CS6110 - OBJECT ORIENTED ANALYSIS AND DESIGN

SMART PARKING SYSTEM USING IoT

PRIYADHARSHINI P - 2019103562

SNEHA S – 2019103583

VASUDHA E - 2019103073

Table of Contents

PROJECT AIM

A smart parking system will help us improve the efficiency of parking operations and help smoother traffic flow. The goal is to automate and decrease the time and fuel spent manually searching for the parking floor, spot and lot.

MOTIVE

The ever growing traffic congestion and uncertainty in parking availability have enforced the need for a smart parking system. This automated parking technology helps in reducing driving time, thus lowering the number of daily vehicle emissions and ultimately reducing city pollution.

ADVANTAGES

· The public can find the best spot available, saving time, resources and effort.

· The parking lot fills up efficiently and space can be utilised properly by the people enhancing the optimised parking.

· This automated parking technology helps in reducing the traffic and accident rate caused due to driving around in search of an open parking space.

INTENDED AUDIENCE

The intended audience for the case are the drivers who come to park their vehicles in the parking arena and the parking management system.

PROPOSED MINI CASE STUDY STATEMENT

The manual searching for a parking space always results in a wastage of time and fuel. This problem is predominant in the urban areas.The mini case study aims at finding the best slot for parking a vehicle within a short span of time aided with technology.

BENEFITS OF THE SYSTEM

Stakeholders (or) Beneficiaries identified :

- Driver (or) User
- Parking Management

PROPOSED CASE STUDY OVERVIEW

i)Key Modules

CODE	Key Modules	Purpose	Use Cases	Actors
A	Registration (or)	To record the details	Login, Verify	User, Server,

	Login and User Input	of the driver and to provide membership ID, To modify user profile and to choose the vehicle details (including insertion and deletion of vehicle details)	credentials, Display Login error, Update user profile, Insert new vehicle details, Delete old vehicle details, Select vehicle details	DB Server
B	Finding suitable parking spot	To find a suitable spot for parking the selected vehicle	Check free space for the vehicle, Lock a spot, Optimise route from entry, Display details of locked spot, Update occupancy details	User, Server, IoT Module, DB Server
C	Aid for car parking	To help in navigating to the slot and to help in car positioning	Show navigation to slot, Confirm correct position, Change the parked status	User, IoT Module, Server
D	Payment	To pay parking fees	Make payment, Generate Bill	User, Server, DB Server

ii)Dependencies with other modules

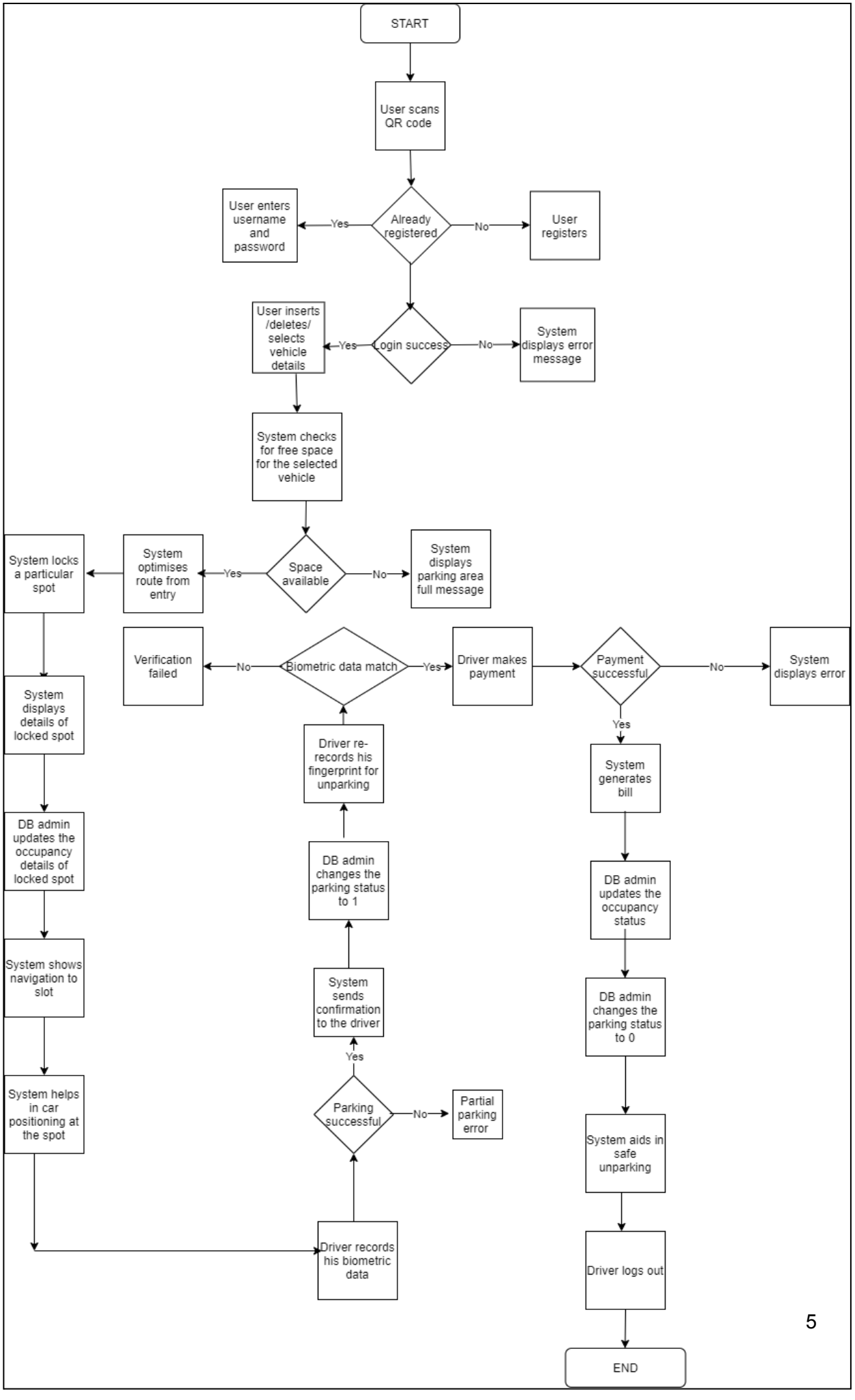
- Interface with payment processing system
    - With payment processing system for credit card payments
    - With UPI management app for UPI payments
    - With net banking portal of the listed banks for net banking payments

SCOPE

Primarily, the scope pertains to the parking arena where the vehicle has to be given a suitable parking space within a short span of time. It focuses on the parking management , the stakeholders and module which helps in computing the shortest path for the vehicle to reach the slot. The details of the vehicle owner or the driver will be populated from the login.

PROCESS FLOW

Flow Chart


START
User scans QR code
Already registered
Yes
No
User enters username and password
User registers
Login success
Yes
No
User inserts /deletes/ selects vehicle details
System displays error message
System checks for free space for the selected vehicle
Space available
Yes
No
System optimises route from entry
System displays parking area full message
System locks a particular spot
System displays details of locked spot
DB admin updates the occupancy details of locked spot
System shows navigation to slot
System helps in car positioning at the spot
Biometric data match
No
Yes
Verification failed
Driver makes payment
Payment successful
No
Yes
System displays error
System generates bill
DB admin updates the occupancy status
DB admin changes the parking status to 0
System aids in safe unparking
Driver logs out
END
Driver re-records his fingerprint for unparking
DB admin changes the parking status to 1
System sends confirmation to the driver
Parking successful
Yes
No
Partial parking error
Driver records his biometric data

PROCESS DESCRIPTION :

Key fields :

1. Membership ID
2. Name of the Driver
3. Username
4. Password
5. Vehicle Number
6. Vehicle type
7. Parking slot ID
8. Sensor ID
9. Parked status
10. Finger print of the driver
11. Date of parking
12. Parking time
13. Unparking time
14. Transaction ID
15. Payment mode
16. Invoice ID

Functional specifications :

- **Login screen :** User inputs username and password followed by authentication using DB
- Registration screen : User registers with username and password
- Vehicle Selection screen :
  - User selects vehicle details from his previous visits
  - Shows the details of vehicle parked after parking
  - Logout feature
- Update profile :
  - User can change his personal details
  - Insert new vehicle details
  - Delete old vehicle details
  - Logout feature
- Navigation screen :
  - Displays the details of slot booked
  - Shows navigation from current location till the driver parks his vehicle
  - Shows directions for unparking
- Parking confirmation screen : Displays message for successful parking or error message for partial parking
- Payment screen :
  - Shows payment methods
  - Navigates to payment processing screen depending on the mode chosen
  - Displays bill generated
- Unparking confirmation : After user re-records his finger print successfully, confirmation message sent

Dependencies with other modules :

Dependent processes

Sl.No	Process Name	Remarks
1	Payment of parking fee	Depends on the payment authorization module
2	Fingerprint recognition	Depends on the biometric device

REPORT

The report generated by the system is the bill generated after successful payment by the driver after he takes his vehicle from the parking slot.

DYNAMIC REQUIREMENTS

When the car is parked correctly at the slot, the number plate of the car will be read using the computer vision module (fixed close to the sensor). The vehicle number will be matched with the one entered by the driver. This is done to enhance security.

The first revision (1.0) of the design project is thus documented keeping in mind the specified stakeholders and their requirements.

ACTORS

ACTORS	DESCRIPTION	GOALS
User	User uses the application to find space to park his vehicle in the parking arena.	User has to find a place in the parking slot to park his vehicle.
Server	Server checks for free space and provides an optimised route to reach the free slot. It also creates a new session when the user logs in.	IoT Module has to use the sensed data provided by the sensor to monitor and signalise the state of availability of each single parking slot.
DB Server	The Database server is used for	CV Module has to capture the number plate of the

	insertion/updating/deleting/selecting records.	user's vehicle and process the captured images to verify.
Payment Processor	Payment Processor is used as an interface to perform payment transactions.	DB Server has to insert/update/delete any record.
IoT Module	Sensor senses and produces the sensed data which is used by the IoT Module for computation purposes.	Payment Processor has to process the payment that the user initiates.
CV Module	Cameras part of CV module, captures the number plate of the vehicle. The captured images are verified by the CV module with the vehicle number entered by the user.	Server has to lock a spot for the user selected vehicle and provide the optimised route to the user. It has to create a new session when the user logs in.

USE CASES

ALERT UNAUTHORISED LOGIN: If an unknown/unregistered device is logged in, then an alert message is sent by the server to the owner of the vehicle.

SEND RESPONSE: The owner of the vehicle sends a response message yes/no indicating whether to authorise login or to restrict access.

REGISTER: If the user is a new user, then he enters his credentials and registers.

UPDATE PROFILE: If the user wishes to update his profile, he selects the option whether to delete vehicle's details or to change his name and proceeds to update.

SELECT VEHICLE DETAILS: The user selects a vehicle from his previous visits or he inserts a new vehicle details.

CHECK FREE SPACE: Once the user selects a vehicle, the server checks for free space in the parking arena for that vehicle.

LOCK A SPOT: If free space is available, the server locks a spot.

OPTIMISE ROUTE: The server computes the optimised route to reach that locked spot.

UPDATE OCCUPANCY DETAILS: The occupancy details for that locked slot is updated in the database.

SHOW NAVIGATION: The server shows the route to the user to reach the locked spot.

AID VEHICLE POSITIONING: The IoT Module helps the user in vehicle positioning at the locked spot.

RECORD FINGERPRINT: The user records his fingerprint at the parking slot.

CAPTURE VEHICLE NUMBER: The CV Module captures the number of the vehicle at the parking slot.

CHANGE PARKED STATUS: The parked status of that locked spot is updated.

ALERT TIME EXCEEDED: The server sends an alert message to the user if he has exceeded the maximum parking time limit.

MAKE PAYMENT: The user makes payment.

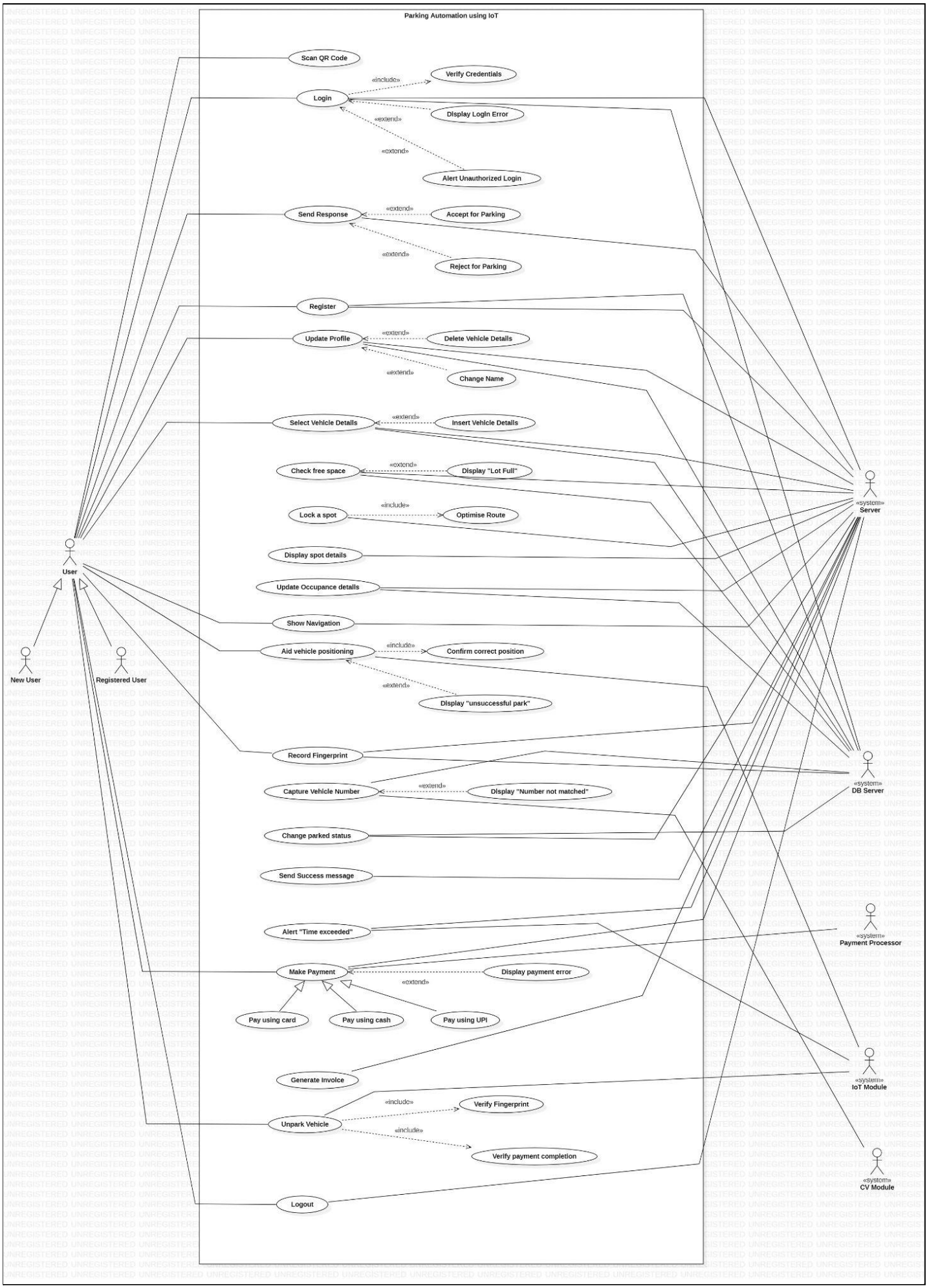
GENERATE BILL: The server generates the bill by calculating the amount for the number of hours the user has parked his vehicle at the parking slot.

UNPARK: The user unparks his vehicle.

VERIFY FINGERPRINT: The user's fingerprint is verified before he unparks his vehicle.

VERIFY PAYMENT COMPLETION: The server verifies whether the user has completed his payment transaction successfully.

USE CASE DIAGRAM


Parking Automation using IoT
Scan QR Code
Verify Credentials
«include»
Login
Display LogIn Error
«extend»
«extend»
Alert Unauthorized Login
Send Response
«extend»
Accept for Parking
«extend»
Reject for Parking
Register
Update Profile
«extend»
Delete Vehicle Details
«extend»
Change Name
Select Vehicle Details
«extend»
Insert Vehicle Details
Check free space
«extend»
Display "Lot Full"
Lock a spot
«include»
Optimise Route
Display spot details
Update Occupance details
Show Navigation
Aid vehicle positioning
«include»
Confirm correct position
«extend»
Display "unsuccessful park"
Record Fingerprint
Capture Vehicle Number
«extend»
Display "Number not matched"
Change parked status
Send Success message
Alert "Time exceeded"
Make Payment
Display payment error
«extend»
Pay using card
Pay using cash
Pay using UPI
Generate Invoice
Unpark Vehicle
«include»
Verify Fingerprint
«include»
Verify payment completion
Logout
User
New User
Registered User
«system»
Server
«system»
DB Server
«system»
Payment Processor
«system»
IoT Module
«system»
CV Module

USE CASE DESCRIPTION :

1.ALERT UNAUTHORISED LOGIN

Scope: System under consideration

Level: User goal

Primary Actors: Server,User, DB Server

Stakeholders and Interests: The user is notified when a person tries to log in using an unknown device

Preconditions: The user(owner) should have been registered already.

