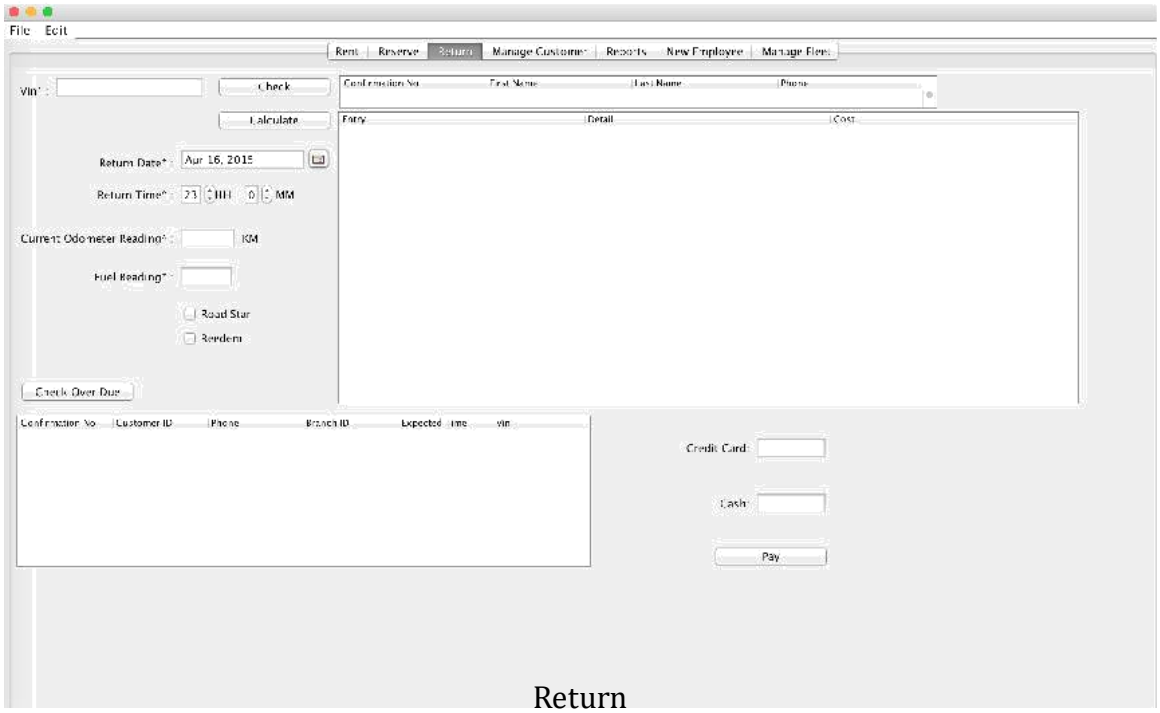
Login interface



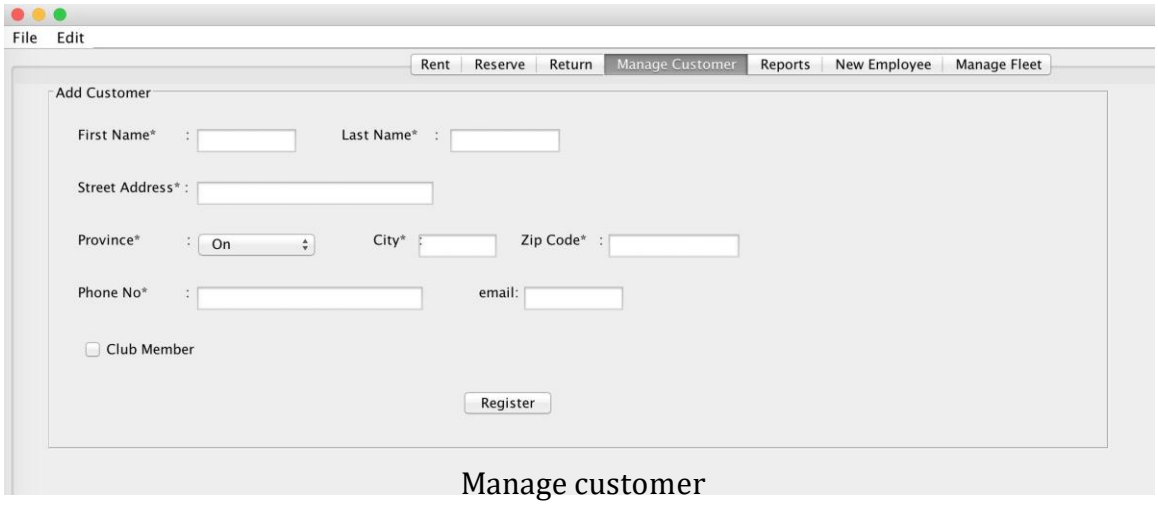
Rent interface



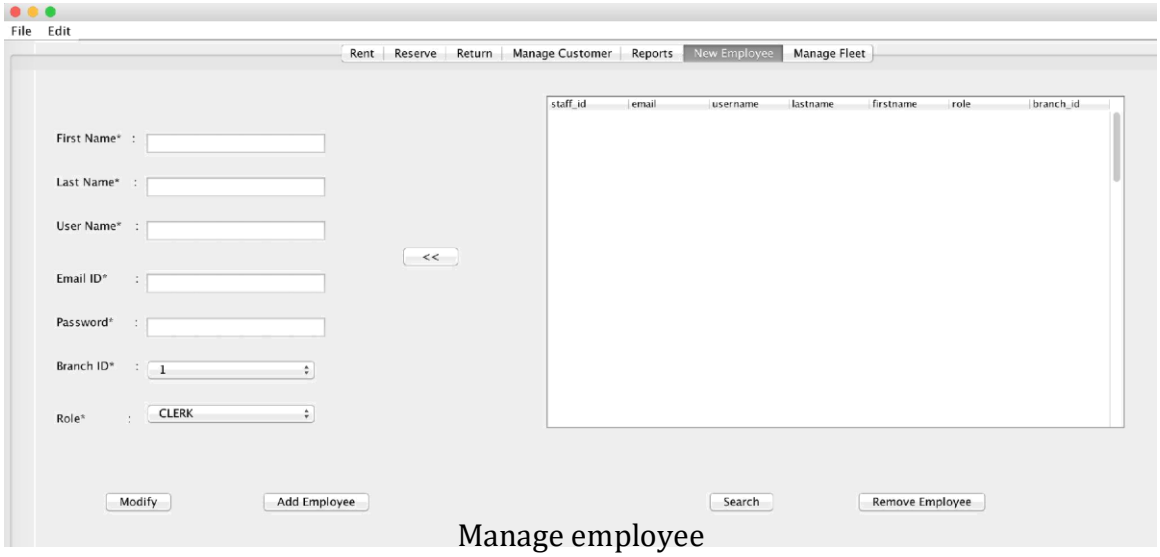
Manage fleet



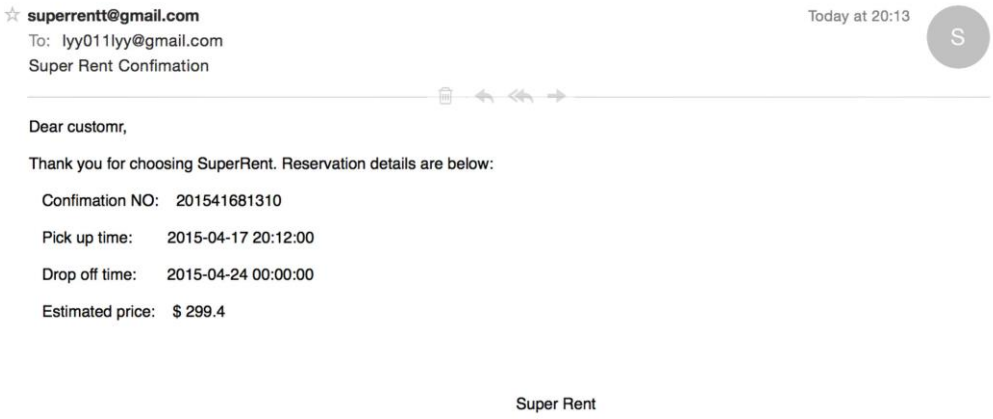
Return



Manage customer



Manage employee



Reservation confirmation email

7 Test activities

7.1 Unit test

7.1.1 TestNG

In unit test, we use TestNG to process test cases. TestNG is a testing framework for the Java programming language inspired by JUnit and NUnit. The design goal of TestNG is to cover a wider range of test categories: unit, functional, end-to-end, integration, etc., with more powerful and easy-to-use functionalities.

TestNG's main features include:

- Annotation support.
- Support for parameterized and data-driven testing (with @DataProvider and/or XML configuration).
- Support for multiple instances of the same test class (with @Factory)
- Flexible execution model. TestNG can be run either by Ant via build.xml (with or without a test suite defined), or by an IDE plugin with visual results. There isn't a TestSuite class, while test suites, groups and tests selected to run are defined and configured by XML files.
- Concurrent testing: run tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc.), and test whether the code is multithread safe.
- Embeds BeanShell for further flexibility.
- Default JDK functions for runtime and logging (no dependencies).
- Dependent methods for application server testing.[clarification needed]
- Distributed testing: allows distribution of tests on slave machines.

