**Vietnam National University**

**International University**



**SOFTWARE ENGINEERING MIDTERM REPORT**

TOPIC 10: FORUM

**Team name:** XINVIAPASS

**Team members:**

1. Hoàng Minh – ITITIU19029 Contribution: 100%
2. Dương Minh Lộc – ITITIU19024 Contribution: 100%
3. Nguyễn Minh Nghiệp – ITITIU19034 Contribution: 100%
4. Lương Phúc Điền – ITITIU19010 Contribution: 100%

Table of Contents

[**Task 1 – Use cases:** 3](#_Toc91776820)

[**Task 2 – Structural Domain Modeling:** 7](#_Toc91776821)

[**Task 3 – Design Class Diagram** 10](#_Toc91776822)

[**Task 4 – Atomic Use Case Specification** 11](#_Toc91776823)

[**Task 5 – Prototype and Testing the Prototype** 24](#_Toc91776824)

# **Task 1 – Use cases:**

1. Identify all the use cases, at the system level, to support the business processes described in the assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Business Process** | **Step** | **Use case** | **Remarks** |
| Renting a Car (BP1) | 1 | SubProcess Reserving a Car(BP1.1a) |  |
| Subprocess Picking-up a Car without  Reservation (BP1.1b) |  |
| 2 | Subprocess Moving Cars on Request  (BP1.2) |  |
| 3 | Picking-up a Reserved Cars (BP1.3) |  |
| 4 | Subprocess Recording the Returning  of a Car (BP1.4) |  |
| Reserving a Car  (BP1.1a) | 1 |  |  |
| 2 | UC1: Get the customer's information |  |
| 3 | UC2: Check if a car is available and  the cost |  |
| 4 | UC3: Fill out a paper form |  |
| 5 | UC4: Set a car status to "HELD" |  |
| Alter1: Release the hold on a car |  |
| 6 | UC5: Check the customer's driver license |  |
| 7 | UC6: Enter the new customer to the customer base |  |
| 8 | UC7: Create a rental and set status "RESERVED" |  |
| Picking-up a Car without Reservation (BP 1.1b) | 1 |  |  |
| 2 | Same as UC1 |  |
| 3 | Same as UC5 |  |
| 4 | UC8: Create a rental and set status to "PICKED-UP" |  |
| Alter2: UC6 |  |
| Moving a Cars on Request | 1 | UC9: Print the list of requested cars |  |
| 2 |  |  |
| Picking-up a Reserved  Car (BP1.3) | 1 | UC10: Replace a unavailable car by by another car |  |
| 2 | UC11: Arrange for a refund |  |
| 3 | UC12: Compare the driver license to the reservation one |  |
| 4 | UC13: Ask for the rest of the payment |  |
| 5 | UC14: Enter the pickup date and time |  |
| UC15: Record the car's mileage |  |
| 6 | Alter3: Cancel a reservation |  |
| Recoding the Returning of a Car (BP1.4) | 1 | Same as UC15 |  |
| 2 |  |  |
| 3 | UC 16: Change the status |  |
| Arranging for Car Maintenance (BP2) | 1 | UC 17: Print the List of Cars to be Inspected |  |
| 2 | UC18: Record that a Car needs to be Serviced |  |
| UC 19: Record that a Car is to be Removed |  |
| Adding a New Car to the Active Pool (BP3) | 1 | UC20: Enter model's details |  |
| 2 | UC21: Enter car's details |  |

1. Describe each of the use cases identified in Part (a) of Task 1 using the Event Flow Format (with Main Flow/Extensions)

* **BP1.1a:**

**Main Flow:**

1. Clerk gets the customer’s requirement.
2. Clerk checks if the appropriate car is available.
3. Clerk selects a car and set it status to “HELD”.
4. Clerk checks the customer’s driver license.
5. The new rental is created and its status is set to “RESERVED”

**Extension:**

1. No car is available:
2. Clerk fills out a paper form.
3. The deal does not go ahead as planned:
4. Clerk releases the hold on a car.
5. The customer is new:
6. Clerk will enter the new customer to the customer base.
7. The customer is blacklisted:
8. End the transaction.

* **BP1.1b:**

**Main Flow:**

1. Clerks gets the customer’s requirement.
2. Clerks check if the appropriate car is available.
3. Clerks check the customer’s driver license.
4. Clerks asks for the rest of the payment.
5. The status of the rental and car are set to “PICKED-UP”.
6. The mileage of the car is recorded.