Success Guarantee(or Postconditions): On logging in from an unknown/unregistered device, an alert message is sent to the owner of the vehicle.

2.SEND RESPONSE

Scope: System under consideration

Level: User goal

Primary Actors: User(owner), Server

Stakeholders and Interests: The owner sends a response for acceptance (yes/no).

Preconditions: A new device must be used to login in.

Success Guarantee(or Postconditions): If the owner sends yes,the vehicle can be parked, else the vehicle should not be allowed to park.

3.REGISTER

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The user wishes to register to the application.

Preconditions: The user should scan the QR code.

Success Guarantee(or Postconditions): The user successfully registers to the application.

4.UPDATE PROFILE

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The user wishes to insert or delete his vehicle details or change his name.

Preconditions: The user should have logged into the application.

Success Guarantee(or Postconditions): The user successfully updates his profile.

5.SELECT VEHICLE DETAILS

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The user wishes to select his vehicle to park.

Preconditions: The user should log in into the application.

Success Guarantee(or Postconditions): The user selects his vehicle.

6.CHECK FREE SPACE

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The server checks for a free space for the vehicle selected by the user.

Preconditions: The user should select a vehicle to park.

Success Guarantee(or Postconditions): The free spaces are filtered.

Extensions(or Alternate Scenarios): If space is unavailable, the system displays “Lot full” message.

7.LOCK A SPOT

Scope: System under consideration

Level: User goal

Primary Actors: Server

Stakeholders and Interests: The server locks a parking slot for the vehicle selected by the user.

Preconditions: The server should filter the free spaces for the vehicle.

Success Guarantee(or Postconditions): If space is available, the server locks a spot for the vehicle.

8.OPTIMISE ROUTE

Scope: System under consideration

Level: User goal

Primary Actors: User, Server

Stakeholders and Interests: The server optimizes the route for the locked spot from the user’s point of entry.

Preconditions: The server should lock a spot for the vehicle.

Success Guarantee(or Postconditions): The server provides an optimized route to the user to reach the locked spot.

9.UPDATE OCCUPANCY DETAILS

Scope: System under consideration

Level: User goal

Primary Actors: Server, DB Server

Stakeholders and Interests: The DB is updated with the occupancy status of the locked spot.

Preconditions: The server should lock a spot for the spot's occupancy details to be updated in the database.

Success Guarantee(or Postconditions): The database is updated with the occupancy details of the locked spot.

10.SHOW NAVIGATION

Scope: System under consideration

Level: User goal

Primary Actors: User, Server

Stakeholders and Interests: The system wishes to show the navigation to the user to reach the locked spot.

Preconditions: The server should have provided an optimised route.

Success Guarantee(or Postconditions): The system shows navigation to the user to reach the locked spot.

11.AID CAR POSITIONING

Scope: System under consideration

Level: User goal

Primary Actor: IOT module, User, Server

Stakeholders and Interests: User- Wishes to park his vehicle at the locked spot. IOT module – Aids in car positioning at that spot.

Preconditions: The system should show the navigation to the user to reach the locked spot.

Success Guarantee(or Postconditions): On success, the IOT module confirms the correct position.

Extensions(or Alternate scenario): If the user is not able to park his vehicle at the locked spot, then the system displays "unsuccessful parking" error.

12.RECORD FINGERPRINT

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: User: Records his fingerprint.

Preconditions: The User should park his vehicle.

Success Guarantee(or Postconditions): The fingerprint of the user is recorded.

13.CAPTURE VEHICLE NUMBER

Scope: System under consideration

Level: User goal

Primary Actors: CV Module, Server, DB Server

Stakeholders and Interests: CV module: Captures the vehicle number of the parked vehicle.

Preconditions: The user should park his vehicle and record his fingerprint.

Success Guarantee(or Postconditions): The vehicle number of the parked vehicle is captured.

Extensions(or Alternate scenario): If the vehicle number is not present in the database, then an “unidentified vehicle” error is displayed.

14.CHANGE PARKED STATUS

Scope: System under consideration

Level: User goal

Primary Actors: Server, DB Server

Stakeholders and Interests: The DB Server changes the parked status of the slot.

Preconditions: The user should park his vehicle and should complete the fingerprint registration process.

Success Guarantee(or Postconditions): On success, the database is updated for the parked status of the slot to 1 and confirmation message is sent to the user.

15.ALERT TIME EXCEEDED

Scope: System under consideration

Level: User goal

Primary Actors: User, Server

Stakeholders and Interests: Server: Alerts the user “time exceeded”,once he crosses the allotted time limit.

Preconditions: The user should park his vehicle and the system should allot a maximum time limit for that vehicle.

Success Guarantee(or Postconditions): “Time exceeded” message is sent to the user.

16.MAKE PAYMENT

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, Payment Processor

Stakeholders and Interests: The user makes the payment. The Payment Processor is used to authorise the payment process.

Preconditions: The user should successfully park his vehicle. The user should choose the mode of payment.

Success Guarantee(or Postconditions): Payment is done successfully by the user.

Extensions(or Alternate scenarios): If the payment fails, “Payment Error” message is displayed to the user.

17.GENERATE BILL

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The system generates the bill.

Preconditions: The user should make the payment.

Success Guarantee(or Postconditions): The bill is generated by the system.

18.UNPARK VEHICLE

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: User – Wishes to unpark his vehicle.

Preconditions: The system should verify the user's fingerprint and also verify whether the user has completed the payment process.

Success Guarantee(or Postconditions): The user unparked his vehicle successfully. The DB Server changes the parked status of that slot to 0.

19.VERIFY FINGERPRINT

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The server verifies the fingerprint with the fingerprints entered in the database.

Preconditions: The user should park his vehicle and should complete fingerprint registration.The user should register his fingerprint again while unparking.

Success Guarantee(or Postconditions): On successful verification,the user is allowed to unpark his vehicle if he has completed the payment process as well.

Extensions(or Alternate scenarios): The user is not allowed to unpark his vehicle.

20.VERIFY PAYMENT COMPLETION

Scope: System under consideration

Level: User goal

Primary Actors: User, Server, DB Server

Stakeholders and Interests: The system checks whether the user has paid.

Preconditions: The user should come to unpark his vehicle.

Success Guarantee(or Postconditions): On successful verification, the user is allowed to unpark his vehicle.

Extensions(or Alternate scenarios): The user is not allowed to unpark his vehicle.

INCLUDE AND EXTEND USE CASES DESCRIPTION:

- **Verify credentials** – It is included in the login use case. Once the user enters his login credentials, it should be verified by the server.

- **Display Login error** – It is extended from the login use case. If the user enters invalid login credentials, then the server displays login error.

- **Alert Unauthorised Login** – It is extended from the login use case. If an unknown/unregistered device is logged in, an alert message is sent to the owner of the vehicle. In this case, login is extended to alert unauthorised login use case.

- **Accept for parking** – It is extended from send response use case. If the user sends yes, the send response is extended to accept for parking use case.

- **Reject for parking** – It is extended from the send response use case. If the user sends no, the send response is extended to reject for the parking use case.

- **Delete vehicle details** – If the user wishes to update profile and chooses delete vehicle details option, then update profile use case is extended to delete vehicle details use case.

- **Change name** – If the user wishes to update the profile and chooses the change name option, then the update profile use case is extended to the change name use case.

- **Insert vehicle details** – If the user's vehicle is not in his vehicle list, then he wishes to insert a new vehicle. In this case, select vehicle details is extended to insert vehicle details use case.

- **Display lot full** – It is extended from check free space use case. The server checks for free space for the selected vehicle. If there is no free space, 'lot full message' is displayed.

- **Optimise route** – It is included in lock a spot use case. Once the server locks a spot for the selected vehicle, the server also provides the optimized route to reach that spot.

- **Confirm correct position** – It is included in aid vehicle positioning use case. The IoT Module confirms the correct position of the vehicle parked in the parking slot.

- **Display unsuccessful park** – It is extended from aid vehicle positioning use case. If the user has not parked correctly, an unsuccessful parking message is displayed.

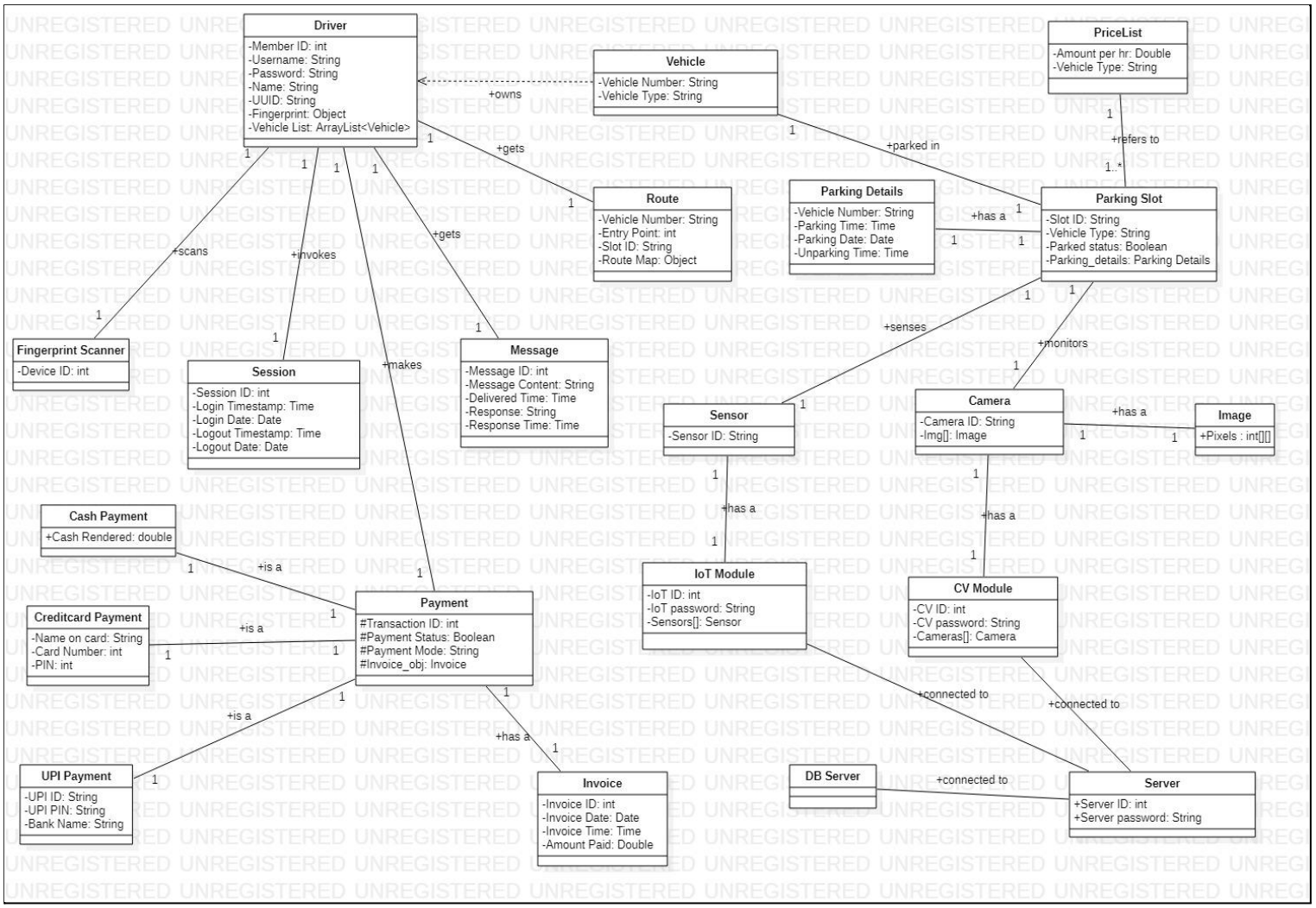
- **Display number not matched** – It is extended from capture vehicle number use case. If the captured vehicle number doesn't match with the user entered vehicle number, then the capture vehicle number use case is extended to display 'number not matched'.

- **Display payment error** – It is extended from ‘make payment’ use case. If the transaction fails, ‘make payment’ use case is extended to display payment error.

- **Verify fingerprint** – It is included in the unpark vehicle use case. The user’s fingerprint is verified.

- **Verify payment completion** – It is included in unpark vehicle use case. The user’s payment transaction is verified.

DOMAIN MODEL



DATA DICTIONARY:

1. Driver : Driver is an actor who brings his vehicle to park in the parking arena. He is a beneficiary of the system where he is aided in the parking process by the IoT module. This class is also a generalised superclass. This class stores the details of the driver when he registers along with the unique ID of the device which he uses to register.
Collaborations : Vehicle, Session, Payment, Message, Route, Fingerprint scanner, Register, Login_logout, Profile_update

2. Vehicle : Vehicle is a class that is used by the driver which has attributes such as vehicle number, type and the owner ID. Depending on the type of the vehicle, a particular slot is blocked by the system where the driver can park his vehicle.
Collaborations : Driver, Parking Slot

3. Parking Slot : Parking slot is a class which describes each parking slot in the entire parking arena. It tells about the type of vehicle that can be parked and whether the slot is free or empty.
Collaborations : Vehicle, Sensor, Parking Details, Price List, Camera

4. Parking Details : It can be considered as a composition in Parking Slot whose details will be updated when a driver parks his vehicle successfully in the parking slot provided to him. Vehicle number, parking date and time are noted for the particular slot when the driver parks his vehicle. Unparking time is noted when he takes his vehicle off the slot.
Collaborations : Parking Slot

5. Camera : Camera is a class which should be instantiated for every Parking slot class and each camera is identified by its unique ID. It helps to capture the number plate of the vehicle when the driver parks his vehicle, which is in turn checked with that entered by the driver.
It forms a part of the CV Module.
Collaborations : CV Module, Image

6. Fingerprint scanner : Fingerprint scanner class helps in capturing the fingerprint of the fingerprint of the driver which is then matched again when he takes his vehicle .Each slot has a fingerprint scanner.
Collaborations : Driver

7. Session : Session is a class which is used to store the login details of the driver such as his unique ID and login timestamp. It also records the logout timestamp of the driver when he takes his vehicle and moves out from the parking arena.
Collaborations : Driver, Server

8. Payment : This class is a superclass which records the transaction process made by the driver to pay the parking fee. The details of the invoice generated are also noted.
Collaborations : Driver, Invoice

9. Sensor : Sensor is a class which must be instantiated for every parking slot as every parking slot has a single sensor to determine whether a car has been parked depending on the threshold distance. It is composed in the IoT module which computes whether the car has been parked successfully.
Collaborations : Parking Slot, IoT Module

10. Price List : This class gives the parking fee for different types of vehicle for a time duration of one hour. The class helps in generating the invoice.
Collaborations : Parking Slot

11. Route : Route is a class which will provide the optimised route for the driver to navigate from his entry to the allotted parking slot. The route for each driver is identified by the unique ID of the driver.
Collaborations : Driver

12. Message : Message class is used to provide all the confirmation, alert and error messages where each message is routed to the appropriate driver with the help of his unique ID. Messages are provided as strings with appropriate message IDs. The delivered time will be recorded. This class also contains an attribute which holds the response given by the user.
Collaborations : Driver

13. IoT module : IoT Module is an actor which computes the data from Sensor class. Its aim is to aid the driver in parking the vehicle properly.
Collaborations : Sensor

14. CV module : CV Module is an actor which takes the captured data from the Camera class to compute the vehicle number using a computer vision algorithm.
Collaborations : Camera

15. Image : Image is a class which is used to store the pictures captured by the Camera class in the form of pixels
Collaborations : Camera

16. CreditCard Payment : This is a specialised class inherited from Payment class to make credit card payments. This is a class which implements the Payment Processor interface.
Collaborations : Payment, Payment Processor

17. Cash Payment : This is a specialised class inherited from Payment class to make cash payments.
Collaborations : Payment

18. UPI Payment : This is a specialised class inherited from Payment class to make credit card payments. This is a class which implements the Payment Processor interface.
Collaborations : Payment, Payment Processor

19. Invoice : This class is used as a composition class inside Payment class to record the details of the invoice for the transaction.
Collaborations : Payment

20. Server : Sever is a class that is used to manage the overall operations of the application.
Collaborations : CV Module, IoT Module, DB Server

21. DB Server : DB Server is a class that is used to manage the Database operations of the applications
Collaborations : Server