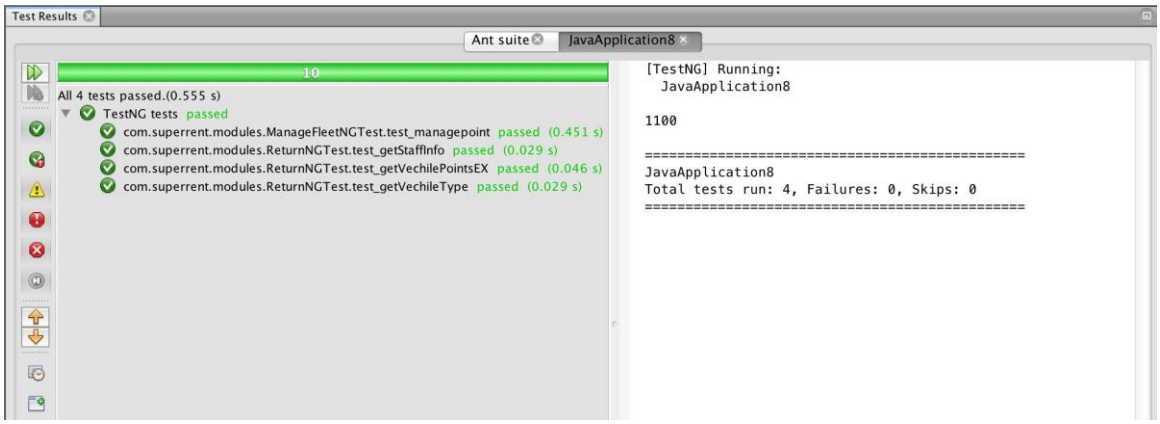
7.1.2 Examples of unit test

7.1.2.1 Test code

```
/*
 * To change this license header, choose License Headers in
Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.superrent.modules;

import java.io.IOException;
import java.sql.SQLException;
import javax.swing.JTextField;
import org.testng.Assert;
import static org.testng.Assert.*;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

/**
 *
 * @author Iniyan
 */
public class RentNGTest {
    Rent rent;
    JTextField Phone= new JTextField();
    public RentNGTest() {
    }
    @BeforeTest
    public void init(){
        rent = new Rent();
        Phone.setText("8867810252");
    }

    @Test
    public void testSomeMethod() throws SQLException,
ClassNotFoundException, IOException {

        int actual = rent.validateConfirmationNo(Phone);
        int expected = 1;
        Assert.assertEquals(actual, expected, "Data is present");
    }
```

}

7.1.2.2 Test result



7.2 System test

7.2.1 Integration test report

system testing						
NO.	module	function	test-feature	testcase	exp-result	result
1	staff management	username	no-longer than	wwwwwwwwwwwwwwwww	error message	N
2	staff management	username	in role	www	pass	Y
3	staff management	username	null		error message	Y
4	staff management	password	no-longer than	wwwwwwwwwwwwwwwww	error message	N
5	staff management	password	no-shorter than	www	error message	N
6	staff management	password	in role	wwwwwwwww	pass	Y
7	staff management	password	null		error message	Y