**Extension:**

1. The customer is new:
2. Clerk will enter the new customer to the customer base.
3. The customer is blacklisted:
4. End the transaction.

* **BP1.2:**

**Main Flow:**

1. Supervisor prints out the list of cars at his/her branch.
2. Supervisor arranges for the cars to be transferred.

* **BP1.3:**

**Main Flow:**

1. Clerk checks if the reserved car is available.
2. Clerk checks the driver license.
3. Clerk asks for the rest of the payment.
4. The status of the rental and the car are set to “PICKED-UP”.
5. The car’s mileage is recorded.

**Extension:**

1. The car is not available and there is another car can be replaced:
   1. Replace the old car by the new one.
   2. Set the status of the old one to “EXCEPTIONAL” and the new one to “HELD”.
2. The substitute cannot be made:
   1. Clerk sets the status of the rental and the car to “EXCEPTIONAL”.
   2. Supervisor arranges for a refund.
3. Different driver license is used, and the new driver is not blacklisted:
   1. A change of driver can be made.
4. The new driver is blacklisted:
   1. End transaction.

* **BP1.4:**

**Main Flow:**

1. Clerk records end mileage.
2. Clerk changes the status of the rental and the car to “RETURNED”.
3. Clerk updates the car’s residing branch.

* **BP2:**

**Main Flow:**

1. Supervisor prints the list of cars to be inspected.
2. Supervisor inspects the cars.
3. Supervisor gets the cars’ status.

* **BP3:**

**Main Flow:**

1. Clerk enters details of the car.
2. Clerk changes the car’s status to “RENT-READY”.

**Extension:**

1. The model is new:
   1. Clerk enters the model’s details.
   2. Clerk specifies which group the model is classified into.

# **Task 2 – Structural Domain Modeling:**

*Step 1: we use the Text Analysis Approach to discover potential candidates*

*Graphical user interface, text, application

Description automatically generated*

*Text

Description automatically generated*

*Text

Description automatically generated*

**⇒ Through above problem statement and business description analysis, we obtained:**

**Candidate classes:** Branch, Car, Rental, Model, Group, Customer, Status, Payment, DepositPayment.  
(Potential Nouns: branch, car, rental, model, group, customer, blacklist, discount list)

*Step 2: identify classes*

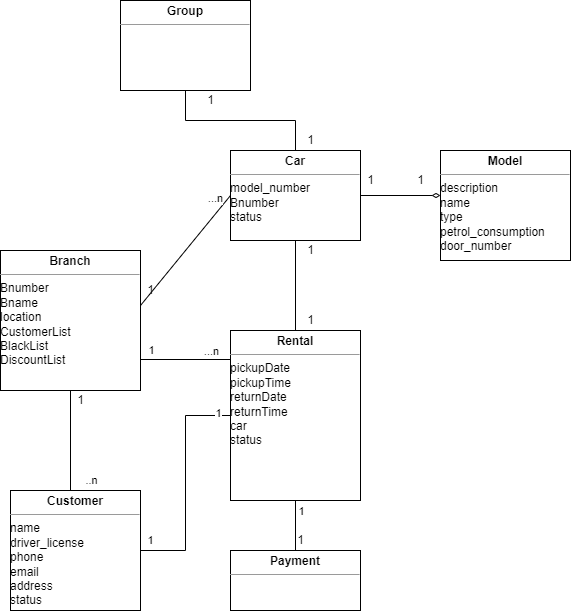
**Classes:** Branch, Car, Rental, Model, Group, Customer, RentalStatus, CarStatus, Payment, DepositPayment.

*Step 3: Identify multiplicities*

1. **Branch - Car: 1 - N**   
   A branch can have lots of car
2. **Branch - Customer: 1 - N**  
   A branch can have lots of customer
3. **Customer - Car: 1 - 1**  
   A customer can rent 1 car at a time
4. **Branch - Rental: 1 - N**  
   Branch can have and manage many rentals.
5. **Car - Model: 1 - 1**  
   A car will have a fixed model
6. **Car - Group: 1 - 1**   
   1 Car model can only belong to 1 group.
7. **Rental - Payment: 1 - 1**