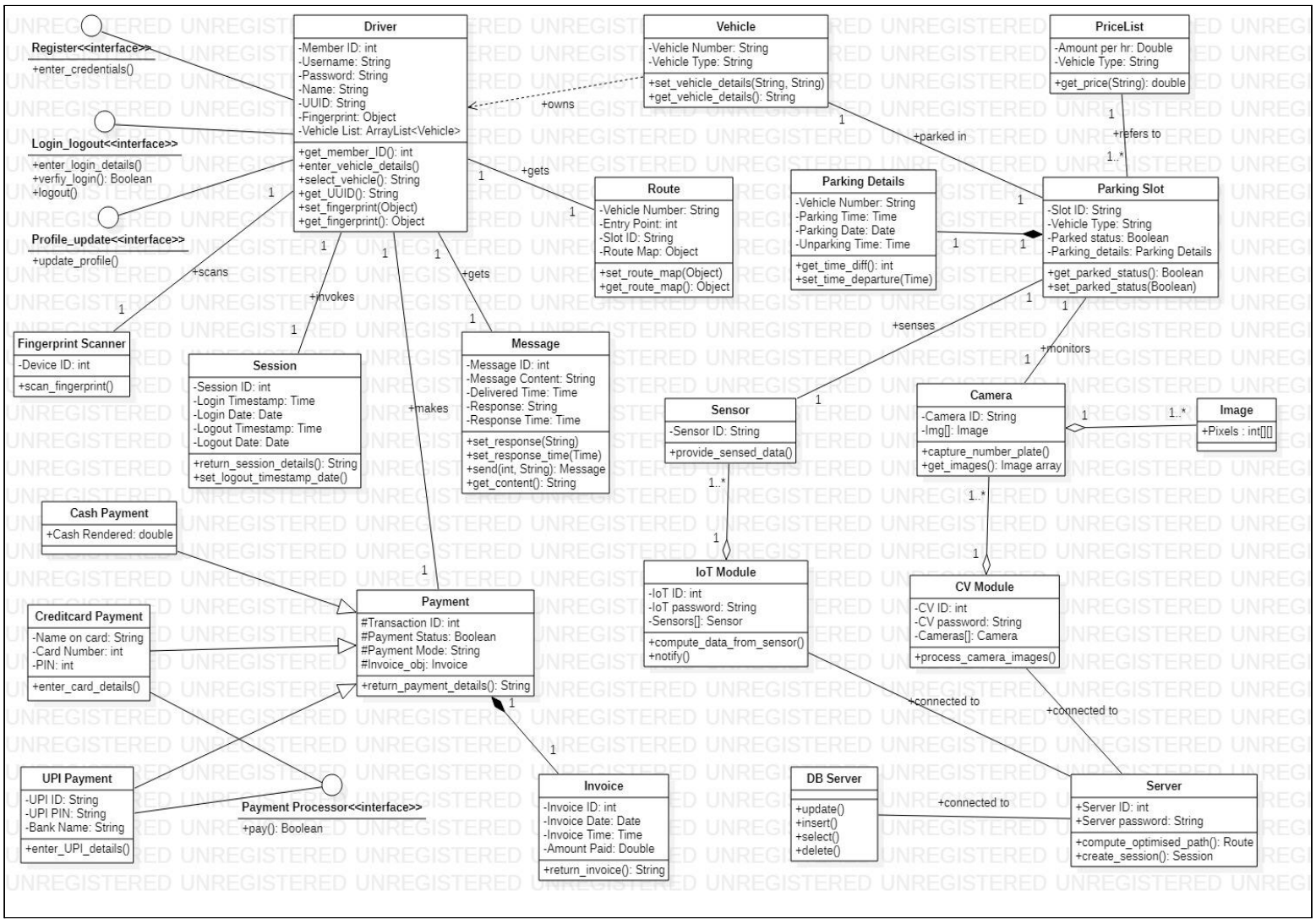
22. Register : Register is an interface which provides a method for the drivers to register themselves along with their vehicles.
Collaborations : Driver

23. Login_logout : This is an interface which provides methods to login, logout and to validate the login credentials.
Collaborations : Driver

24. Profile_Update : This interface provides a method for the drivers to update their profile.
Collaborations : Driver

25. Payment_Processor : This interface is implemented by the CreditCardPayment, UPIPayment classes to process the payment which the driver makes.
Collaborations : CreditCardPayment, UPIPayment

CLASS DIAGRAM



CRC (Class Responsibility Collaborator)

CLASS	ATTRIBUTES	METHODS	INTERACTIONS
Driver	- Member ID - Username - Password - Name - UUID - Fingerprint - Vehicle_list	- get_member_ID() : int - enter_vehicle_details() - select_vehicle_details() : String - park() - unpark() - get_fingerprint() : Object - set_fingerprint(Object)	Register Login_logout Profile_update Fingerprint Scanner Vehicle Session Route Payment Message

Vehicle	- Vehicle Number - Vehicle Type	- set_vehicle_details(String, String) - get_vehicle_details() :String	Driver Parking Slot
Price List	- Amount per hr - Vehicle Type	- get_price(String) : double	Parking Slot
Session	- Session ID - Login Timestamp - Login Date - Logout Timestamp - Logout Date	- return_session_details() :String -set_logout_timestamp_date()	Driver Server
Parking Slot	- Slot ID - Vehicle Type - Parked status	- get_parked_status() :Boolean - set_parked_status(Boolean)	Vehicle Sensor Camera Parking Details PriceList
Parking Details	- Vehicle Number - Parking Time - Parking Date - Unparking Time	- get_time_diff(String, int) : int - set_time_departure(Time)	Parking Slot
Fingerprint Scanner	- Device ID	- scan_fingerprint()	Driver
Route	- Vehicle Number - Entry Point - Slot ID - Route Map	- set_route_map(Object) - get_route_map() : Object	Driver
Message	- Message ID - Message Content - Delivered Time - Response - Response Time	- set_response(String) - set_response_time(Time) - send(int) : Message - get_content() : String	Driver
Image	- Pixels		Camera
Camera	- Camera ID - Img[]	- capture_number_plate() - get_images(): Image[]	Parking Slot CV Module Image
Sensor	- Sensor ID	- provide_sensed_data()	Parking Slot IoT Module
CV Module	- CV ID - CV Password -Cameras[]	- process_camera_images()	Camera Server

IoT Module	- IoT ID - IoT Password - Sensors[]	-compute_data_from_sensor() - notify()	Sensor Server
Payment	- Transaction ID - Payment Status - Payment Mode - Invoice Obj	- return_payment_details() : String	Driver Invoice
CreditCard Payment	- Name on Card - Card Number - PIN	- enter_card_details()	Payment Payment Processor<<interface>>
Cash Payment	- Cash Rendered		Payment
UPI Payment	- UPI ID - UPI PIN - Bank Name	- enter_UPI_details()	Payment Payment Processor<<interface>>
Invoice	- Invoice ID - Invoice Date - Invoice Time - Amount paid	- return_invoice() : String	Payment
Server	- Server ID - Server Password	- compute_optimised_path() : Route - create_session() : Session	IoT Module CV Module DB Server
DB Server		- update() - insert() - select() - delete()	Server
Register <<interface>>		- enter_credentials()	Driver
Login_logout <<interface>>		- enter_login_details() - verify_login() : Boolean - logout()	Driver
Profile_Update <<interface>>		- update_profile()	Driver
Payment Processor <<interface>>		–pay(): Boolean	CreditCard Payment UPI Payment

LIST OF CLASSES :

Descriptive Classes :

1. Cash Payment
2. Image

Abstract Class :

3. Payment (super)

Interfaces :

4. Register
5. Login_logout
6. Profile_Update
7. Payment Processor

Singleton classes:

8. Server
9. DB Server
10. CV Module
11. IoT Module

Other classes :

12. Driver
13. Vehicle
14. Price List
15. Parking Slot
16. Parking Details
17. Camera
18. Sensor
19. Route
20. Message
21. Invoice
22. Session
23. Fingerprint Scanner

Subclasses and Superclasses (Generalisation) :

For the Super Class Payment :
The subclasses are :

24. Credit Card Payment
25. Cash Payment
26. UPI Payment

<u>THE LIST OF RELATIONS :</u>

Aggregation :

- Image - Camera (Many-to-One)
- Camera - CV Module (Many-to-One)
- Sensor - IOT Module (Many-to-One)

Composition :

- Parking Details - Parking Slot(One-to-One)
- Invoice - Payment (One-to-One)

Dependency :

- Vehicle - Driver (Many-to-One)

Inheritance :

- Payment - Credit Card Payment, Cash Payment, UPI Payment

Interface realisation :

- Register - Driver
- Login_Logout - Driver
- Profile_Update - Driver
- Payment Processor - Server

Association :

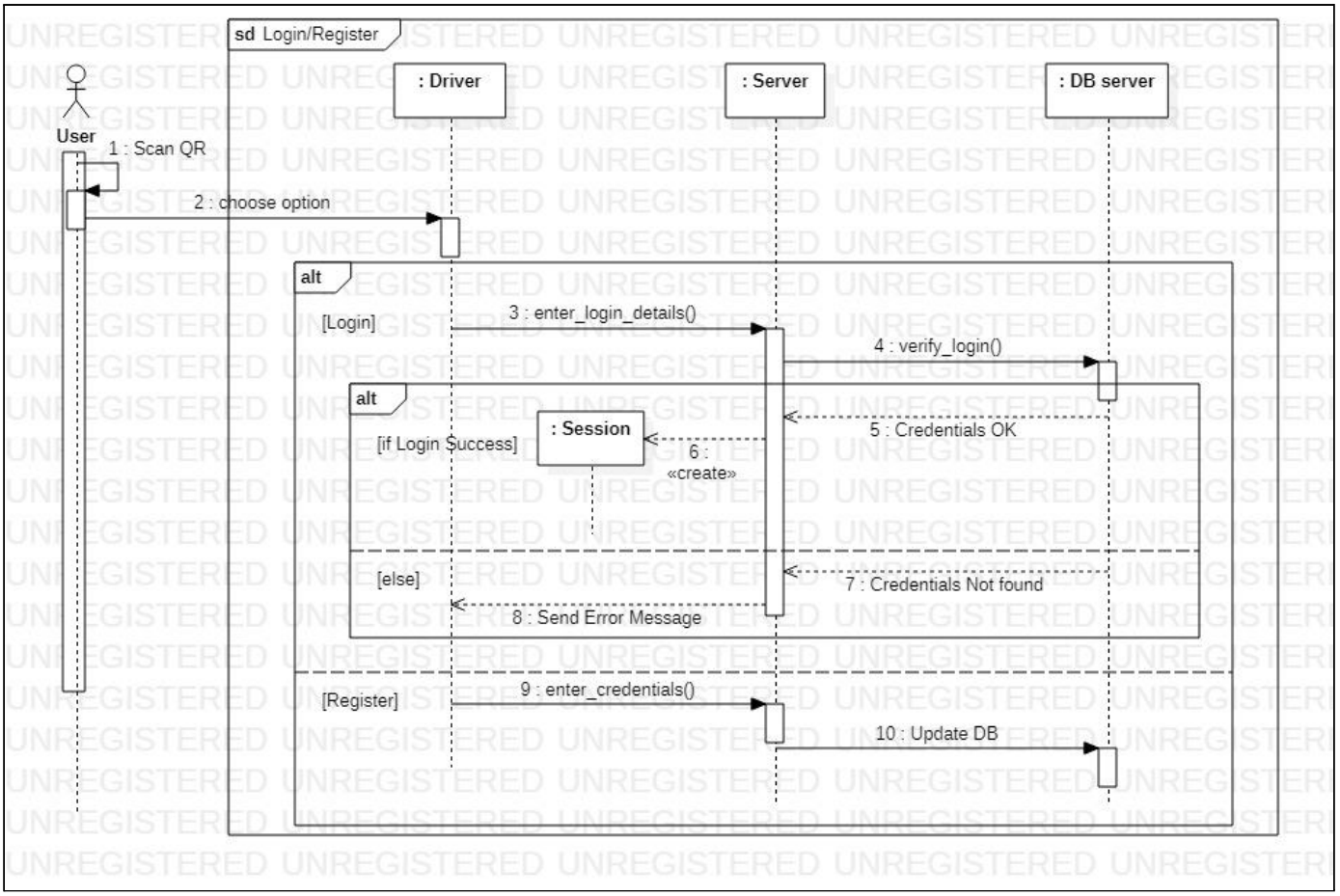
- Driver - Vehicle (One-to-Many) : Association name - owns
- Driver - Session (One-to-One) : Association name - invokes
- Vehicle - Parking Slot (One-to-One) : Association name - parked in
- Camera - Parking Slot (One-to-One) : Association name - monitors
- Sensor - Parking Slot (One-to-One) : Association name - senses
- Driver - Route (One-to-One) : Association name - gets
- Driver - Message (One-to-Many) : Association name - gets
- Fingerprint Scanner - Driver (One-to-One) : Association name - scans
- Driver - Payment (One-to-One) : Association name - makes
- IOT Module - Server (One-to-One) : Association name - connected to
- DB Server - Server (One-to-One) : Association name - connected to

SEQUENCE DIAGRAM

Scenarios:

1) Login/Register
2) Alert Unauthorised Login, Send Response
3) Select Vehicle Details
4) Update Profile
5) Check Free Space, Lock a Spot
6) Show Navigation, Record Fingerprint
7) Aid Vehicle Positioning
8) Capture Vehicle Number
9) Verify Fingerprint
10) Alert Time Exceeded
11) Make Payment, Generate Invoice
12) Unpark, Logout

1) Login/Register:

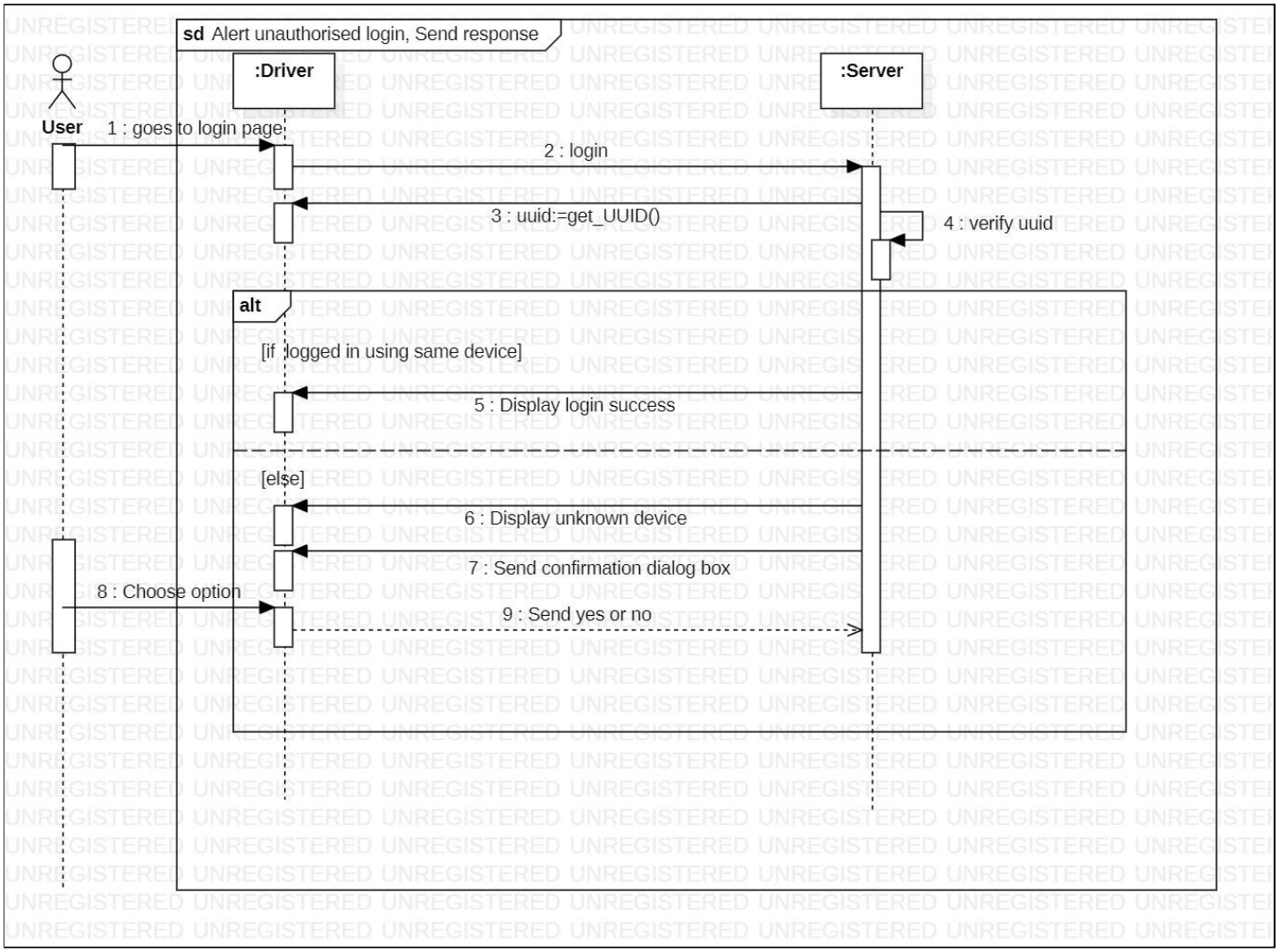


Actors: User
Objects: Driver, Server, DB Server

★ The User scans the QR code and if he's a new user, he is asked to register and the details are recorded in the DB Server.