8	staff management	password	any specific rules	???	pass	Y
9	staff management	firstname	no numbers	fir2	error message	Y
10	staff management	firstname	no symbols	fir#	error message	Y
11	staff management	firstname	no space	fir wwe	error message	Y
12	staff management	firstname	no longer than	wwwwwwwwwwwwwwwww	error message	N
13	staff management	firstname	in rule	bobie	pass	Y
14	staff management	lastname	no numbers	fir2	error message	Y
15	staff management	lastname	no symbols	fir#	error message	Y
16	staff management	lastname	no space	fir wwe	error message	Y
17	staff management	lastname	no longer than	wwwwwwwwwwwwwwwww	error message	N
18	staff management	lastname	in rule	bobie	pass	Y
19	staff management	email	in rule	niceandsimple@example.com	pass	Y
20	staff management	email	in rule	very.common@example.com	pass	Y
21	staff management	email	in rule	a.little.lengthy.but.fine@dept.example.com	pass	Y
22	staff management	email	in rule	disposable.style.email.with+symbol@example.com	pass	Y
23	staff management	email	in rule	other.email-with-dash@example.com	pass	Y
24	staff management	email	in rule	much.more unusual@example.com	pass	Y
25	staff management	email	in rule	very.unusual.@.unusual.com@example.com	pass	Y
26	staff management	email	in rule	very.(),:;<>[]\.VERY.\"very@\\ \"very\".unusual"@strange.example.com	pass	Y
27	staff management	email	local domain name with no TLD	admin@mailserver1	error message	N
28	staff management	email	in rule	#!$%&'*+-/=?^_`{}|~@example.org	pass	Y
29	staff management	email	in rule	()<>[]:,;@\\\!#$%&'*+-/=?^_`{}|~.a"@example.org	pass	Y
30	staff management	email	space between the quotes	@example.org	error message	N
31	staff	email	in rule	üñîçøðé@example.	pass	Y

	management			com		
32	staff management	email	Unicode characters in local part	üñîçøðé@üñîçøðé.com	error message	N
33	staff management	email	an @ character must separate the local and domain parts	Abc.example.com	error message	N
34	staff management	email	only one @ is allowed outside quotation marks	A@b@c@example.com	error message	N
35	staff management	email	none of the special characters in this local part is allowed outside quotation marks	a"b(c)d,e:f;g<h>i[j\k]l@example.com	error message	N
36	staff management	email	quoted strings must be dot separated or the only element making up the local-part	just"not"right@example.com	error message	N
37	staff management	email	spaces, quotes, and backslashes may only exist when within quoted strings and preceded by a backslash	this is"not\allowed@example.com	error message	N
38	staff management	email	even if escaped (preceded by a backslash), spaces, quotes, and backslashes must still be contained by quotes	this\ still\"not\\allowed@example.com	error message	N
39	staff management	email	double dot before @	john..doe@example.com	error message	N
40	staff management	email	double dot after @	john.doe@example..com	error message	N
41	staff management	access	DBA	DBA	yes	Y
42	staff management	access	clerk	clerk	no	Y
43	staff management	access	manager	manager	yes	Y
44	staff management	reset password	in rule	wwwwwwwww	yes	Y
45	generate report	daily report	branch exists today	1	pass	Y
46	generate report	daily report	branch exists today	1	pass	Y
47	generate report	daily report	branch not exists today	111	error message	Y
48	generate report	daily report	exists branch with dates	1 Feb 12th, 2015 to Feb 28th, 2015	pass	N

49	generate report	daily report	exists branch with not dates	1 Feb 29th, 2015	error message	N
50	generate report	daily report	dates in future	1 May 1st, 2015	error message	N
51	generate report	daily report	today	today	pass	Y
52	generate report	monthly report	this month branch exists	1 April	pass	Y
53	generate report	monthly report	this month branch exists	1 April	pass	Y
54	generate report	monthly report	this month branch not exists	April	error message	Y
55	generate report	monthly report	this month	this month	pass	Y
56	generate report	monthly report	dates exist	Feb to may , 2015	pass	N
57	generate report	monthly report	dates in future	May 1st, 2015	error message	N
58	generate report	access	DBA	DBA	pass	Y
59	generate report	access	clerk	clerk	pass	Y
60	generate report	access	manager	manager	pass	Y
61	manage customer	firstname	no numbers	fir2	error message	N
62	manage customer	firstname	no symbols	fir#	error message	N
63	manage customer	firstname	no space	fir wwe	error message	N
64	manage customer	firstname	no longer than	wwwwwwwwwwwwwwww	error message	N
65	manage customer	firstname	in rule	bobie	pass	Y
66	manage customer	lastname	no numbers	fir2	error message	N
67	manage customer	lastname	no symbols	fir#	error message	N
68	manage customer	lastname	no space	fir wwe	error message	N
69	manage customer	lastname	no longer than	wwwwwwwwwwwwwwww	error message	N
70	manage customer	lastname	in rule	bobie	pass	Y
71	manage customer	email	in rule	niceandsimple@example.com	pass	Y
72	manage customer	email	in rule	very.common@example.com	pass	Y
73	manage customer	email	in rule	a.little.lengthy.but.fine@dept.example.com	pass	Y
74	manage customer	email	in rule	disposable.style.email.with+symbol@example.com	pass	Y
75	manage customer	email	in rule	other.email-with-dash@example.com	pass	Y