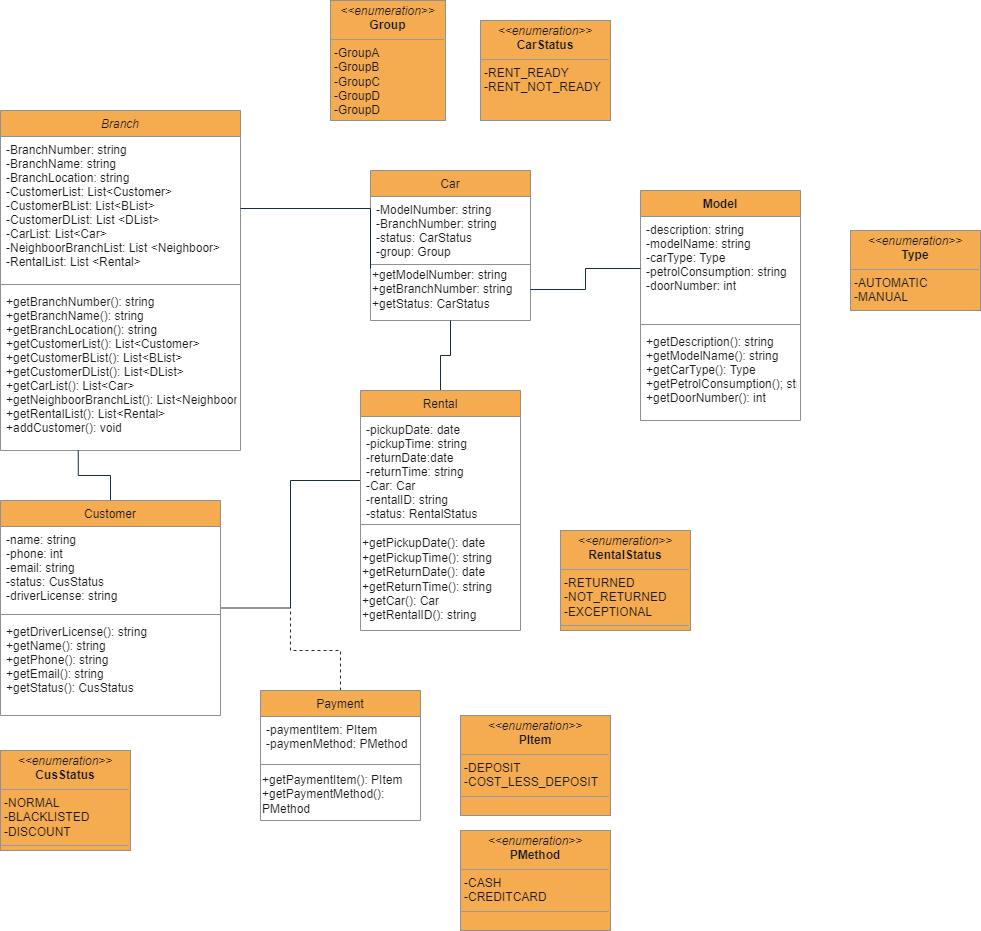
1 payment for 1 rental

*Step 4: Draw Analysis class diagram*

****

# **Task 3 – Design Class Diagram**

From the analysis class diagram from Task 2, a design class diagram can be constructed.



# **Task 4 – Atomic Use Case Specification**

* **Add a branch**

In:

branchNumber?: String

name?: String

Out:

NONE

Pre:

// branchNumber? not exists

not exists branch in branchList | branch.branchNumber = branchNumber?

Post:

// create a new branch

let b = new Branch(branchNumber?, name?) |

b.branchNumber = branchNumber?

b.name = name?

// add new branch to the list of branches

add b to branchList

* **Make a pair of branches neighbors to each other**

In:

branch1?: String

branch2?: String

Out:

NONE

Pre:

// branch1? exists

exists branch in branchList | branch.branchNumber = branch1?

// branch2? exists

exists branch in branchList | branch.branchNumber = branch2?

// check if branch 1 and the branch 2 are not already neighbor

let b1 = element in branchList | b1.branchNumber = branch1?

then

not exists branch in b1.neighborList | branch.branchNumber = branch2?