★ If the User already has an Account, he/she is asked to enter details to login. The details entered are verified with the DB Server and if they match, an Session object is created. Else an error message is sent to the User.

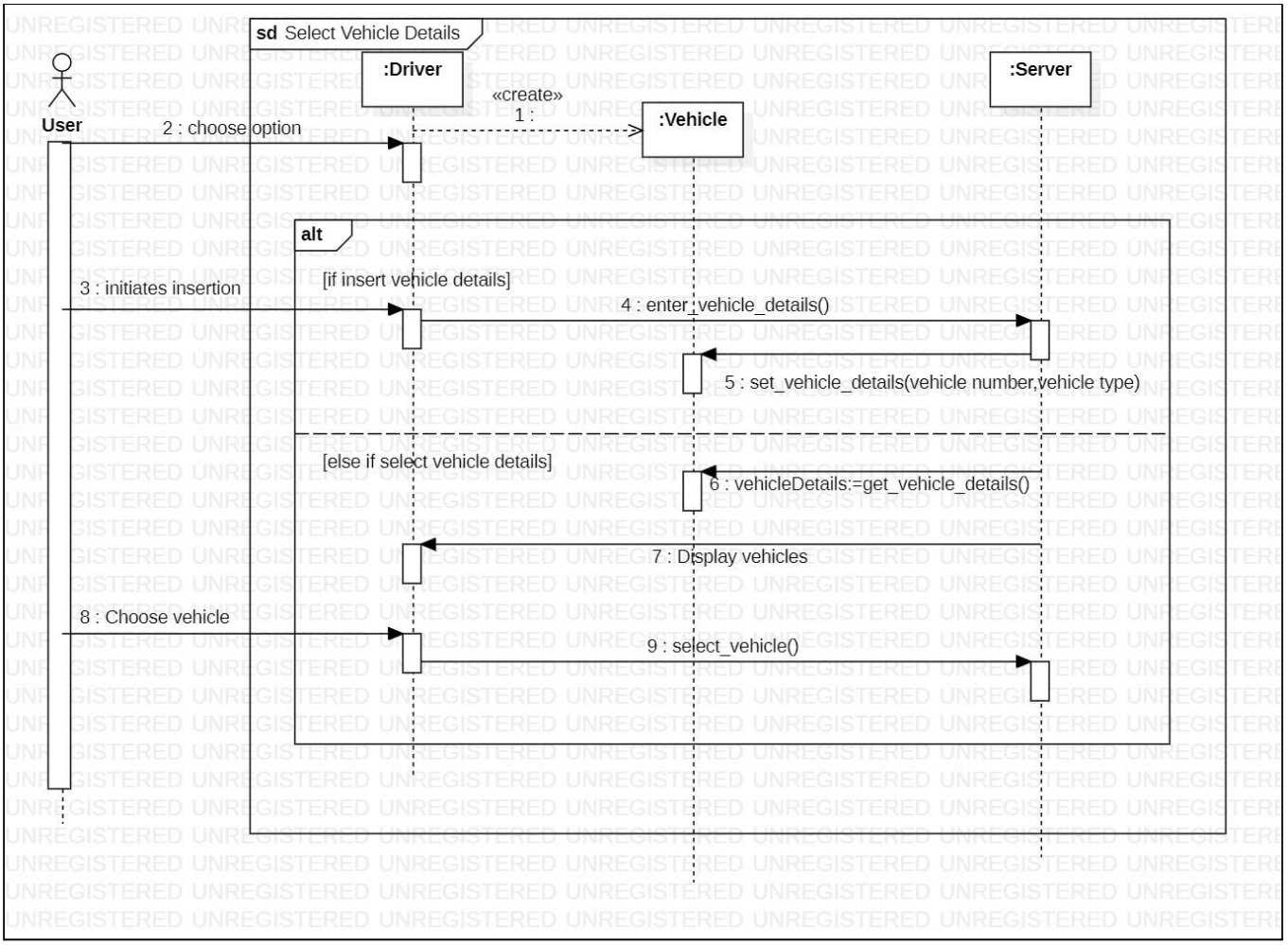
2) Alert Unauthorised Login, Send Response:



Actors: User
Objects: Driver, Server, DB Server

★ When the user logs in, the Server checks for the device ID.
★ If the user logs in using a different device, an alert message is sent to the owner of the vehicle and he sends a yes/no to allow access or to restrict.
★ If the same device is used, The message ‘Login success’ is displayed

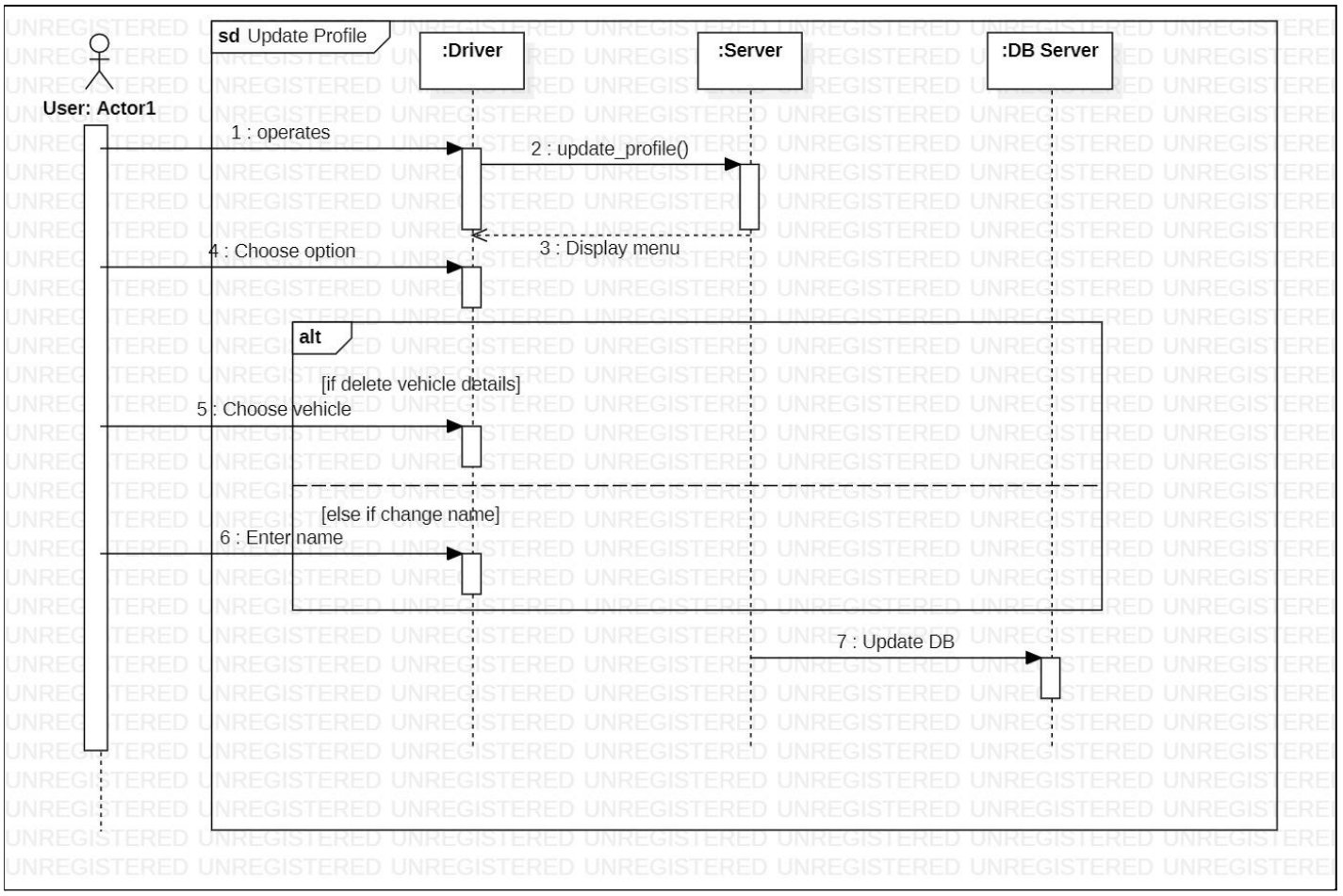
3) Select Vehicle Details



Actors: User
Objects: Driver, Vehicle, Server

★ To select a vehicle, the user can either choose a new vehicle's details that needs to be inserted or he/she can select a vehicle's details from the list displayed.
★ If the user chooses to insert a new vehicle, he enters the vehicle details.
★ Else, the user selects a vehicle from the list displayed.

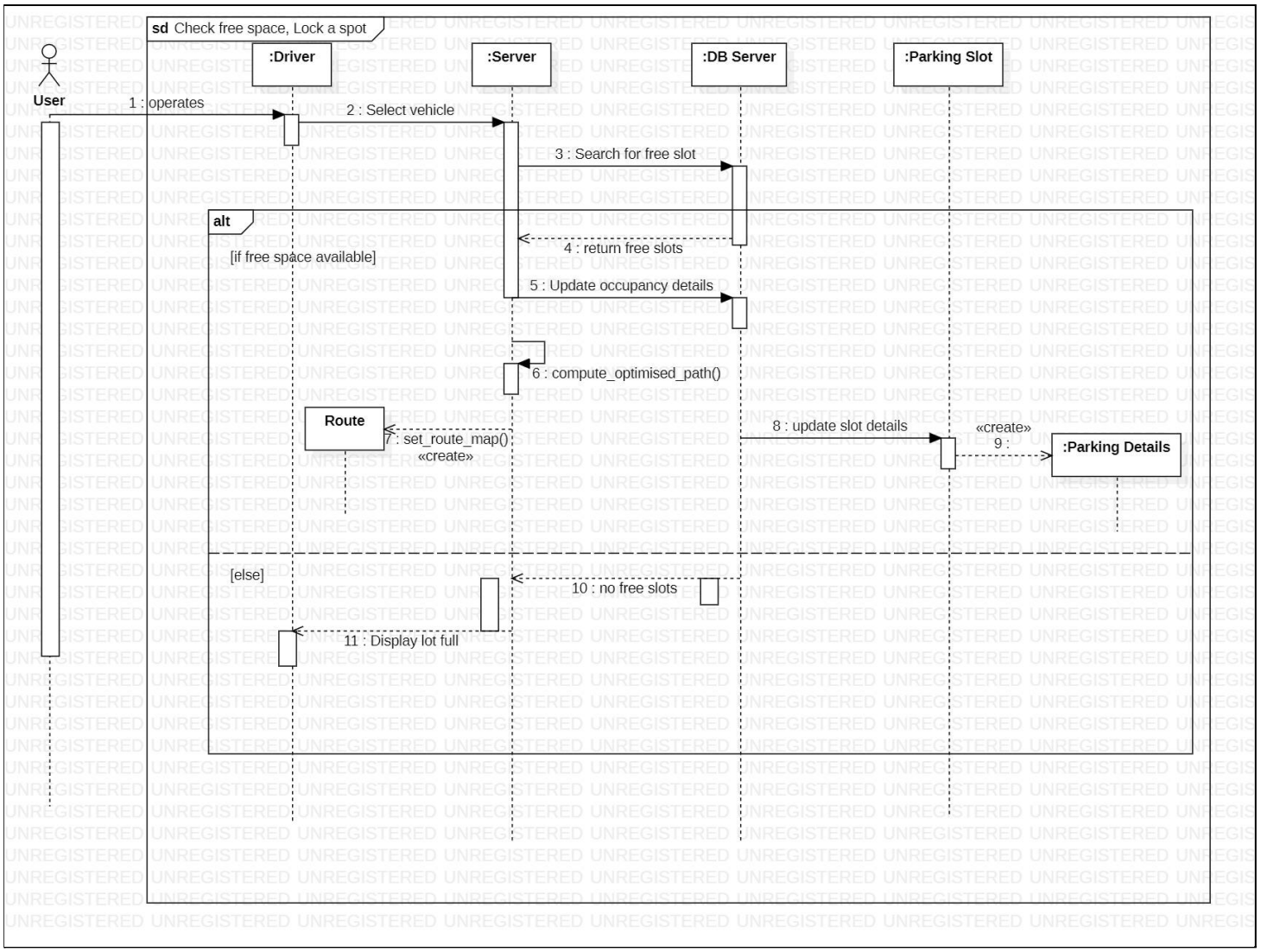
4) <u>Update Profile</u>



Actors: User
Objects: Driver, Server, DB Server

★ The user chooses to update their profile. The server displays the menu.
★ The user chooses whether to delete vehicle details or change username.
★ If the user chooses to delete vehicle details, he selects a vehicle to delete.
★ Else the user enters the new name and modifies the name.
★ The user's profile is updated.

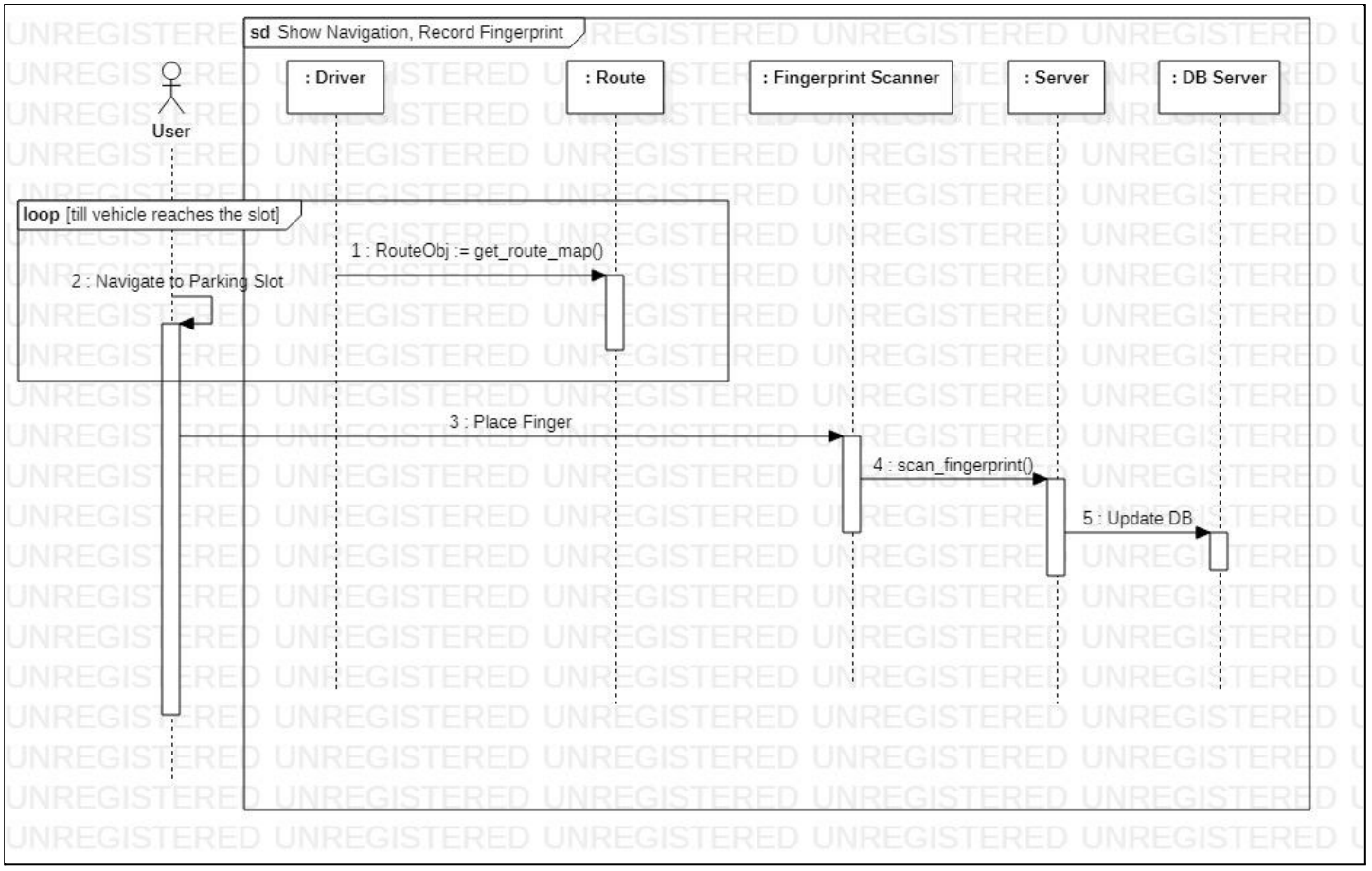
5) <u>Check Free Space, Lock a Spot</u>



Actors: User
Objects: Driver, Server, DB Server

★ Once a vehicle is selected by the driver, the server finds a free space and locks a spot.The user selects a vehicle.
★ The server checks for a free slot.
★ The server returns a free slot, if free space is available.
★ Else the server informs the user that no space is available.
★ The occupancy details of the slot are updated.
★ The optimised route to reach that locked slot is provided by the server.