76	manage customer	email	in rule	much.more unusual@example.com	pass	Y
77	manage customer	email	in rule	very.unusual.@.unusual.com@example.com	pass	Y
78	manage customer	email	in rule	very.(),:;<>[]\.VERY.\"very@\\ \"very\".unusual"@strange.example.com	pass	Y
79	manage customer	email	local domain name with no TLD	admin@mailserver1	error message	N
80	manage customer	email	in rule	#!$%&'*+-/=?^_`{}|~@example.org	pass	Y
81	manage customer	email	in rule	()<>[]:,;@\\\!#$%&'*+-/=?^_`{}|~.a"@example.org	pass	Y
82	manage customer	email	space between the quotes	@example.org	error message	N
83	manage customer	email	in rule	üñîçøðé@example.com	pass	Y
84	manage customer	email	Unicode characters in local part	üñîçøðé@üñîçøðé.com	error message	N
85	manage customer	email	an @ character must separate the local and domain parts	Abc.example.com	error message	N
86	manage customer	email	only one @ is allowed outside quotation marks	A@b@c@example.com	error message	N
87	manage customer	email	none of the special characters in this local part is allowed outside quotation marks	a"b(c)d,e:f;g<h>i[j\k]l@example.com	error message	
88	manage customer	email	quoted strings must be dot separated or the only element making up the local-part	just"not"right@example.com	error message	N
89	manage customer	email	spaces, quotes, and backslashes may only exist when within quoted strings and preceded by a backslash	this is"not\allowed@example.com	error message	N

90	manage customer	email	even if escaped (preceded by a backslash), spaces, quotes, and backslashes must still be contained by quotes	this\ still\"not\\allowed @example.com	error message	N
91	manage customer	email	double dot before @	john..doe@example.com	error message	N
92	manage customer	email	double dot after @	john.doe@example..com	error message	N
93	manage customer	email	exist email	niceandsimple@example.com	error message	N
94	manage customer	address	no longer than	aaaaaaaaaaaaaaaaaaaaa	error message	N
95	manage customer	address	null		error message	Y
96	manage customer	address	in rule	aaaaaa	pass	Y
97	manage customer	city	no numbers	aaa1	error message	N
98	manage customer	city	no symbols	aaa@	error message	N
99	manage customer	city	no space	aa aa	error message	N
100	manage customer	city	no longer than	aaaaaaaaaaaaaaaaaaaaa	error message	N
101	manage customer	city	in rule	aaa	pass	N
102	manage customer	city	null		error message	Y
103	manage customer	province	no numbers	aaa1	error message	N
104	manage customer	province	no symbols	aaa@	error message	N
105	manage customer	province	no space	aa aa	error message	N
106	manage customer	province	no longer than	aaaaaaaaaaaaaaaaaaaaa	error message	N
107	manage customer	province	in rule	aaa	pass	Y
108	manage customer	province	null		error message	Y
109	manage customer	postcode	no symbols	234@34	error message	N
110	manage customer	postcode	no longer than 6 digit	22222222	error message	N
111	manage customer	postcode	no shorter than 6 digit	222	error message	N
112	manage customer	postcode	no space	w sdt4	error message	N
113	manage customer	postcode	in rule	q2s4d8	pass	Y
114	manage customer	postcode	null		error message	Y
115	manage customer	phone number	in rule	7788898213	pass	Y

116	manage customer	phone number	too long	778888888888	error message	N
117	manage customer	phone number	exists number	7788898213	error message	N
118	manage customer	phone number	too short	777	error message	N
119	manage customer	phone number	null		error message	N
120	manage customer	create a new customer			show in customer list	Y
121	manage customer	customer list	show customer list	click	pass	Y
122	manage customer	customer list	delete a customer	1	no 1 in customer	Y
123	manage customer	customer list	modify a customer	phone 7788893023	changes updated in customer list	Y
124	manage customer	customer list	search a customer by phone number exists	7788898213	show result	Y
125	manage customer	customer list	search a customer by phone number not exists	7788898214	no result	Y
126	manage customer	member list	show member list	click	pass	Y
127	manage customer	member list	delete a member	1	no 1 in member	Y
128	manage customer	member list	modify a member	phone 5586940394	changes updated in member list	Y
129	manage customer	member list	search a member by phone number exists	7788898213	show result	Y
130	manage customer	member list	search a member by phone number not exists	7788898214	no result	Y
131	manage customer	member list	enroll from customer	enroll from customer	show in memberlist	Y
132	manage customer	member list	payment for membership	amount for two year membership	expire in two years	Y
133	manage customer	member list	not enroll when create customer	not enroll when create customer	not show in memberlist	Y
134	manage customer	access	DBA	DBA	pass	Y
135	manage customer	access	clerk	clerk	pass	Y
136	manage customer	access	manager	manager	pass	Y
137	reserve	branch	exist branch	1	pass	Y
138	reserve	branch	not exist branch	111	error message	Y
139	reserve	branch	query branch through postcode	v6t	show list	Y
140	reserve	branch	null		error message	Y

141	reserve	choose type	all car exists in car category	car	pass	Y
142	reserve	choose type	all truck exists in truck category	truck	pass	Y
143	reserve	choose type	null		error message	Y
144	reserve	choose dates	dates not exist	feb 29st, 2015, april 23rd, 2015	error message	Y
145	reserve	choose dates	dates in the past	feb 1st, 2015, april 23rd, 2015	error message	Y
146	reserve	choose dates	pickup later than dropoff	May 1st, 2015, april 23rd, 2015	error message	Y
147	reserve	choose dates	In rule	april 23rd, 2015 May 1st, 2015,	pass	Y
148	reserve	choose dates	null		error message	Y
149	reserve	check equipment	all car equipment exists in car category	car	pass	Y
150	reserve	check equipment	all truck equipment exists in truck category	truck	pass	Y
151	reserve	check equipment	add equipment	baby chair for car	pass	Y
152	reserve	check equipment	add two types of equipment	baby chair 1 skate 1 for car	pass	Y
153	reserve	check equipment	add multiple equipment	baby chair 2 for car	pass	Y
154	reserve	check equipment	null		pass	y
155	reserve	reserve	search for car list	search for car list	available car list	Y
156	reserve	reserve	make reservation	make reservation	reservation list	Y
157	reserve	reserve	show estimated cost	show estimated cost	pass	Y
158	reserve	reserve	show confirmation no	show confirmation no	pass	Y
159	reserve	reserve	cancel reservation	cancel reservation	dispear from reservation list	Y
160	reserve	search	search reservation by confirmation no	111	pass	Y
161	reserve	search	search reservation by phone no and dates	7788898213 02/03/2015	pass	Y
162	reserve	search	search for not existing reservation by confirmation no	1110	error message	Y

163	reserve	search	search for not existing reservation by phone no and dates		error message	Y
164	reserve	access	DBA	DBA	no	Y
165	reserve	access	clerk	clerk	pass	Y
166	reserve	access	manager	manager	pass	Y