Post:

// add neighbor branch to branch’s neighborList

let b1 = element in branchList | b1.branchNumber = branch1?

then

add b1 to b2.neighborList

let b2 = element in branchList | b2.branchNumber = branch2?

then

add b2 to b1.neighborList

* **Add a car rental group**

In:

NONE

Out:

NONE

Pre:

NONE

Post:

let branch = element in branchList

then

let rental = element in branch.rentalList

then

group = rental.group

// add (group, rental) to rentalGroup of the branch

add (group, rental) to branch.rentalGroup

* **Add a model**

In:

number?: String

name?: String

transmission: Transmission

consumption?: float

numDoor?: int

group?: Group

Out:

NONE

Pre:

// modelNumber? not exists

not exists model in modelList | model.number = number?

Post:

// create a new model

let m = new Model(number?, name?, transmission?, consumption?, numDoor?, group?)|

m.number = number?

m.name = name?

m.transmission = transmission?

m.consumption = consumption?

m.numDoor = numDoor?

m.group = group?

// add new model to the list of models

add m to modelList

* **Add a car**

In:

regNum?: int

color?: String

year?: int

modelNumber?: String

branchNumber?: String

Out:

NONE

Pre:

// regNum? not exists

not exists c in carList | c.regNum = regNum?

// branchNumber? exists

exists b in branchList | b.branchNumber = branchNumber?

// modelNumber? exists

exists m in modelList | m.modelNumber = modelNumber?

Post:

// create a new car

let c = new Car(regNum?, color?, year?, modelNumber?, status?, branchNumber?) |

c.regNum = regNum?

c.color = color?

c.year = year?

c.status = status?

c.branchNumber = branchNumber?

let b = element in branchList | b.branchNumber = branchNumber?

// add new car to the list of cars

add c to carList

add c to b.carList

// add new car to model pool

let model = element in modelList | model.number = modelNumber?

then

add c to model.carList

* **Add a customer**

In:

name?: String

driverLicense?: String

email?: String

phone?: String

Out:

NONE

Pre:

// driverLicense? not exists

not exists cus in customerList | cus.driverLicense = driverLicense?

Post:

// create a new customer

let c = new Customer(name?, driverLicense?, email?, phone?) |

c.name = name?

c.driverLicense = driverLicense?

c.email = email?

c.phone = phone?

// add new customer to the list of customers

add c to customerList

* **List cars that are available at a specified branch and belong to a specified rental group (do not include the cars at neighbor branches)**

In:

group?: Group

branchNumber?: String

Out:

carList!

Pre:

// branchNumber? exists

exists b in branchList | b.branchNumber = branchNumber?

Post:

// retrieve branch

let branch = element in branchList | branch.branchNumber = branchNumber?

then

// retrieve the rentalList from group?

let rentalList = element in branch.rentalGroup.get(group) | group = group?

// add each Car object from rentalList to result list

for each r in rentalList do {

add r.getCar() to carList!

}

* **Record the return of a car**

In:

rentalNumber?: String

realReturnDate?: Date

realReturnBranchNo?: String

Out:

NONE

Pre:

// rentalNumber? exists

exists r in rentalList | r.rentalNumber = rentalNumber?

// realReturnBranchNo? exists

exists b in branchList | b.branchNumber =realReturnBranchNo?

// check if rental status is not RETURNED

exists rental in rentalList | rental.rentalNumber = rentalNumber?

rental.status != RETURNED

Post:

let r = element in rentalList |

r.rentalNumber = rentalNumber?

and r.realReturnDate = NONE

and r.status = NOT\_RETURNED

then

// update rental return date, return branch, rental’s status

set r.realReturnBranch = realReturnBranchNo?

set r.realReturnDate = realReturnDate?

set r.status = RETURNED

// update car’s status

set.r.car.status = RETURNED

if car.branchNumber = realReturnBranchNo? then

// set branch number to new branch

set car.branchNumber = realReturnBranchNo?

// retrieve old branch

let b = element in branchList | b.branchNumber = car.branchNumber

then

// remove car in old branch car list

remove car from b.carList

// retrieve the new branch

let newBranch = element in branchList | newBranch.branchNumber = realReturnBranchNo?

then

// add the car to the return branch’s carList

add car to newBranch.carList

**Addition atomic use case**

* **Add a car rental**

In:

pickupBranch?: String

returnBranch?: String

pickupDate?: Date

returnDate?: Date

year?: int

modelNumber?: String

color?: String

driverLicense?: String

Out:

NONE

Pre:

// driverLicense exists

exists cus in customerList | cus.driverLicense = driverLicense?