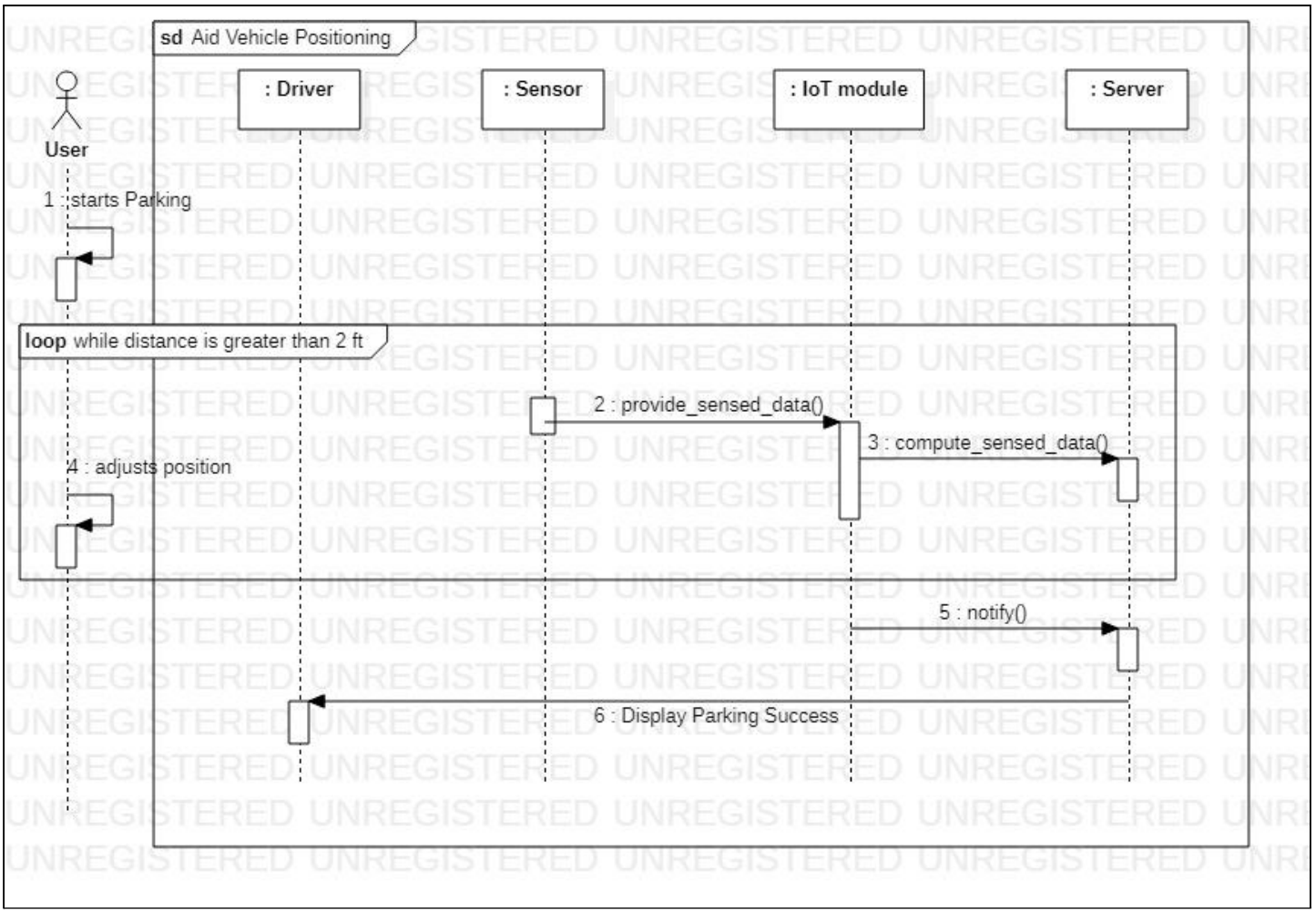
6) <u>Show Navigation, Record Fingerprint</u>



Actors: User
Objects: Driver, Route, Fingerprint Scanner, Server, DB Server

★ Once a spot is locked by the server, the server shows navigation to reach that locked spot.
★ The driver records the fingerprint at the spot.
★ The route is provided by the server to reach the locked spot.
★ The user uses the route and navigates to the slot.
★ Once the user reaches the slot, his fingerprint is recorded using the fingerprint scanner.

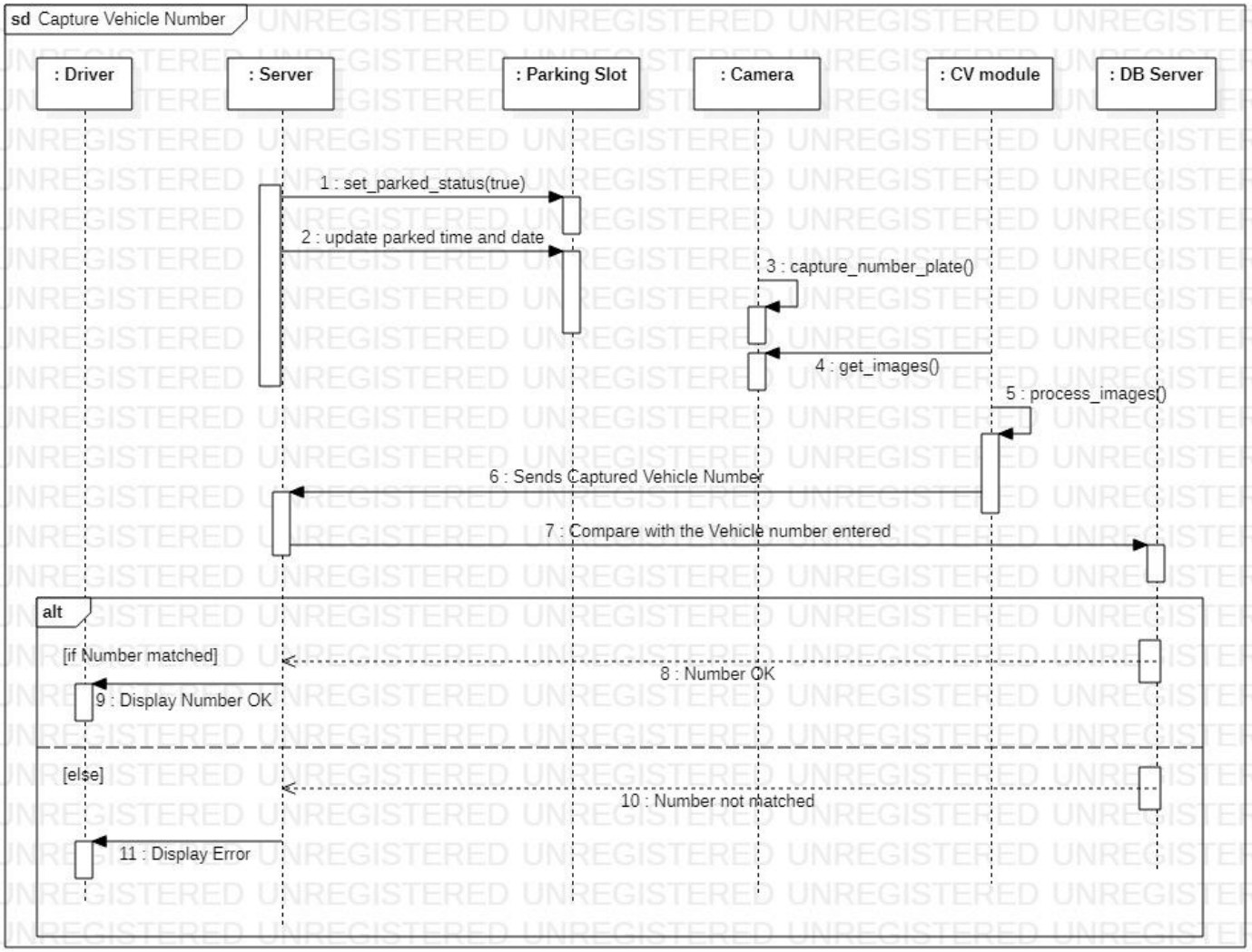
7) Aid Vehicle Positioning



Actors: User
Objects: Driver, Sensor, IoT Module, Server

★ At the parking slot, the IoT Module helps the user in vehicle positioning.
★ Once the driver starts parking, the sensor sends the data to the IoT module which computes the distance.
★ When the desired threshold distance is attained, The IoT module notifies the server about the successful park.
★ 'Parking Success' message is displayed.

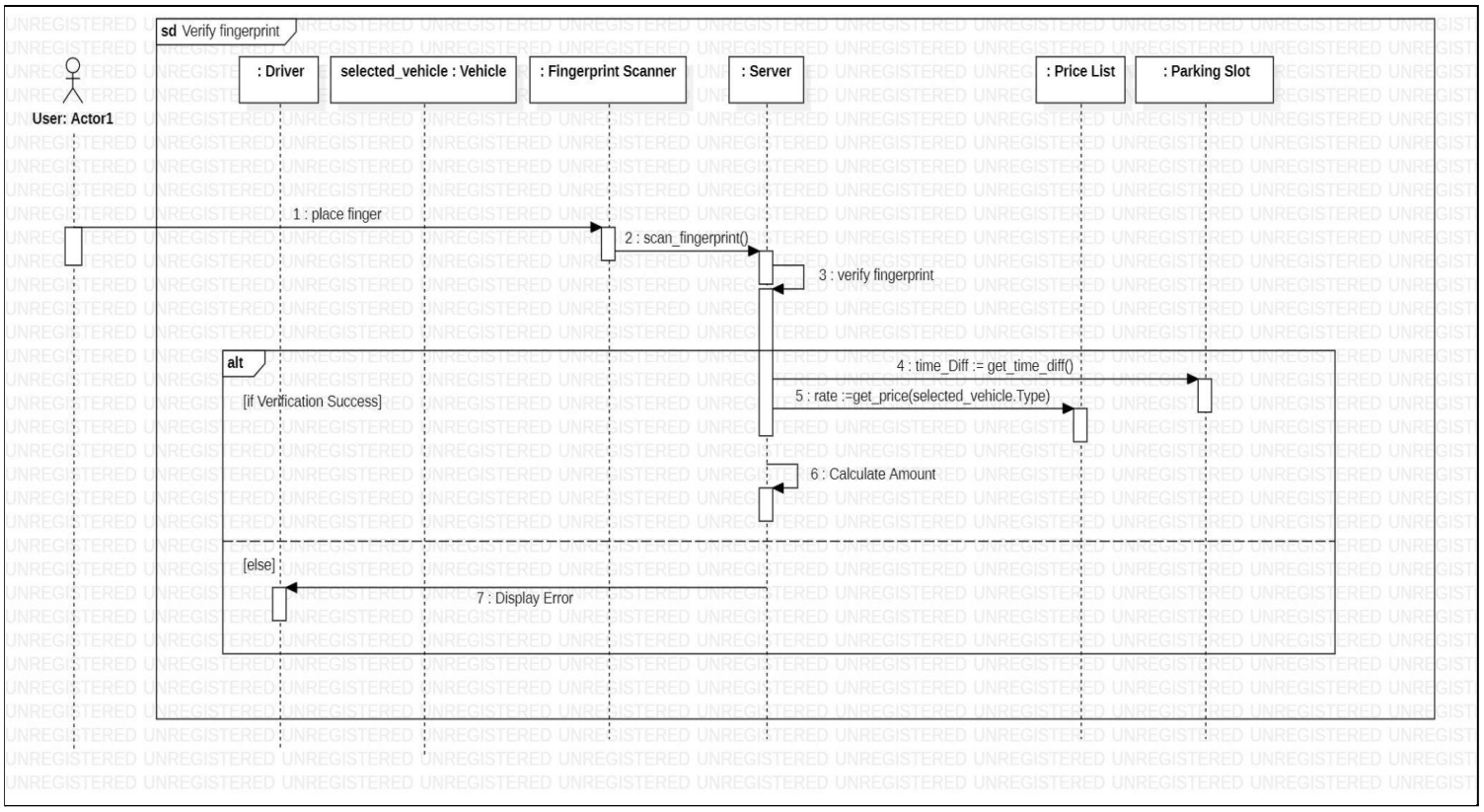
8) <u>Capture Vehicle Number</u>



Actors: -
Objects: Driver, Server, Parking slot, Camera, CV Module, DB Server

★ The camera, a part of CV Module, captures the vehicle plate of the user.
★ The parked status of the slot is updated.
★ The parked time and date is updated.
★ The user's vehicle plate is captured.
★ The CV module processes the images captured by camera.
★ The vehicle number extracted from the image captured is compared with the vehicle number entered by the user.
★ If the number matches, the server displays Number OK.
★ Else the server informs the user that the number is not matched.

9) <u>Verify Fingerprint</u>

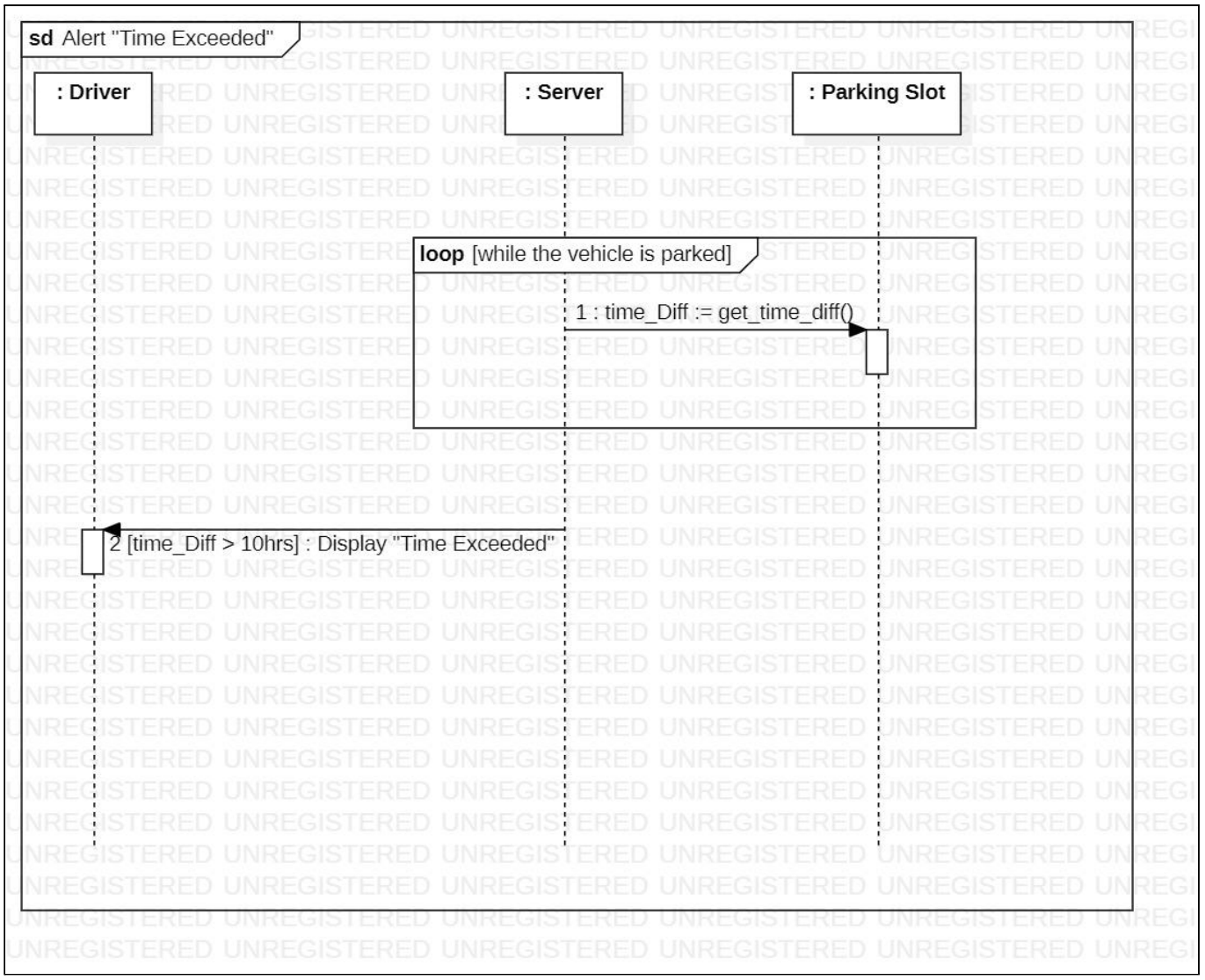


Actors: User

Objects: Driver, Vehicle, Fingerprint Scanner, Server, PriceList, Parking slot, Parking details

- ★ The user re-records his fingerprint. The server verifies his fingerprint.
- ★ If the verification succeeds, the number of hours for which the vehicle was parked is calculated.
- ★ The amount for the number of hours parked is also calculated.
- ★ If verification fails, an error message is displayed.