7.2.2 Performance test

No	Step Description	Expected Result	Transaction Name	Response Time
1	Invoke application from desktop icon. Log in with {Username} and {Password}	main menu screen is displayed	user_login	0.1s
2	Select tab reserve from menu	reserve screen is displayed	select_reserve	0.09s
3	Enter {conditions} in exact find field. Press reserve button.	item list screen is displayed.	search_by_condition	0.5s
4	press manage fleet tab	fleet screen is displayed	select_fleet	0.09s
5	Press Edit button	item is displayed in edit mode.	press_edit	0.11S
6	modify the {itemDescription} in the Description field.Press save button.	item propertied screen is displayed	modify	0.2S
7	press return tab	return screen is displayed	select return	0.09s
8	Enter {vin} in exact find field. Press reserve button.	item list screen is displayed.	search_by_condition	0.9s
9	return to step 2 and repeat.			

7.2.3 Security test

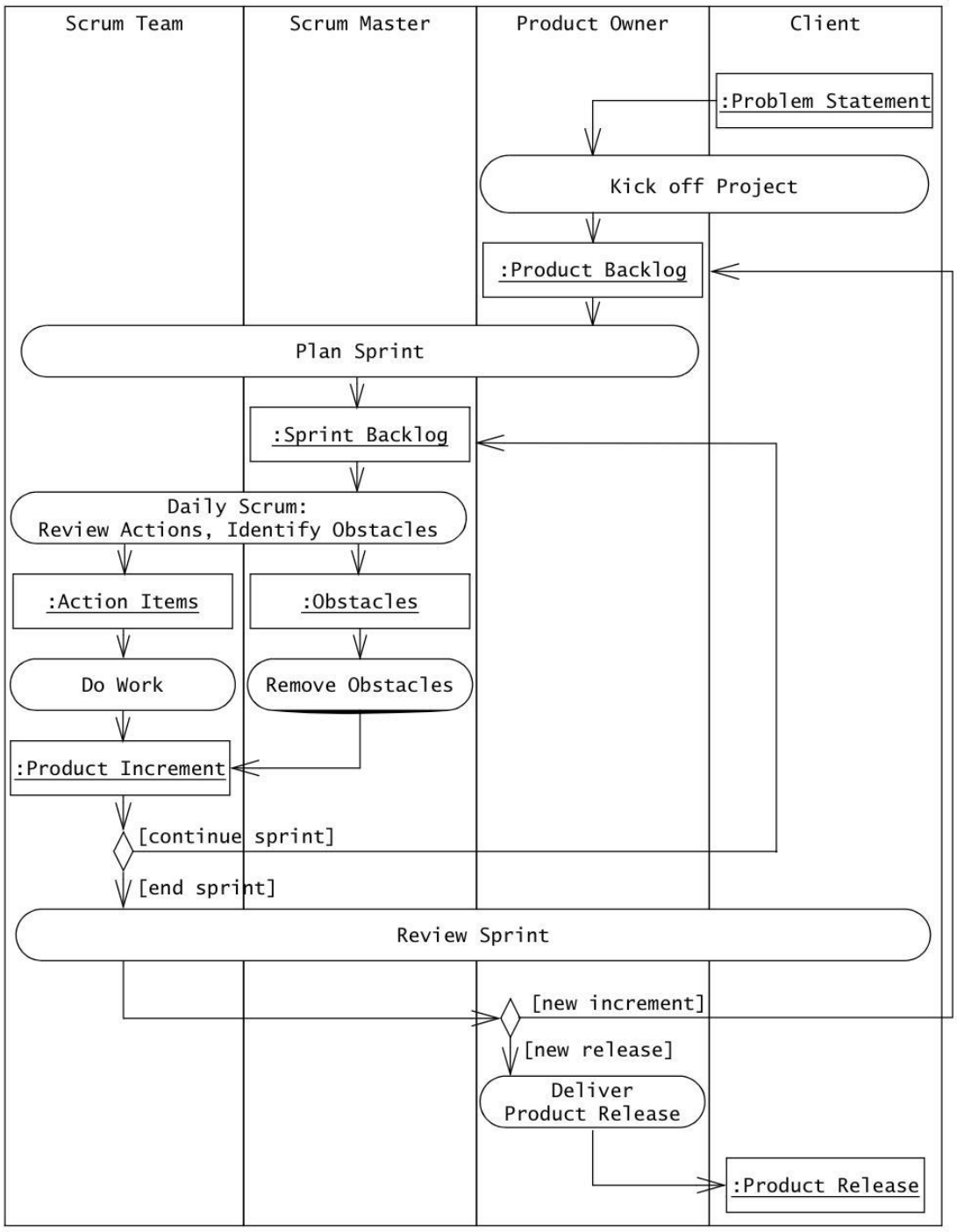
No	Description	Expected Result	result
1	login with administor account	all features are accessible	y
2	login with manager account	manage new employee is not accessible	y
3	login with clerk account	manager account less fleet and report	y
4	password check in database	encoded	y

8 Management activities

8.1 0verview

Agile is a software development philosophy encompassing rules of thumb and methodologies which enable teams to be self-organized, cross-functional, and quick to adapt to changing requirements. Of the different implementations, we felt that SCRUM was a most appropriate mindset as it does not require a formal definition of scope nor major milestones to be well defined. This means that our team is able to inquire and adapt to the requirements of more specialized functionality as they are uncovered during development. Also, the use of a backlog with SCRUM means that we could adjust the priority of story items throughout development.

8.2 SCRUM processs



Overview of a Scrum process (UML activity diagram).