// check if customer is not in blacklist

not exists cus in blacklist | cus.driverLicense = driverLicense?

// check if returnDate is bigger than pickupDate

returnDate? >= pickupDate?

// model exists

exists model in modelList |

model.modelNumber = modelNumber?

model.group = group?

// check if pickupBranch and returnBranch exist

exists pickUp in branchList | pickUp.branchNumber = pickupBranch?

exists reTurn in branchList | reTurn.branchNumber = returnBranch?

// check if car with the description is available

exists car in pickup.carList |

car.modelNumber = modelNumber?

car.status = Status.READY

car.color = color?

car.year = year?

// if car is not free, search in neighbor branches

let thisBranch = element in branchList | thisBranch.branchNumber = pickupBranch?

then

let neighbor = element in branch.neighborList |

then

exists car in neighbor.carList |

car.modelNumber = modelNumber?

car.status = Status.READY

car.color = color?

car.year = year?

Post:

// retrieve customer

let customer = element in customerList | customer.driverLicense = driverLicense?

// set the status to HELD first

set car.status = HELD

// clerk asks for customer license to see if he/she is in the blacklist or discount

// if the rental is successful

// set the status to RESERVED and add new rental

set car.status = RESERVED

let rental = new Rental(customer, pickupBranch?, returnBranch?, pickupDate?, returnDate?, modelNumber?, car) |

rental.customer = customer

rental.pickupBranch = pickupBranch?

rental.returnBranch = returnBranch?

rental.pickupDate = pickupDate?

rental.returnDate = returnDate?

rental.modelNumber = modelNumber?

rental.car = car

add rental to rentalList

// add to branch rentalList and rentalGroup

add rental to pickup.rentalList

add rental to pickup.rentalGroup

* **Add a customer to blacklist**

In:

driverLicense?: String

Out:

NONE

Pre:

// check if driverLicense exists in normal customer list

exists c in customerList | c.driverLicense = driverLicense?

// check if driverLicense already existed in blacklist

exists c in blacklist | c.driverLicense = driverLicense?

Post:

// retrieve the customer

let customer = element in customerList | customer.driverLicense = driverLicense?

then

// remove customer from customerList

remove customer from customerList

// add customer to blacklist

add customer to blacklist

* **Update status of a car after being returned**

In:

regNum?: String

status?: Status

Out:

NONE

Pre:

// check if the car exists and the status is either RETURNED or READY

exists car = element carList |

car.regNum = regNum?

car.status = RETURNED || car.status = READY

// check if the input status is READY, SERVICE\_NEEDED or REMOVED

status? = READY

status? = SERVICE\_NEEDED

status? = REMOVED

Post:

// retrieve the car

let car = element in carList | car.regNum = regNum?

then

// set the status of the car

set car.status = status?

// if the input status is REMOVED then delete the car in carList, modelCarList, branchCarList

if status? = REMOVED

then

remove car from carList

remove car from car.model.carList

remove car from car.branch.carList

# **Task 5 – Prototype and Testing the Prototype**

Some of the tables are cropped due to the excessive length, the cropped parts are pasted one after another.

The source code for this lab has already been included for further examination, please refer to the code for more verbose testing.

1. **Add a branch**

Text

Description automatically generated

Text

Description automatically generated

1. **Make a pair of branches neighbors to each other**

Text

Description automatically generated

Text

Description automatically generated

**Normal case:**

Text

Description automatically generated

**Error cases:**

Text

Description automatically generated

Text

Description automatically generated

1. **Add a car rental group**

Text

Description automatically generated

Text

Description automatically generated

**Normal case:**

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

1. **Add a model**

**Text

Description automatically generated**

**Normal cases:**

Graphical user interface, text

Description automatically generated

A picture containing text

Description automatically generated

**Error case:**

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

1. **Add a car**

Text

Description automatically generated

Text

Description automatically generated

**Normal cases:**

* **Table of branches**

Text

Description automatically generated

* **Table of cars**

Graphical user interface

Description automatically generated

* **Table of models**

A screenshot of a computer

Description automatically generated with medium confidence

Rectangle

Description automatically generated

**Error cases:**

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

1. **Add a customer**

Text

Description automatically generated

Graphical user interface

Description automatically generated

1. **List cars that are available at a specified branch and belong to a specified rental group (do not include the cars at neighbor branches)**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Normal case:**