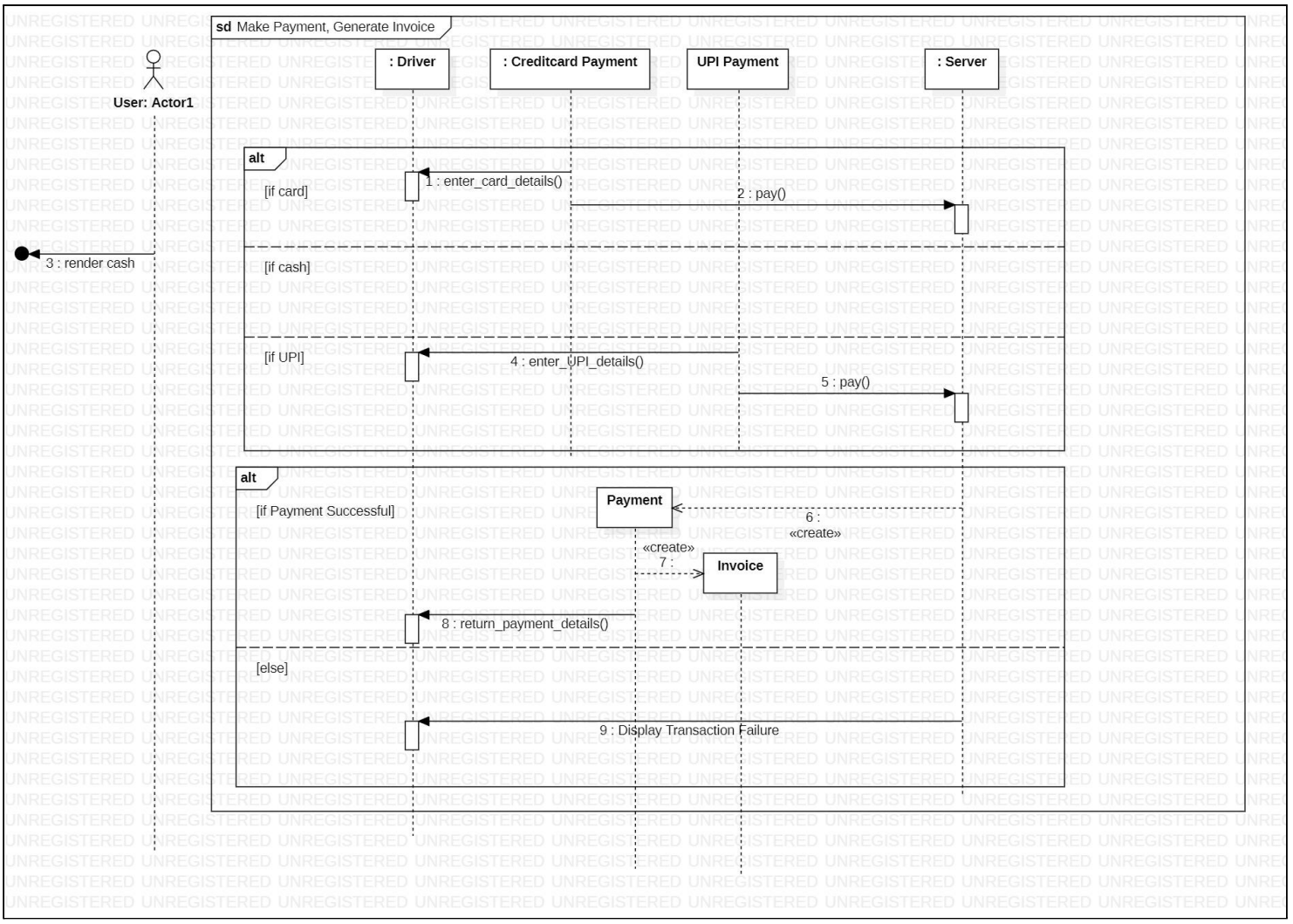
10) <u>Alert Time Exceeded</u>



Actors: -
Objects: Driver, Server, Parking slot

- ★ If the user has exceeded his maximum parking time limit, the server sends an alert message to the user indicating time exceeded.
- ★ The number of hours for which the vehicle was parked is calculated.
- ★ If the number of hours exceeds 10, an alert message indicating time exceeded is displayed.

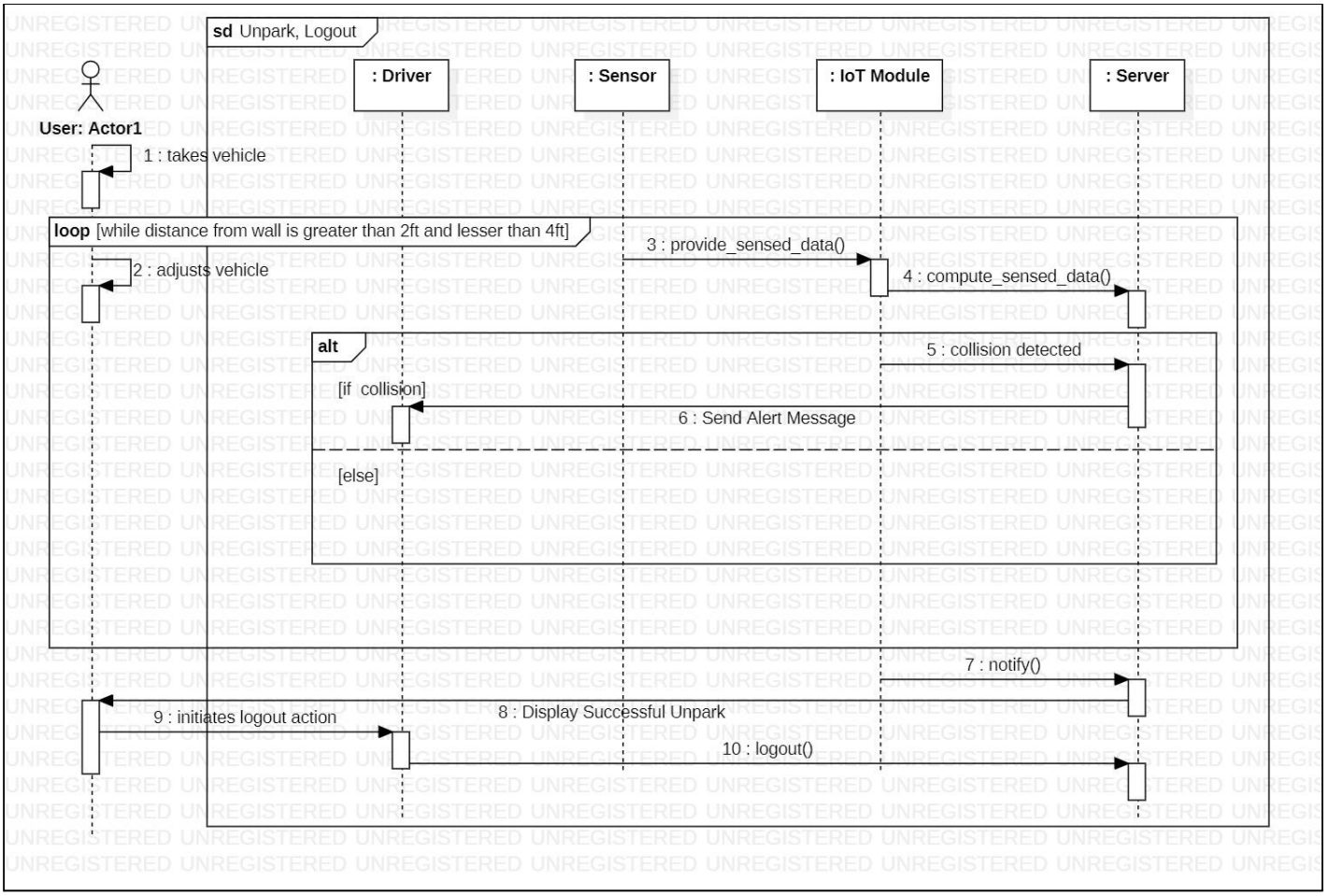
11) <u>Make Payment, Generate Invoice</u>



Actors: User
Objects: Driver, Server
Description:

★ The user makes payment.
★ If the transaction is successful, an invoice is generated.
★ The user chooses a mode of payment to make payment.
★ If he chooses cash, he renders cash.
★ If he chooses credit card payment, he enters the card details.
★ If he chooses UPI payment, he enters the UPI details.
★ The payment processor processes the payment.
★ An invoice is generated after successful payment transaction.
★ If the transaction fails, transaction failure error message is displayed.

12) <u>Unpark, Logout</u>



Actors: User
Objects: Driver, Sensor, IoT Module, Server

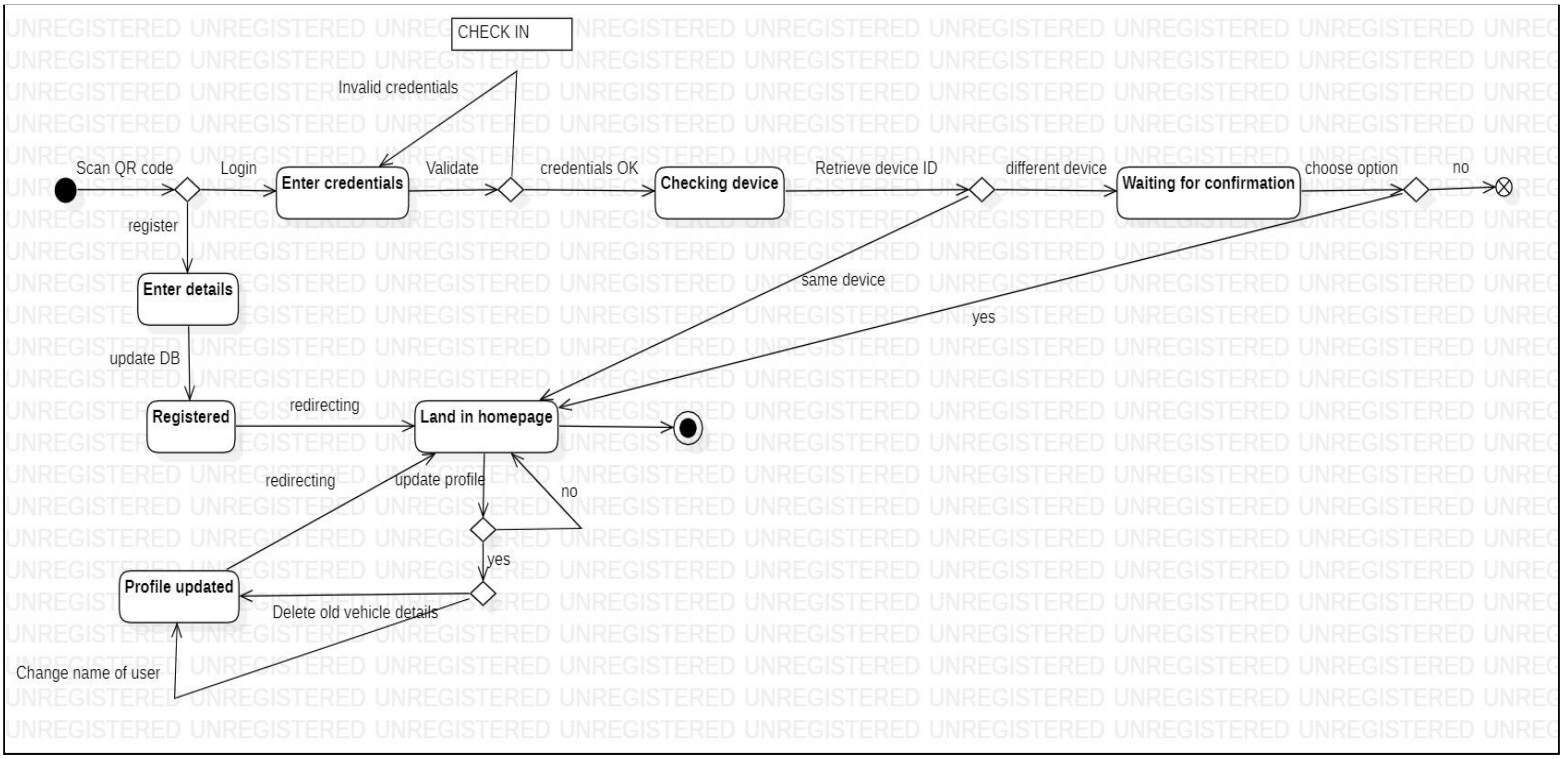
- ★ To Unpark, the User starts his/her vehicle.
- ★ The User adjusts his vehicle till he meets the successful unpark distance. The Sensor and IoT module help in calculating the distance left and also detect Collision.
- ★ In case of collision, The Server is alerted and it sends a message to the User.
- ★ When the required distance is finally reached, the IoT module is notified and the 'Successful Unpark' message is displayed.
- ★ Once successfully unparked, The logout action is initiated and the User is logged out of the account.

STATE MODEL

1. **Check in**
2. **Parking Automation**
3. **Payment and Check out**

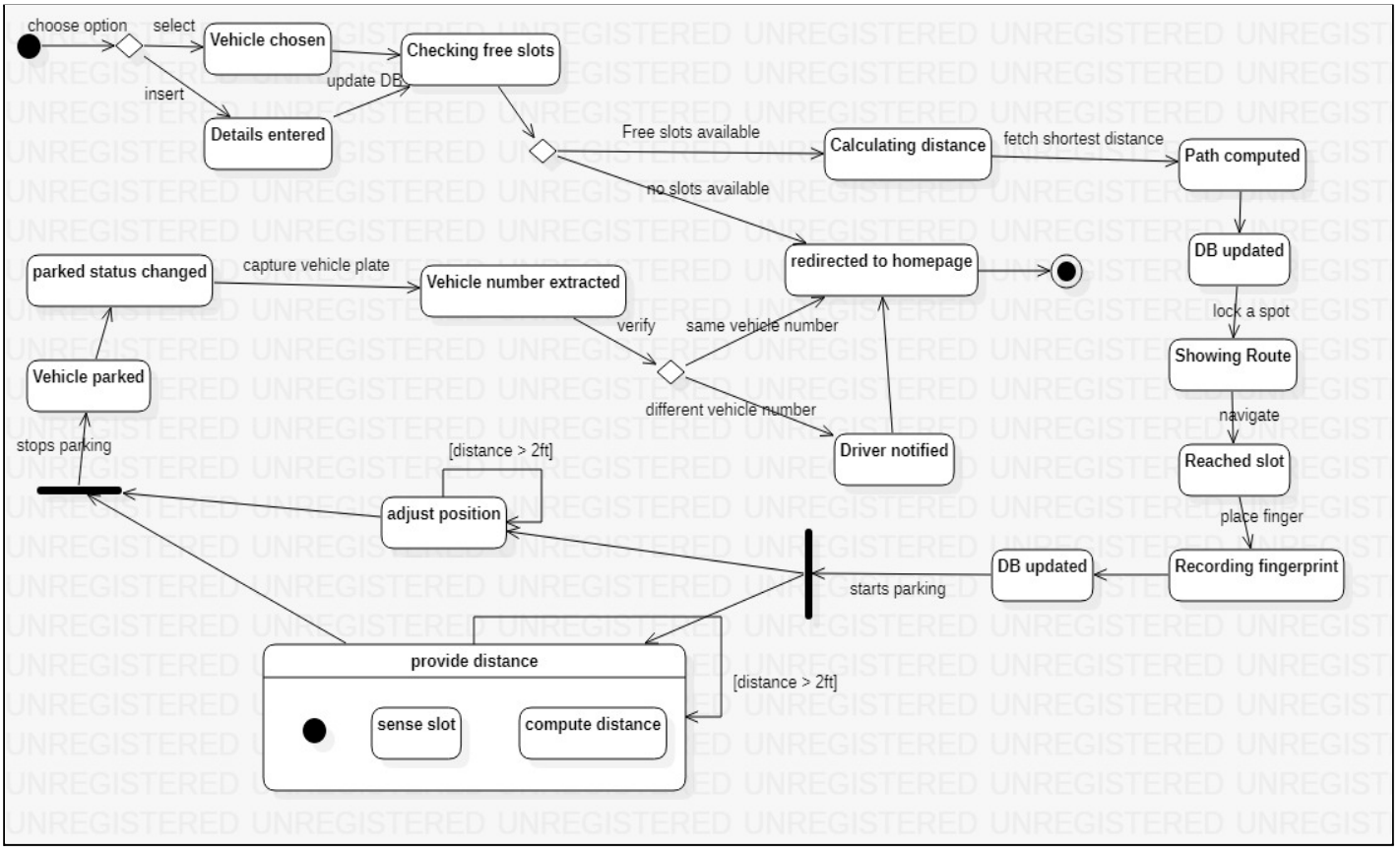
i) CHECK IN

1. Firstly, the User scans the QR code and can either login(has an existing account) or register (new User). For login, goto step 3. For Register goto step 2.
2. For registering the new User, the details entered are recorded and the User is directed to the homepage.
3. The user is asked to enter credentials which are then checked with the DB Server, If they are valid, goto step 3. Else the User is redirected to enter his/her credentials again.
4. After validation success, The Device ID from the device using which the user has logged in is retrieved and is again checked to see if the device details are already recorded.If yes, The user is landed in the homepage. Else the user is notified of the new device and awaits a response. If the user acknowledges, the new device details are recorded and the device is granted access into account and directed to the homepage. Else, In case of negative response from the user, the control flow is terminated as the device is unrecognised.
5. If the user wants to update his/her profile, the Update Profile button is clicked and the old vehicle details can be removed or the user name can be modified. Once the changes are recorded in the DB Server, the User is redirected to the homepage.

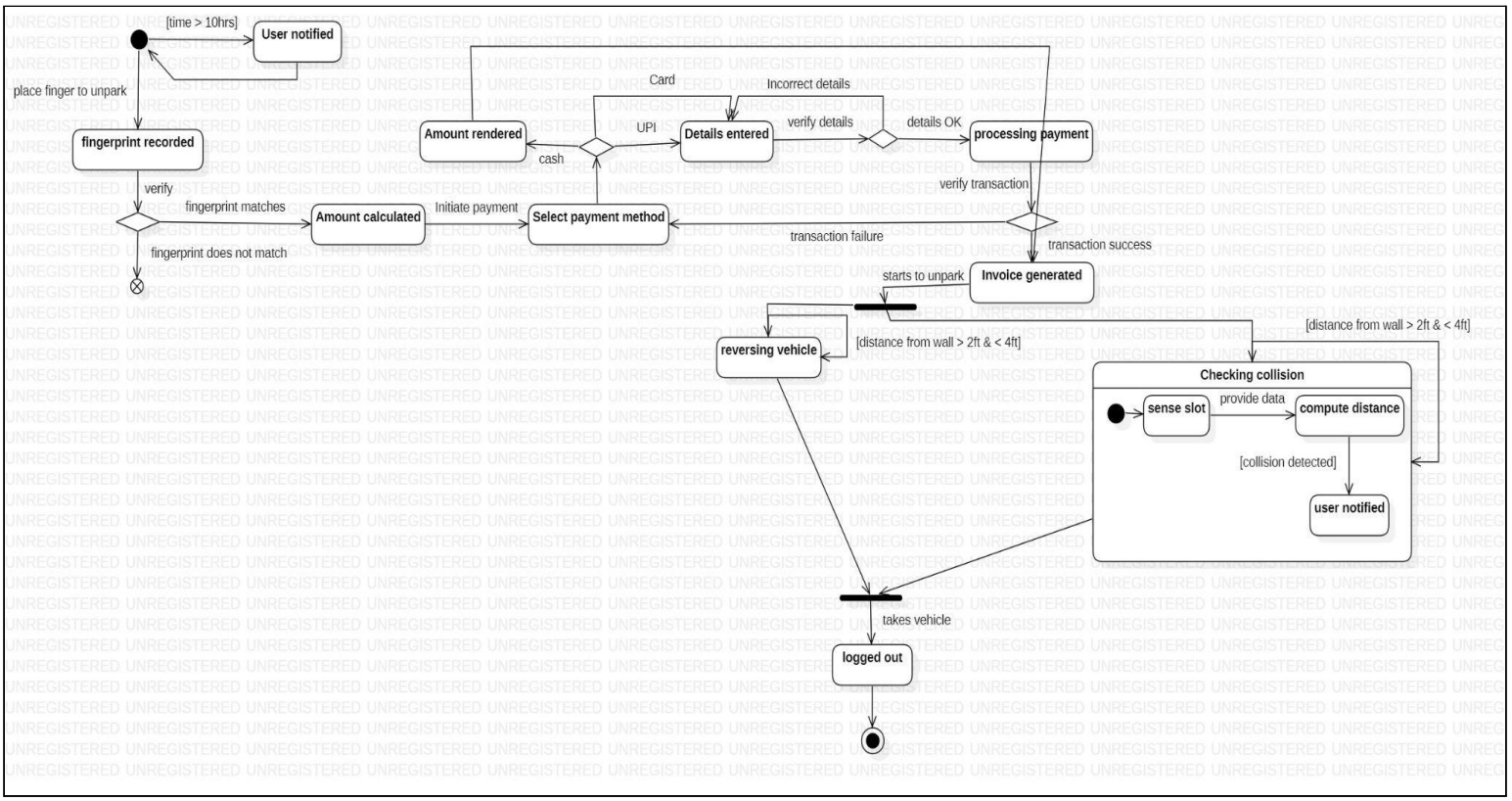


ii) PARKING AUTOMATION

1. From the homepage, the User can either select a vehicle from the list of Vehicles displayed to him which were recorded during previous visits or include a new vehicle that he is currently parking.
2. The DB Server is updated with the vehicle details and the Server checks for free spots. If there are no slots available at the moment, the User is redirected to the homepage. Else goto step 3.
3. When there are free slots available, the Server computes the distance to the slot that is closest to the vehicle, locks it, updates DB Server and sends a Route Object to the User for navigation.
4. Once the Vehicle reaches the directed slot, The User's fingerprint is scanned and uploaded to be used at the time of unparking to prevent thefts.
5. To park in the slot, the Sensors around the slot send data to their IoT Module which computes the distance to successfully park within the slot. The Vehicle is deemed to be successfully parked when the data provided by the sensors are within the IoT module's range.
6. Once parked, The Parked status of the slot is changed and The Camera captures the number plate and extracts the Vehicle Number from the image sent to eliminate the possibility of human error occurring. The User is redirected to the homepage when the numbers match.
7. In case of a mismatch, the User is notified and is directed to the homepage.



iii) PAYMENT AND CHECK OUT



1. The Parking time duration is 10 hrs, so when the time exceeds the driver is notified.
2. To Unpark, The User is asked to place finger and the fingerprints need to match for the User to continue unparking. If the fingerprints mismatch, the process flow is terminated (It is most likely a theft attempt).
3. After verification of fingerprint, the amount to be paid is calculated and the User can pay in one of the three methods.
    a. Cash Payment - Step 4
    b. CreditCard Payment - Step 5
    c. UPI payment - Step 6
4. For the cash method, the User renders the Amount and change is returned by the cashier. Goto Step 7
5. For the CreditCard payment, the required details are entered and verified with the help of a third-party gateway that helps in CreditCard payment (The Payment Processor). Goto Step 7
6. For UPI payment, the details entered are verified with the payment processor. Goto Step 7
7. After verification, the amount is debited thereby making a successful transaction. Invoice is generated. If there's a failure, the control is transferred to retry payment by selecting a method.

8. During unpark, the IoT Module with the help of its sensors checks for any collision. In case of a collision, the user is notified.
9. After successful unpark, the User is logged out of the account.

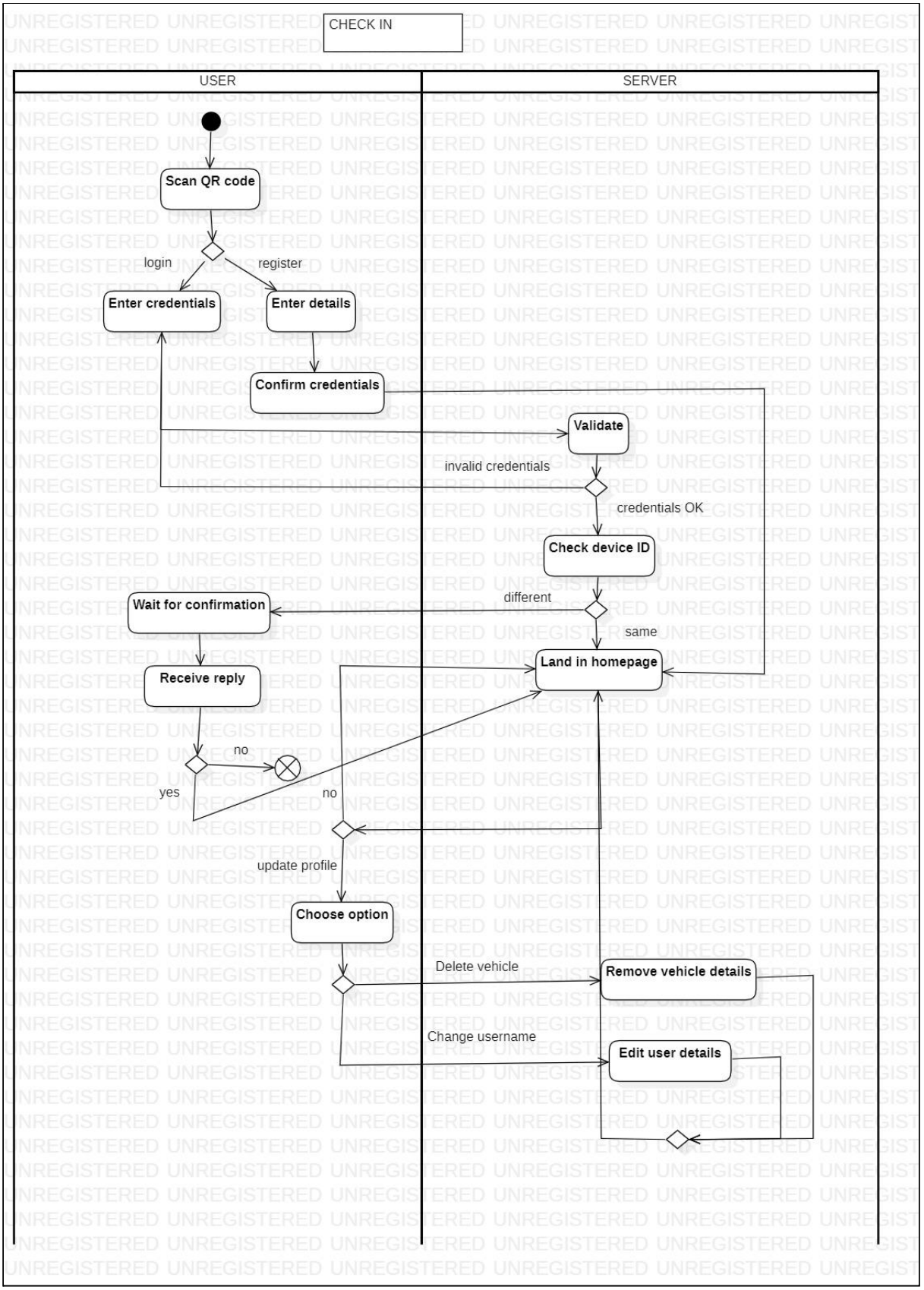
ACTIVITY DIAGRAM

1. **Check in**
2. **Parking Automation**
3. **Payment and Check out**

i) CHECK IN

Objects identified: User, Server

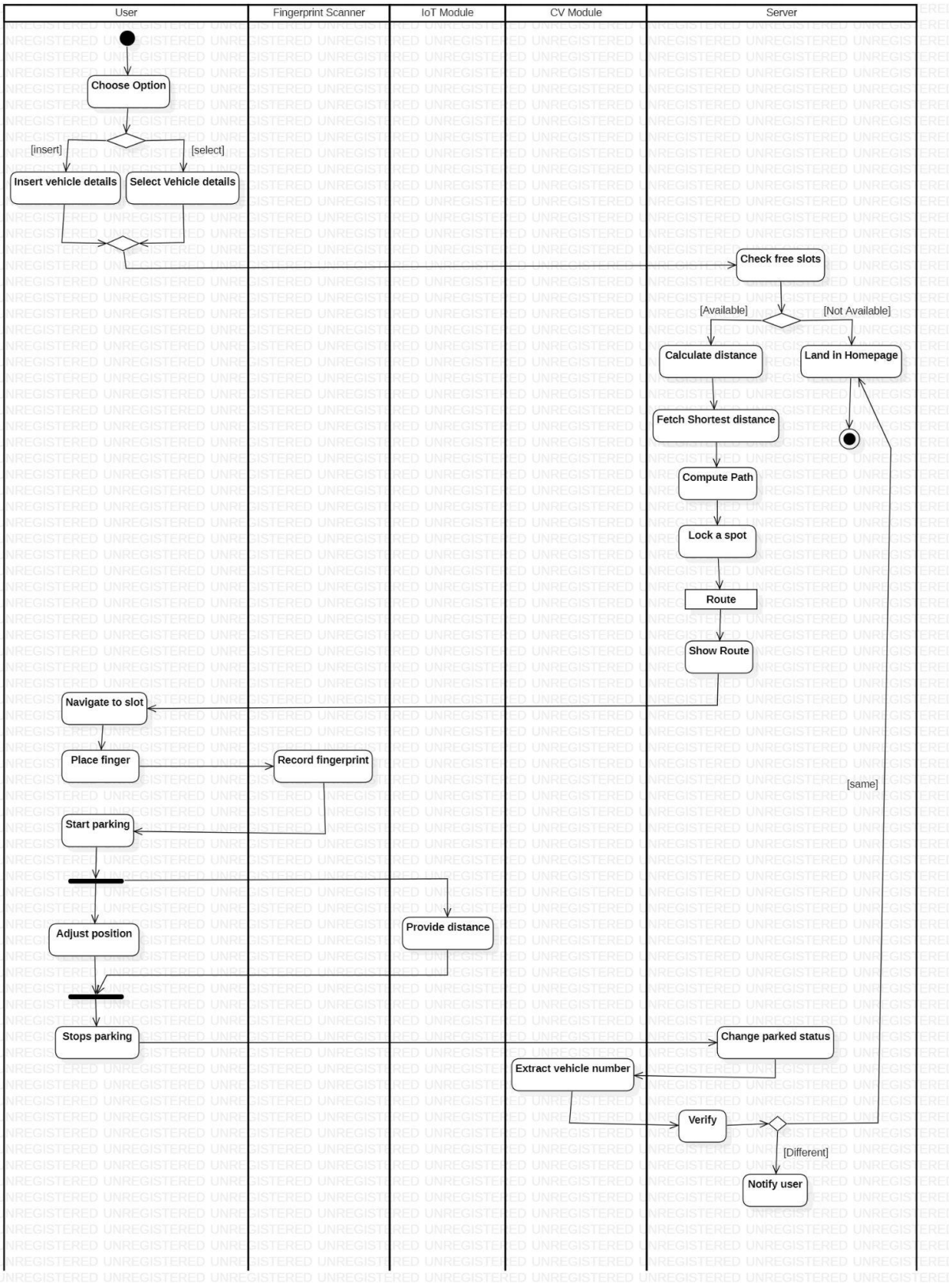
- Firstly, the user scans the QR code.
- If the user is a new user, he registers by entering his credentials.
- If he is an existing user, he enters his login credentials.
- The server verifies the login credentials entered by the user.
- If the credentials are invalid, login fails.
- If the credentials are valid, then the device ID is checked by the server.
- If the device ID is the same as the one he used before, the user is redirected to the homepage.
- If the device ID is different, then an alert message is sent to the owner of the vehicle.
- The owner of the vehicle sends a response message (yes/no) indicating whether to grant access or not.
- If the user sends ‘yes’, the user is redirected to the homepage.
- If the user sends ‘no’, login fails.
- If the user is interested to update profile, he can either choose ‘delete vehicle details’ or ‘change username’.
- If the user chooses the ‘delete vehicle details’ option, he should choose which vehicle to delete. The server then queries the DB server and DB server deletes the details of the vehicle chosen.
- If the user chooses the ‘change username’ option, he should enter his new username. The DB will be updated.


CHECK IN
USER
SERVER
Scan QR code
login
register
Enter credentials
Enter details
Confirm credentials
Validate
invalid credentials
credentials OK
Check device ID
different
same
Wait for confirmation
Land in homepage
Receive reply
no
yes
no
update profile
Choose option
Delete vehicle
Remove vehicle details
Change username
Edit user details

ii) PARKING AUTOMATION

Objects identified: User, Fingerprint Scanner, Server, IoT Module, Payment Processor

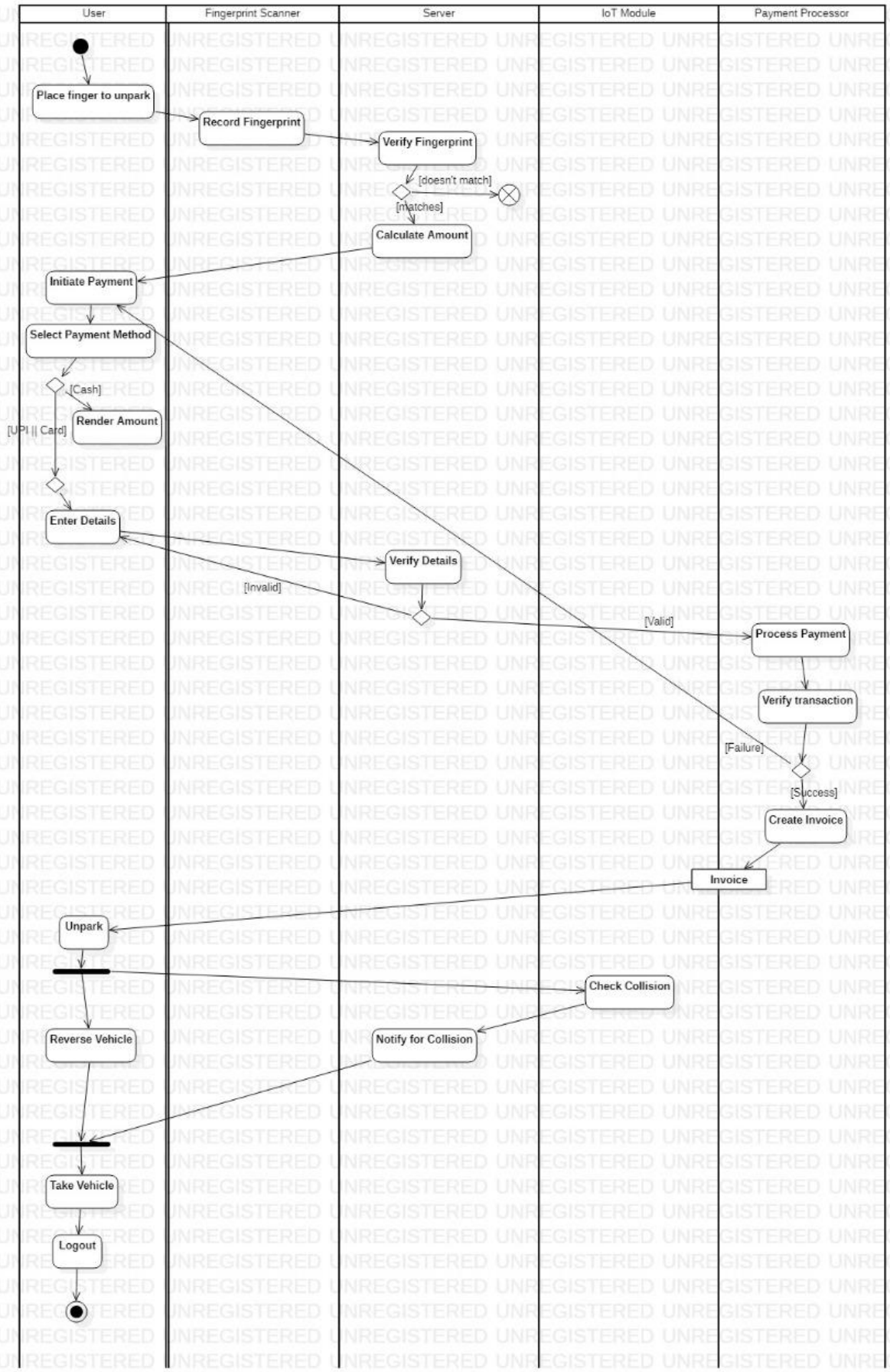
- The user chooses either to select a vehicle from his previous visits or to insert a new vehicle.
- Once the user selects a vehicle, the server checks for free space for that vehicle.
- If the space is available, the server computes the optimised path to a free slot.
- The server locks the spot.
- The server provides the route to the user to reach that locked spot.
- The user navigates to the slot using the route provided by the server.
- At the parking slot, the user records his fingerprint.
- The user starts parking.
- Sensor senses and signals until the user has parked at the right position.
- The user adjusts position to park his vehicle at the right position.
- The user stops parking.
- Once the user has parked his vehicle, the parked status of the parking slot is updated.
- The CV Module extracts the vehicle number from the image of the vehicle number plate captured.
- The server verifies the vehicle number with the number entered by the user.
- If the number doesn't match, then the user is notified.
- If the number matches, the user is redirected to the homepage.


User
Fingerprint Scanner
IoT Module
CV Module
Server
Choose Option
[insert]
[select]
Insert vehicle details
Select Vehicle details
Check free slots
[Available]
[Not Available]
Calculate distance
Land in Homepage
Fetch Shortest distance
Compute Path
Lock a spot
Route
Show Route
Navigate to slot
Place finger
Record fingerprint
[same]
Start parking
Adjust position
Provide distance
Stops parking
Change parked status
Extract vehicle number
Verify
[Different]
Notify user

iii) PAYMENT AND CHECK OUT

Objects identified: User, Server, Fingerprint Scanner, IoT Module, CV Module

- The user places his finger in the fingerprint scanner before unparking.
- The fingerprint scanner records his fingerprint.
- The server verifies the fingerprint with the fingerprint recorded by the user before he parked his vehicle.
- If the fingerprints match, the server calculates the amount for the number of hours the user has parked his vehicle.
- The user makes payment.
- The user selects the mode of payment.
- If he chooses cash payment, he will render the amount.
- If he chooses a card or UPI, then he will enter his credentials.
- The server verifies the card or UPI details.
- The payment processor will process the payment and verifies the transaction.
- If the transaction is successful, an invoice is generated.
- If the transaction fails, the user will be again asked to choose the mode of payment.
- The user unparks his vehicle.
- The IoT Module checks for any collision. The user reverses his vehicle.
- If collision exists, the IoT Module notifies the user.
- Else, the user takes his vehicle.
- Finally, he logs out.



CODE GENERATION FROM CLASS MODEL

1. Driver

```
import java.util.*;

/**
 *
 */
public class Driver implements Profile_update<<interface>>,
Login_logout<<interface>>, Register<<interface>> {

    /**
     * Default constructor
     */
    public Driver() {
    }

    /**
     *
     */
    private int Member ID;

    /**
     *
     */
    private String Username;

    /**
     *
     */
    private String Password;

    /**
     *
     */
    private String Name;

    /**
     *
     */
    private String UUID;

    /**
```

```
 *
 */
private Object Fingerprint;

/**
 *
 */
private ArrayList<Vehicle> Vehicle List;




/**
 * @return
 */
public int get_member_ID() {
    // TODO implement here
    return 0;
}

/**
 *
 */
public void enter_vehicle_details() {
    // TODO implement here
}

/**
 * @return
 */
public String select_vehicle() {
    // TODO implement here
    return "";
}

/**
 * @return
 */
public String get_UUID() {
    // TODO implement here
    return "";
}
```

```
/**
 * @param Object
 */
public void set_fingerprint(void Object) {
    // TODO implement here
}

/**
 * @return
 */
public Object get_fingerprint() {
    // TODO implement here
    return null;
}

/**
 *
 */
public void update_profile() {
    // TODO implement here
}

/**
 *
 */
public void enter_login_details() {
    // TODO implement here
}

/**
 * @return
 */
public Boolean verfiy_login() {
    // TODO implement here
    return null;
}

/**
 *
 */
public void logout() {
    // TODO implement here
}
```

```
    /**
     *
     */
    public void enter_credentials() {
        // TODO implement here
    }

}
```

2. Vehicle

```
import java.util.*;

/**
 *
 */
public class Vehicle {

    /**
     * Default constructor
     */
    public Vehicle() {
    }

    /**
     *
     */
    private String Vehicle Number;

    /**
     *
     */
    private String Vehicle Type;


    /**
     * @param String
     * @param String
     */
    public void set_vehicle_details(void String, void String) {
        // TODO implement here
    }
```

```
    /**
     * @return
     */
    public String get_vehicle_details() {
        // TODO implement here
        return "";
    }

}
```

3. Route

```
import java.util.*;

/**
 *
 */
public class Route {

    /**
     * Default constructor
     */
    public Route() {
    }

    /**
     *
     */
    private String Vehicle Number;

    /**
     *
     */
    private int Entry Point;

    /**
     *
     */
    private String Slot ID;

    /**
     *
     */
    private Object Route Map;
```

```
    /**
     * @param Object
     */
    public void set_route_map(void Object) {
        // TODO implement here
    }

    /**
     * @return
     */
    public Object get_route_map() {
        // TODO implement here
        return null;
    }

}
```

4. Message

```
import java.util.*;

/**
 *
 */
public class Message {

    /**
     * Default constructor
     */
    public Message() {
    }

    /**
     *
     */
    private int Message ID;

    /**
     *
     */
    private String Message Content;

    /**
     *
     */
    private Time Delivered Time;
```

```
    /**
     *
     */
    private String Response;

    /**
     *
     */
    private Time Response Time;


    /**
     * @param String
     */
    public void set_response(void String) {
        // TODO implement here
    }

    /**
     * @param Time
     */
    public void set_response_time(void Time) {
        // TODO implement here
    }

    /**
     * @param int
     * @param String
     * @return
     */
    public Message send(void int, void String) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public String get_content() {
        // TODO implement here
        return "";
    }

}
```

5. Fingerprint Scanner

```
import java.util.*;

/**
 *
 */
public class Fingerprint Scanner {

    /**
     * Default constructor
     */
    public Fingerprint Scanner() {
    }

    /**
     *
     */
    private int Device ID;


    /**
     *
     */
    public void scan_fingerprint() {
        // TODO implement here
    }

}
```

6. Session

```
import java.util.*;

/**
 *
 */
public class Session {

    /**
     * Default constructor
     */
    public Session() {
    }

    /**
     *
     */
    private int Session ID;
```

```
    /**
     *
     */
    private Time Login Timestamp;

    /**
     *
     */
    private Date Login Date;

    /**
     *
     */
    private Time Logout Timestamp;

    /**
     *
     */
    private Date Logout Date;


    /**
     * @return
     */
    public String return_session_details() {
        // TODO implement here
        return "";
    }

    /**
     *
     */
    public void set_logout_timestamp_date() {
        // TODO implement here
    }

}
```

7. Pricelist

```
import java.util.*;

/**
 *
 */
public class PriceList {
```

```
    /**
     * Default constructor
     */
    public PriceList() {
    }

    /**
     *
     */
    private Double Amount per hr;

    /**
     *
     */
    private String Vehicle Type;


    /**
     * @param String
     * @return
     */
    public double get_price(void String) {
        // TODO implement here
        return 0.0d;
    }

}
```

8. Parking Slot

```
import java.util.*;

/**
 *
 */
public class Parking Slot {

    /**
     * Default constructor
     */
    public Parking Slot() {
    }

    /**
     *
     */
    private String Slot ID;
```

```
    /**
     *
     */
    private String Vehicle Type;

    /**
     *
     */
    private Boolean Parked status;

    /**
     *
     */
    private Parking Details Parking_details;




    /**
     * @return
     */
    public Boolean get_parked_status() {
        // TODO implement here
        return null;
    }

    /**
     * @param Boolean
     */
    public void set_parked_status(void Boolean) {
        // TODO implement here
    }

}
```

9. Parking Details

```
import java.util.*;

/**
 *
 */
public class Parking Details {

    /**
     * Default constructor
```

```
     */
    public Parking Details() {
    }

    /**
     *
     */
    private String Vehicle Number;

    /**
     *
     */
    private Time Parking Time;

    /**
     *
     */
    private Date Parking Date;

    /**
     *
     */
    private Time Unparking Time;


    /**
     * @return
     */
    public int get_time_diff() {
        // TODO implement here
        return 0;
    }

    /**
     * @param Time
     */
    public void set_time_departure(void Time) {
        // TODO implement here
    }

}
```

10. Image

```
import java.util.*;

/**
 *
 */
public class Image {

    /**
     * Default constructor
     */
    public Image() {
    }

    /**
     *
     */
    public void Pixels : int[][];


}
```

11. Camera

```
import java.util.*;

/**
 *
 */
public class Camera {

    /**
     * Default constructor
     */
    public Camera() {
    }

    /**
     *
     */
    private String Camera ID;

    /**
     *
     */
    private Image Img[];
```

```
    /**
     *
     */
    public void capture_number_plate() {
        // TODO implement here
    }

    /**
     * @return
     */
    public Image array get_images() {
        // TODO implement here
        return null;
    }

}
```

12. CV Module

```
import java.util.*;

/**
 *
 */
public class CV Module {

    /**
     * Default constructor
     */
    public CV Module() {
    }

    /**
     *
     */
    private int CV ID;

    /**
     *
     */
    private String CV password;

    /**
     *
     */
    private Camera Cameras[];
```

```
    /**
     *
     */
    public void process_camera_images() {
        // TODO implement here
    }

}
```

13. Sensor

```
import java.util.*;

/**
 *
 */
private class Sensor {

    /**
     * Default constructor
     */
    private Sensor() {
    }

    /**
     *
     */
    private String Sensor ID;



    /**
     *
     */
    public void provide_sensed_data() {
        // TODO implement here
    }

}
```

14. IoT Module

```
import java.util.*;

/**
 *
 */
public class IoT Module {

    /**
     * Default constructor
     */
    public IoT Module() {
    }

    /**
     *
     */
    private int IoT ID;

    /**
     *
     */
    private String IoT password;

    /**
     *
     */
    private Sensor Sensors[];



    /**
     *
     */
    public void compute_data_from_sensor() {
        // TODO implement here
    }

    /**
     *
     */
    public void notify() {
        // TODO implement here
    }

}
```

15. Server

```
import java.util.*;

/**
 *
 */
public class Server {

    /**
     * Default constructor
     */
    public Server() {
    }

    /**
     *
     */
    public int Server ID;

    /**
     *
     */
    public String Server password;




    /**
     * @return
     */
    public Route compute_optimised_path() {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public Session create_session() {
        // TODO implement here
        return null;
    }

}
```

16. DB Server

```
import java.util.*;

/**
 *
 */
public class DB Server {

    /**
     * Default constructor
     */
    public DB Server() {
    }


    /**
     *
     */
    public void update() {
        // TODO implement here
    }

    /**
     *
     */
    public void insert() {
        // TODO implement here
    }

    /**
     *
     */
    public void select() {
        // TODO implement here
    }

    /**
     *
     */
    public void delete() {
        // TODO implement here
    }

}
```

17. Payment

```
import java.util.*;

/**
 *
 */
public class Payment {

    /**
     * Default constructor
     */
    public Payment() {
    }

    /**
     *
     */
    protected int Transaction ID;

    /**
     *
     */
    protected Boolean Payment Status;

    /**
     *
     */
    protected String Payment Mode;

    /**
     *
     */
    protected Invoice Invoice_obj;



    /**
     * @return
     */
    public String return_payment_details() {
        // TODO implement here
        return "";
    }

}
```

18. Invoice

```
import java.util.*;

/**
 *
 */
public class Invoice {

    /**
     * Default constructor
     */
    public Invoice() {
    }

    /**
     *
     */
    private int Invoice ID;

    /**
     *
     */
    private Date Invoice Date;

    /**
     *
     */
    private Time Invoice Time;

    /**
     *
     */
    private Double Amount Paid;


    /**
     * @return
     */
    public String return_invoice() {
        // TODO implement here
        return "";
    }

}
```

19. Cash Payment

```
import java.util.*;

/**
 *
 */
public class Cash Payment extends Payment {

    /**
     * Default constructor
     */
    public Cash Payment() {
    }

    /**
     *
     */
    public double Cash Rendered;

}
```

20. Credit Card Payment

```
import java.util.*;

/**
 *
 */
public class Creditcard Payment extends Payment implements Payment
Processor<<interface>> {

    /**
     * Default constructor
     */
    public Creditcard Payment() {
    }

    /**
     *
     */
    private String Name on card;

    /**
     *
```

```
     */
    private int Card Number;

    /**
     *
     */
    private int PIN;

    /**
     *
     */
    public void enter_card_details() {
        // TODO implement here
    }

    /**
     * @return
     */


}
```

21. UPI Payment

```
import java.util.*;

/**
 *
 */
public class UPI Payment extends Payment implements Payment
Processor<<interface>> {

    /**
     * Default constructor
     */
    public UPI Payment() {
    }

    /**
     *
     */
    private String UPI ID;

    /**
     *
     */
    private String UPI PIN;
```

```
    /**
     *
     */
    private String Bank Name;

    /**
     *
     */
    public void enter_UPI_details() {
        // TODO implement here
    }

    /**
     * @return
     */



}
```

Interfaces:

1) **Register**

```
public interface Login_logout {
    public void enter_login_details();
    public Boolean verify_login();
    public void logout();
}
```

2) **Login_logout**

```
public interface Register {
    public void enter_credentials();
}
```

3) **Profile_update**

```
public interface Profile_update{
    public void update_profile();
}
```

4) **Payment Processor**

```
public interface Payment_Processor {
    public Boolean pay();
}
```

APPLYING GRASP PATTERN FOR OUR PROJECT :

In order to achieve high cohesion (in Server class) and Polymorphism) , implementation of Payment processor interface is henceforth done by Credit Card payment class and UPI Payment class.

- **Polymorphism -** Credit Card Payment class and UPI Payment class extend Payment class and implement Payment Processor.
- **Indirection -** Server uses Message class for handling confirmation messages and alerts. DB Server class is used by other classes to handle DB operations.
- **Pure Fabrication -** DB Server is created to handle DB related operations (update, insert, delete and select).
- **Creator -** Payment class creates Invoice class (composition), Parking Slot creates Parking Details (composition), Driver creates Vehicle (records).
- **Information Expert -** Price List, Route, Parking Details, Session, Driver, Message, Invoice, Vehicle, Parking Slot.
- **Low Coupling** and **High Cohesion** has been implemented for all the classes.