8.3 Assigning management roles

8.3.1 Product owner: Dandan

- demonstrates the solution to key stakeholders who were not present in a normal iteration demo
- announces releases
- communicates team status
- organizes milestone reviews
- educates stakeholders in the development process
- negotiates priorities, scope, funding, and schedule
- ensures that the product backlog is visible, transparent, and clear

8.3.2 development team

- analyse: Yaoyao, Cindy, Inian, Mohammad, Alex, Dandan
- design: Yaoyao, Cindy, Inian, Mohammad, Alex, Dandan
- develop: Yaoyao, Cindy, Inian, Mohammad, Alex
- test: Dandan
- technical communication: Inian
- document: Dandan

8.3.3 Scrum master: Dandan

- Helping the product owner maintain the product backlog in a way that ensures the project is well defined and the team can continually advance forward on the project at any given time
- Helping the team to determine the *definition of done* for the product, with input from key stakeholders

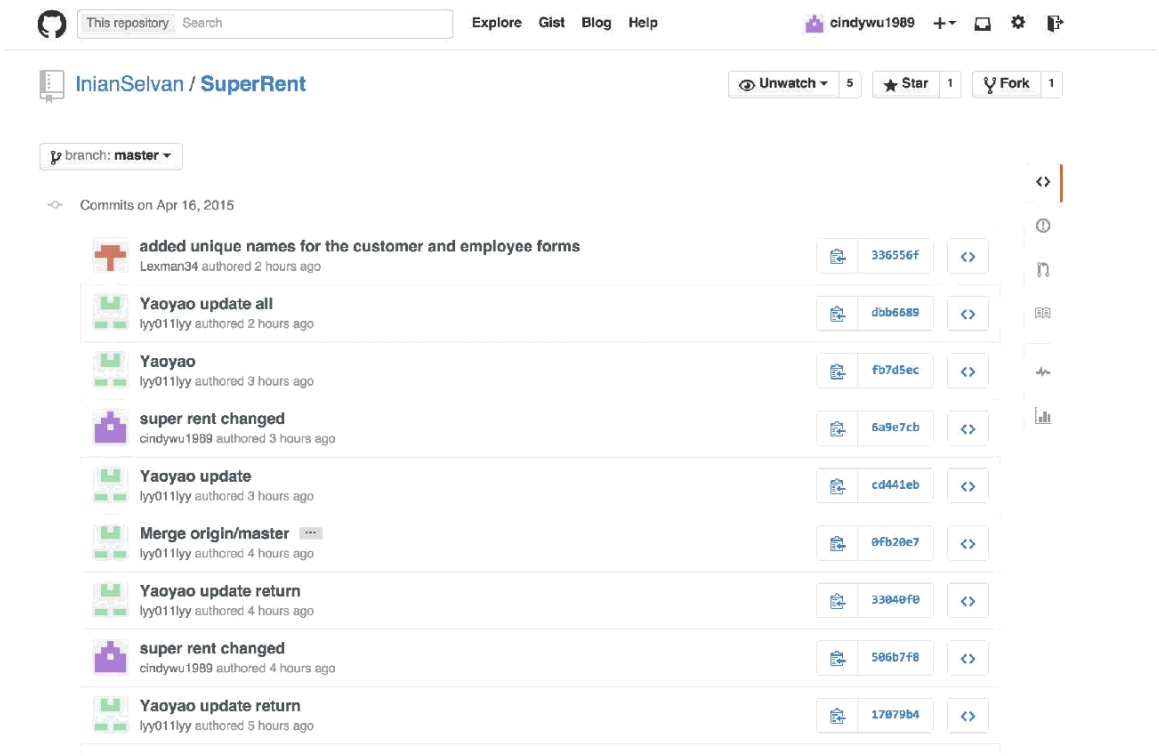
- Coaching the team, within the scrum principles, in order to deliver quality valuable features for their product
- Promoting self-organization within the team
- Helping the scrum team to avoid or remove impediments to their progress, whether internal or external to the team
- Facilitating team events to ensure regular progress
- Educating key stakeholders in the product on scrum principles

8.4 Configuration management

8.4.1 Git&Github

Our team is using Git as a versioning system and GitHub as a free service to host our Git repository. The following is an example of these tools in use to manage our project and maintain an up-to date version accessible to all members of the team.

https://github.com/InianSelvan/SuperRent


This repository Search
Explore Gist Blog Help
cindywu1989
IlanSelvan / SuperRent
Unwatch 5
Star 1
Fork 1
branch: master
Commits on Apr 16, 2015
added unique names for the customer and employee forms
Lexman34 authored 2 hours ago
336556f
Yaoyao update all
lyy011lyy authored 2 hours ago
dbb6689
Yaoyao
lyy011lyy authored 3 hours ago
fb7d5ec
super rent changed
cindywu1989 authored 3 hours ago
6a9e7cb
Yaoyao update
lyy011lyy authored 3 hours ago
cd441eb
Merge origin/master
lyy011lyy authored 4 hours ago
0fb20e7
Yaoyao update return
lyy011lyy authored 4 hours ago
33040f0
super rent changed
cindywu1989 authored 4 hours ago
506b7f8
Yaoyao update return
lyy011lyy authored 5 hours ago
17079b4

Mar 8, 2015 – Apr 16, 2015

Contributions to master, excluding merge commits

Contributions: **Commits**

InianSelvan / SuperRent

Unwatch 5 | Star 1 | Fork 1

April 9, 2015 – April 16, 2015

Period: **1 week**

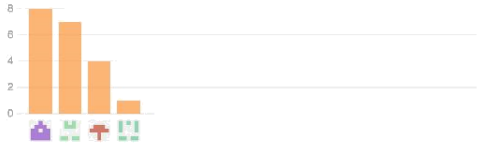
Overview

0 Active Pull Requests

0 Active Issues

0 Merged Pull Requests	**0** Proposed Pull Requests	**0** Closed Issues	**0** New Issues

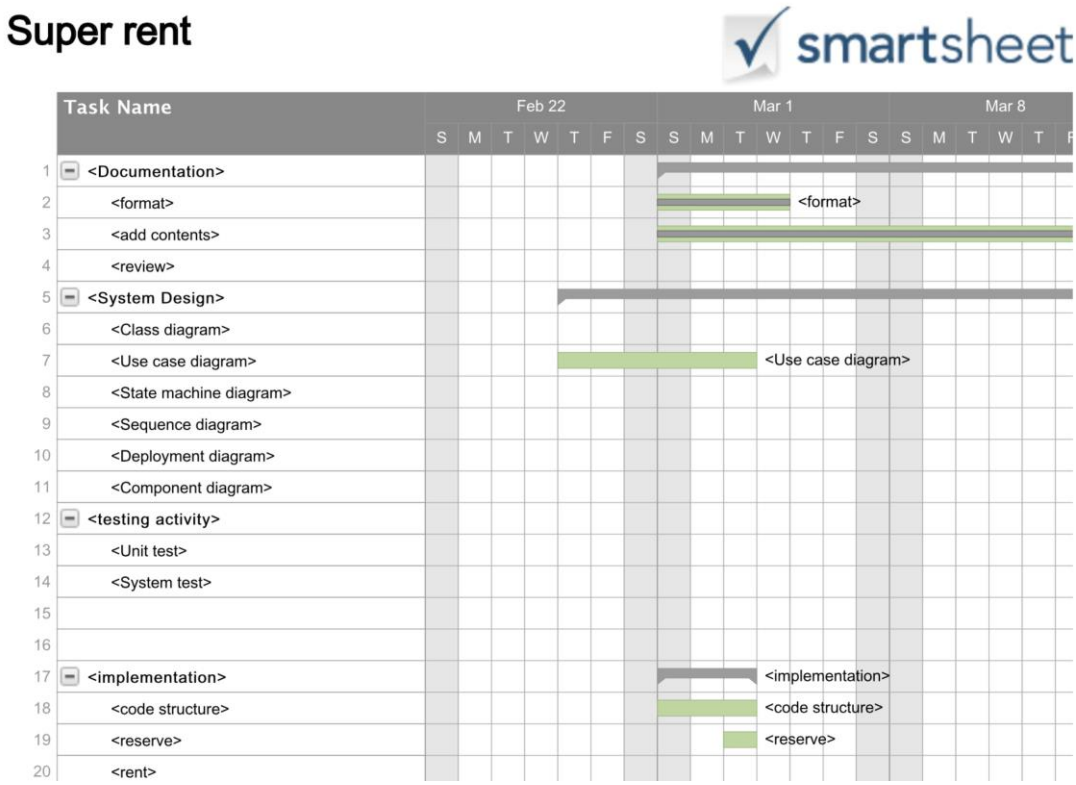
Excluding merges, **4 authors** have pushed **20 commits** to master and **20 commits** to all branches. On master, **11 files** have changed and there have been **3,356 additions** and **1,952 deletions**.



8.5 Scheduling

SmartSheet is an online collaborative scheduler, which enabled our team to allocate our time efficiently and track our progress.

https://app.smartsheet.com/b/home?lx=rN_LaPWr49MsFD7GvqpOmg



8.6 future plan

- ❖ Automated report delivery
- ❖ Automated communication with clients (overdue/return reminders)

- Dynamic fleet management
- Financial system
- Campaign settings
- Existing bug fix

9 Supporting information

9.1 background information of super rent

The system that you are to implement in this project is for a typical Car Renting Company called SuperRent or simply "the company"or "the store". The rest of this section provides a detailed description of the basic operations performed by the users of the database system.

The main function of SuperRent is to rent cars and trucks to its customers. The company keeps a number of offices in a variety of cities and locations within each city. Each location maintains a number of cars and tracks. The car types include: Economy, Compact, Mid-size, Standard, Full-size, Premium, Luxury, SUV, and Van. Track types include 24-foot, 15-foot, 12 foot, Box Trucks and Cargo Vans. Each type has different features, different daily, weekly and hourly rates, and different per-kilometer-rates (charged for kilometers driven above the limit), and different weekly. daily and hourly insurance coverage.

SuperRent maintains a list of all their customers. When a customer first rents a vehicle, the company records the customer name, address, phone number (with the area code). A customer is usually identified by their phone number. The company also maintains a list of the SuperRent Club members. To become a SuperRent Club member, a customer needs to fill in an application with their name and address and pay the annual fee determined by the company. When a customer first joins the club, she/he gets 500 points. After that, a club member gains 1 point for

every $5 she/he spends. A customer can exchange 1000 points for a one- day rent of a premium or lower -ranking car, or they can use 1500 points for a one-day rent of a luxury car, SUV, van or truck.

A customer can reserve a vehicle for specific days, can rent a vehicle, or return the vehicle that she/he has rented. To make a reservation, a customer provides the location, the type of the vehicle and the day and time for which she/he would like to pick up and return the vehicle. If there is a vehicle of the requested type available in that location, the system asks the customer for any additional options and shows an estimation of the cost. The customer can then proceed and make a reservation or cancel it. To make a reservation, the customer provides her/his name and phone number, and the system prints a confirmation number. To cancel a reservation, a customer must provide either the confirmation number or their phone number and the dates.

There are two options for "additional equipment" that is available for each car or SUV: A ski rack and a child safety seat. The additional equipment available for a track consists of a lift gate and a car-towing equipment. There is not weekly rate for the additional

equipment. The renting charges for any additional equipment is always calculated by the daily and hourly rates.

To rent a vehicle, a customer provides the same information as that required for a reservation. If a customer has already made a reservation, she/he needs only to provide the confirmation number or their phone number. The system gets the rest of the information from the reservation record. To complete the rent agreement, a customer has to provide their driver's license number and credit card information consisting of the card number and expiry date. SuperRent accepts only American Express, MasterCard and Visa.

When a customer returns a vehicle the clerk enters the the date, the time, the odometer reading and whether the gas tank is full. The system

calculates the charges by applying weekly rates to whole weeks, daily rate to remaining days and hourly rate to additional hours. It calculates the insurance cost in a similar manner taking into account the insurance rates for the vehicle. If the customer is a club member and has enough points, it uses the points to reduce the cost. In addition, if the customer is a "Road Star" (a designation given to drivers by the insurance corporation), the customer pays for insurance 50% of the regular insurance rates. To pay their bill, customers can use their credit card or cash.

SuperRent maintains a fleet of fairly new cars. Every year the managers sell a number of cars to customers or to car dealers and buy new cars. When they decide to sell a car, the car is removed from the list of the cars that can be rented and is advertised for sale. In addition to the transactions mentioned above the system must be able to generate a number of reports. At the end of each day, the company wants to produce the following reports:

- Daily Rentals: The report contains all the vehicles rented out during the day. The entries are grouped by branch, and within each branch, the entries are grouped by vehicle category. The report shows the number of vehicles rented from each category, the subtotal for each branch and the grand total for the day.
- Daily Rentals for Branch: This is the same as the Daily Rental report by it is for one specified branch
- Daily Returns: The report contains all the vehicles returned during the day. The entries are grouped by branch, and within each branch, the entries are grouped by vehicle category. The report shows the number of vehicles rented from each category and the amount of money paid by the customers, the subtotals of the number of vehicles and amount for each branch and the grand totals for the day.
- Daily Returns for Branch: This is the same as the Daily Returns report, but it is for one specified branch.

The system you design will be used by three types of user:

- customer: can make reservations, can apply for the Club membership and check the points she/he has accumulated.
- clerk: a typical clerk who processes all the customer services, like renting a vehicle, returning a vehicle, etc.
- manager: buys new vehicles and sells the old vehicles and sets all the rates and costs.

In addition to those defined above, there are a number of simple queries a clerk of the company should be able to ask. Clerks should be able to

- Show the vehicles of a specified category that are available in a given location for a given set of dates (usually given as from-date and to-date).
- Show the vehicles in a specified location and category that are overdue.
- Show the vehicles in a specified location and category that are for sale and their sale prices.

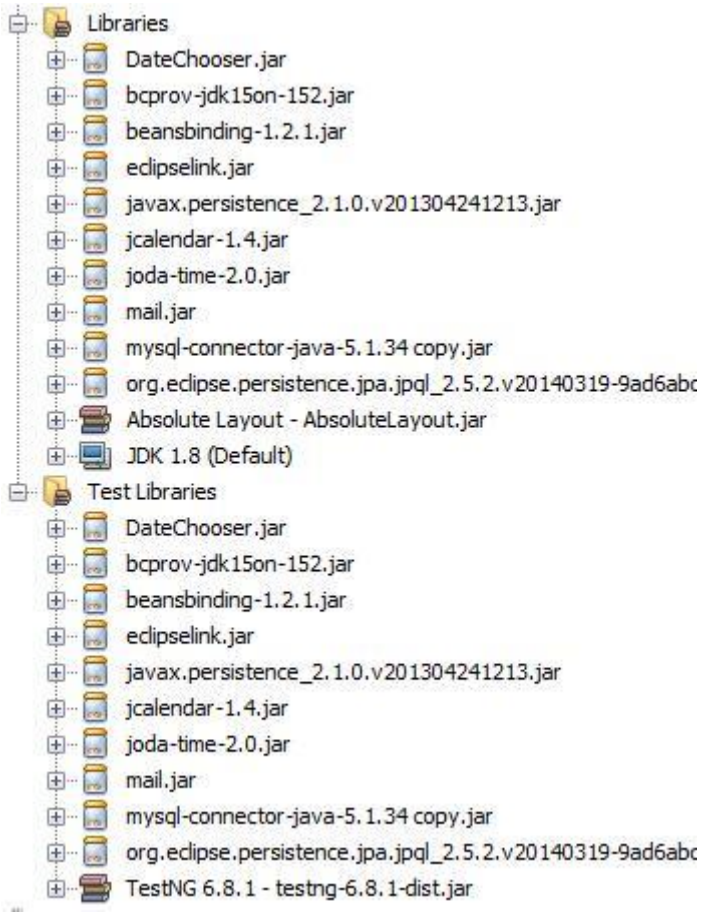
If the category is not specified, vehicles in all categories are shown, grouped by category. If the branch is left out, vehicles from all branches are shown, grouped by branch.

Finally, managers must be able to perform the following tasks:

- Show the vehicles in a specified location and category that are older than a specifies number of year. If the location or category is left out all qualifying vehicles are shown grouped by category and/or location.
- Remove a number of vehicles from the for-rent list and move them in the for-sale list.
- Sell a vehicle to a customer or dealer.
- Add more vehicles.

When a user starts the system, the program asks the user for an id and a password. Then the system starts up the appropriate menu for the current type of user. A system administrator can access all the menus and can add and remove users and change their passwords at any time.

9.2 Special Supporting libraries



9.3 Notations

- **Abstract Factory** this pattern shields a system from the concrete classes provided by a specific platform, such as a windowing style or an operating system.
- **Adapter** This pattern encapsulates a piece of legacy code that was not designed to work with the system. It also limits the impact of substituting the piece of legacy code for a different component.
- **Bridge** This pattern decouples the interface of a class from its implementation. Unlike the Adapter pattern, the developer is not constrained by an existing piece of code.
- **Command** This pattern enables the encapsulation of control objects so that user requests can be treated uniformly, independent of the specific request. This pattern protects these objects from changes required by new functionality.
- **Composite** This pattern represents a hierarchy which has dynamic width and depth. The services related to the hierarchy are factored out using inheritance, allowing a system to treat leaves and intermediate composite nodes uniformly.
- **Facade** This pattern reduces dependencies among classes by encapsulating a subsystem with a simple unified interface.
- **Observer** This pattern allows to maintain consistency across the states of one Publisher and many Subscribers.
- **Proxy** This pattern improves the performance or the security of a system by delaying expensive computations, using memory only when needed, or checking access before loading an object into memory.
- **Strategy** This pattern decouples a policy from a set of mechanisms or a context from a set of implementations. Depending on the policy or context, the mechanisms or implementations can be changed at runtime transparently from the client.
- **UML use case diagrams** Use case diagrams are used to represent the functionality of the system, as seen by an actor.
- **UML class diagrams** Class diagrams are used to represent the structure of the system, in terms of subsystems, classes, attributes, operations, and associations.
- **UML sequence diagrams** Sequence diagrams represent behavior in terms of a series of interactions among a set of objects.

- **UML statechart diagrams** Statechart diagrams represent behavior of a single object as a state machine in terms of events and transitions.
- **Activity diagrams** Activity diagrams are flow diagrams which represent behavior of a set of objects as activities and transitions.
- **UML deployment diagrams** Deployment diagrams represent the mapping of software components to hardware nodes.
- **Issue models** Issue models represent the justification of decisions in terms of issues, proposals, arguments, criteria, and resolutions.
- **PERT charts** PERT charts represent the division of work into tasks and ordering constraints.