* **Table of branches**

Text

Description automatically generated

* **Table of rentals**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing diagram

Description automatically generated

Text

Description automatically generated

**Error case:**

Text

Description automatically generated with low confidence

1. **Record the return of a car**

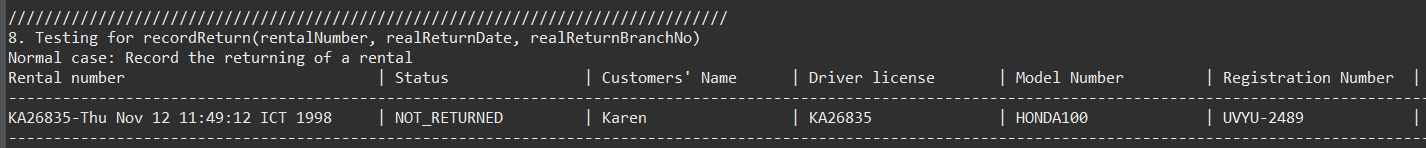
**Text

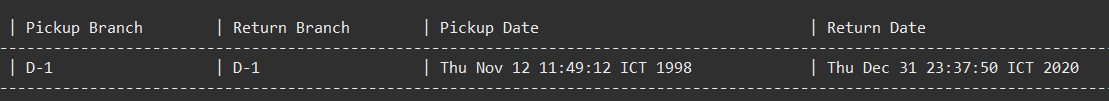
Description automatically generated**

**Text

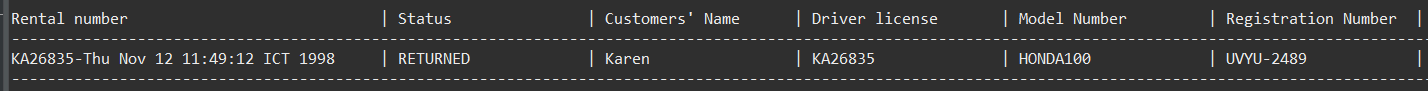
Description automatically generated**

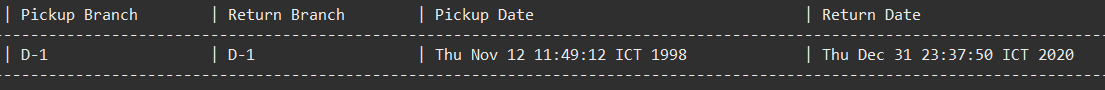
**Normal case:**

****

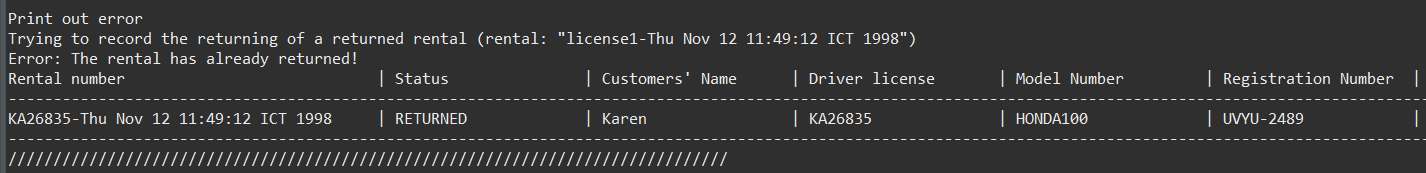
****

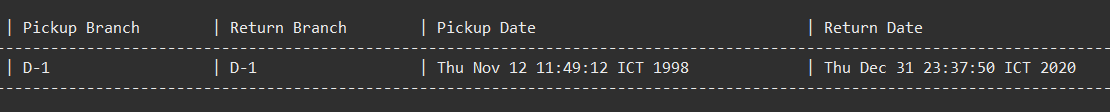
After recording the return

****

****

**Error case:**

****

****

1. **Add a rental**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Normal case:**

Graphical user interface, text, website

Description automatically generated

Text

Description automatically generated with low confidence

A picture containing graphical user interface

Description automatically generated

**Error cases:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, website

Description automatically generated

A picture containing text

Description automatically generated

1. **Add a customer to blacklist**

Text

Description automatically generated

**Normal case:**

Text

Description automatically generated

**Error case:**

Text

Description automatically generated with medium confidence

1. **Update status of a car after being returned**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Normal cases:**

Graphical user interface, text

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

**Error cases:**

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated